Jin-Yi Cai
S. Barry Cooper
Hong Zhu (Eds.)

Theory and Applications of Models of Computation

LNCS 4484

4th International Conference, TAMC 2007
Shanghai, China, May 2007
Proceedings

Springer

Lecture Notes in Computer Science 4484

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Moshe Y. Vardi
Rice University, Houston, TX, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Jin-Yi Cai S. Barry Cooper
Hong Zhu (Eds.)

Theory and Applications of Models of Computation

4th International Conference, TAMC 2007
Shanghai, China, May 22-25, 2007
Proceedings

Springer

Volume Editors

Jin-Yi Cai
University of Wisconsin, Madison, Computer Sciences Department
1210 West Dayton Street, Madison, WI 53706, USA
E-mail: jyc@cs.wisc.edu

S. Barry Cooper
University of Leeds, School of Mathematics
Leeds LS2 9JT, UK
E-mail: s.b.cooper@leeds.ac.uk

Hong Zhu
Fudan University, Department of Computer Science and Engineering
220 Handan Road, Shanghai, 200433, China
E-mail: hzhu@fudan.edu.cn

Library of Congress Control Number: 2007926619

CR Subject Classification (1998): F.1.1-2, F.2.1-2, F.4.1, I.2.6, J.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-72503-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-72503-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12063888 06/3180 5 4 3 2 1 0

Preface

Theory and Applications of Models of Computation (TAMC) is an international conference series with an interdisciplinary character, bringing together researchers working in computer science, mathematics (especially logic) and the physical sciences. This cross-disciplinary character, together with its focus on algorithms, complexity and computability theory, gives the conference a special flavor and distinction.

TAMC 2007 was the fourth conference in the series. The previous three meetings were held May 17 – 19, 2004 in Beijing, May 17 – 20, 2005 in Kunming, and May 15 – 20, 2006 in Beijing, P. R. China. TAMC 2007 was held in Shanghai, May 22 – 25, 2007. Future annual meetings are planned.

At TAMC 2007 we were very pleased to have two plenary speakers, Miklós Ajtai and Fan Chung Graham, each gave a one hour invited talk. Miklós Ajtai spoke on "Generalizations of the Compactness Theorem and Godel's Completeness Theorem for Nonstandard Finite Structures," and Fan Chung Graham spoke on "Detecting Sharp Drops in PageRank and a Simplified Local Partitioning Algorithm." Their respective papers accompanying the talks are included in these proceedings.

In addition, there were two special sessions organized by Barry Cooper and Andrew Lewis on "Computability and Randomness" and by Manindra Agrawal and Angsheng Li on "Algorithms and Complexity." The invited speakers included George Barmpalias, Cristian Calude, Bjorn Kjos-Hanssen, Andre Nies, and Jan Reimann in the "Computability and Randomness" special session, and Manindra Agrawal, Alberto Apostolico, Jin-Yi Cai, Andreas Dress, Naveen Garg and Jaikumar Radhakrishnan in the "Algorithms and Complexity" special session.

The TAMC conference series arose naturally in response to important scientific developments affecting how we compute in the twenty-first century. At the same time, TAMC is already playing an important regional and international role, and promises to become a key contributor to the scientific resurgence seen throughout China and other parts of Asia.

The enthusiasm with which TAMC 2007 was received by the scientific community is evident in the large number of quality articles submitted to the conference. There were over 500 submissions, originating from all over the world. This presented the Program Committee with a major assessment task. The Program Committee finally selected 67 papers for presentation at the conference and inclusion in this LNCS volume. This results in an acceptance rate of just over 13%, making TAMC an extremely selective conference, compared to other leading international conferences.

We are very grateful to the Program Committee, and the many outside referees they called on, for the hard work and expertise which they brought to

the difficult selection process. We also wish to thank all those authors who submitted their work for our consideration. The Program Committee could have accepted many more submissions without compromising standards, and were only restrained by the practicalities of scheduling so many talks, and by the inevitable limitations on the size of this proceedings volume.

Finally, we would like to thank the members of the Editorial Board of *Lecture Notes in Computer Science* and the Editors at Springer for their encouragement and cooperation throughout the preparation of this conference.

Of course TAMC 2007 would not have been possible without the support of our sponsors, and we therefore gratefully acknowledge their help in the realization of this conference.

May 2007

Jin-Yi Cai
S. Barry Cooper
Hong Zhu

Organization

The conference was organized by: the Software School of Fudan University; University of Leeds, UK; and the University of Wisconsin–Madison, USA.

Conference Chair

Jin-Yi Cai (University of Wisconsin)

Program Committee Co-chairs

S. Barry Cooper (University of Leeds)
Hong Zhu (Fudan University)

Program Committee

Giorgio Ausiello (University of Rome)
Eric Bach (University of Wisconsin)
Nicolò Cesa-Bianchi (University of Milan)
Jianer Chen (Texas A&M University)
Yijia Chen (Shanghai Jiaotong University)
Francis Chin (University of Hong Kong)
C.T. Chong (National University of Singapore)
Kyung-Yong Chwa (KAIST, Korea)
Decheng Ding (Nanjing University)
Rodney Downey (University of Wellington)
Martin Dyer (University of Leeds)
Rudolf Fleischer (Fudan University)
Oscar Ibarra (UC Santa Barbara)
Hiroshi Imai (University of Tokyo)
Kazuo Iwama (Kyoto University)
Tao Jiang (University of California-Riverside/Tsinghua)
Satyanarayana Lokam (Microsoft Research-India)
D. T. Lee (Academia Sinica, Taipei)
Angsheng Li (Institute of Software, CAS, Beijing)
Giuseppe Longo (CNRS - Ecole Normale Superieure, Paris)
Tian Liu (Peking University)
Rüdiger Reischuk (Universität zu Lübeck)
Rocco Servedio (Columbia University)
Alexander Shen (Institute for Information Transmission Problems, Moscow)
Yaoyun Shi (University of Michigan)
Theodore A. Slaman (UC Berkeley)

Xiaoming Sun (Tsinghua University)
Shanghua Teng (Boston University)
Luca Trevisan (UC Berkeley)
Christopher Umans (Cal Tech)
Alasdair Urquhart (University of Toronto)
Hanpin Wang (Peking University)
Osamu Watanabe (Tokyo Institute of Technology)
Zhiwei Xu (Institute of Computing Technology, CAS, Beijing)
Frances Yao (City University of Hong Kong)
Mingsheng Ying (Tsinghua University)

Organizing Committee

Jin-Yi Cai (The University of Wisconsin-Madison, USA)
S. Barry Cooper (University of Leeds, UK)
Angsheng Li (Institute of Software, Chinese Academy of Sciences, China)
Hong Zhu (Fudan University, China)

Sponsoring Institutions

The National Natural Science Foundation of China
Chinese Academy of Sciences
Microsoft Research, Asia
Science and Technology Commission of Shanghai Municipality
Software School of Fudan University

Table of Contents

Plenary Lectures

Detecting Sharp Drops in PageRank and a Simplified Local Partitioning Algorithm 1
Reid Andersen and Fan Chung

Generalizations of the Compactness Theorem and Gödel's Completeness Theorem for Nonstandard Finite Structures 13
Miklós Ajtai

Contributed Papers

Approximation Algorithms for 3D Orthogonal Knapsack 34
Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas

A Comparative Study of Efficient Algorithms for Partitioning a Sequence into Monotone Subsequence 46
Bing Yang, Jing Chen, Enyue Lu, and S.Q. Zheng

The Hardness of Selective Network Design for Bottleneck Routing Games 58
Haiyang Hou and Guochuan Zhang

A Polynomial Time Algorithm for Finding Linear Interval Graph Patterns 67
Hitoshi Yamasaki and Takayoshi Shoudai

Elementary Differences Among Jump Hierarchies 79
Angsheng Li

Working with the LR Degrees 89
George Barmpalias, Andrew E.M. Lewis, and Mariya Soskova

Computability on Subsets of Locally Compact Spaces 100
Yatao Xu and Tanja Grubba

A New Approach to Graph Recognition and Applications to Distance-Hereditary Graphs 115
Shin-ichi Nakano, Ryuhei Uehara, and Takeaki Uno

Finding a Duplicate and a Missing Item in a Stream 128
Jun Tarui

Directed Searching Digraphs: Monotonicity and Complexity 136
Boting Yang and Yi Cao

Protecting Against Key Escrow and Key Exposure in Identity-Based Cryptosystem 148
Jin Wang, Xi Bai, Jia Yu, and Daxing Li

Encapsulated Scalar Multiplications and Line Functions in the Computation of Tate Pairing 159
Rongquan Feng and Hongfeng Wu

A Provably Secure Blind Signature Scheme 171
Xiaoming Hu and Shangteng Huang

Construct Public Key Encryption Scheme Using Ergodic Matrices over GF(2) 181
Pei Shi-Hui, Zhao Yong-Zhe, and Zhao Hong-Wei

New Left-to-Right Radix-r Signed-Digit Recoding Algorithm for Pairing-Based Cryptosystems 189
Fanyu Kong, Jia Yu, Zhun Cai, and Daxing Li

The Strongest Nonsplitting Theorem 199
Mariya Ivanova Soskova and S. Barry Cooper

There is an Sw-Cuppable Strongly c.e. Real 212
Yun Fan

On Computation Complexity of the Concurrently Enabled Transition Set Problem 222
Li Pan, Weidong Zhao, Zhicheng Wang, Gang Wei, and Shumei Wang

Synchronization of Some DFA 234
A.N. Trahtman

On the Treewidth and Pathwidth of Biconvex Bipartite Graphs 244
Sheng-Lung Peng and Yi-Chuan Yang

On Exact Complexity of Subgraph Homeomorphism 256
Andrzej Lingas and Martin Wahlen

Searching a Polygonal Region by Two Guards 262
Xuehou Tan

On the Internal Steiner Tree Problem 274
Sun-Yuan Hsieh, Huang-Ming Gao, and Shih-Cheng Yang

Approximately Optimal Trees for Group Key Management with Batch Updates 284
Minming Li, Ze Feng, Ronald L. Graham, and Frances F. Yao

On Deciding Deep Holes of Reed-Solomon Codes 296
Qi Cheng and Elizabeth Murray

Quantum Multiparty Communication Complexity and Circuit Lower Bounds 306
Iordanis Kerenidis

Efficient Computation of Algebraic Immunity of Symmetric Boolean Functions 318
Feng Liu and Keqin Feng

Improving the Average Delay of Sorting 330
Andreas Jakoby, Maciej Liśkiewicz, Rüdiger Reischuk, and Christian Schindelhauer

Approximating Capacitated Tree-Routings in Networks 342
Ehab Morsy and Hiroshi Nagamochi

Feedback Arc Set Problem in Bipartite Tournaments 354
Sushmita Gupta

Studying on Economic-Inspired Mechanisms for Routing and Forwarding in Wireless Ad Hoc Network 362
Yufeng Wang, Yoshiaki Hori, and Kouichi Sakurai

Enhancing Simulation for Checking Language Containment 374
Jin Yi and Wenhui Zhang

QBF-Based Symbolic Model Checking for Knowledge and Time 386
Conghua Zhou, Zhenyu Chen, and Zhihong Tao

A Characterization of the Language Classes Learnable with Correction Queries 398
Cristina Tîrnăucă and Satoshi Kobayashi

Learnable Algorithm on the Continuum 408
Zhimin Li and Xiang Li

Online Deadline Scheduling with Bounded Energy Efficiency 416
Joseph Wun-Tat Chan, Tak-Wah Lam, Kin-Sum Mak, and Prudence W.H. Wong

Efficient Algorithms for Airline Problem 428
Shin-ichi Nakano, Ryuhei Uehara, and Takeaki Uno

Efficient Exact Arithmetic over Constructive Reals 440
Yong Li and Jun-Hai Yong

Bounding Run-Times of Local Adiabatic Algorithms 450
M.V. Panduranga Rao

A Note on Universal Composable Zero Knowledge in Common Reference String Model 462
Andrew C.C. Yao, Frances F. Yao, and Yunlei Zhao

A Note on the Feasibility of Generalized Universal Composability 474
Andrew C.C. Yao, Frances F. Yao, and Yunlei Zhao

t-Private and Secure Auctions 486
Markus Hinkelmann, Andreas Jakoby, and Peer Stechert

Secure Multiparty Computations Using a Dial Lock 499
Takaaki Mizuki, Yoshinori Kugimoto, and Hideaki Sone

A Time Hierarchy Theorem for Nondeterministic Cellular Automata ... 511
Chuzo Iwamoto, Harumasa Yoneda, Kenichi Morita, and Katsunobu Imai

Decidability of Propositional Projection Temporal Logic with Infinite Models 521
Zhenhua Duan and Cong Tian

Separation of Data Via Concurrently Determined Discriminant Functions 533
Hong Seo Ryoo and Kwangsoo Kim

The Undecidability of the Generalized Collatz Problem 542
Stuart A. Kurtz and Janos Simon

Combinatorial and Spectral Aspects of Nearest Neighbor Graphs in Doubling Dimensional and Nearly-Euclidean Spaces 554
Yingchao Zhao and Shang-Hua Teng

Maximum Edge-Disjoint Paths Problem in Planar Graphs..... 566
Mingji Xia

An Efficient Algorithm for Generating Colored Outerplanar Graphs 573
Jiexun Wang, Liang Zhao, Hiroshi Nagamochi, and Tatsuya Akutsu

Orthogonal Drawings for Plane Graphs with Specified Face Areas 584
Akifumi Kawaguchi and Hiroshi Nagamochi

Absolutely Non-effective Predicates and Functions in Computable Analysis 595
Decheng Ding, Klaus Weihrauch, and Yongcheng Wu

Linear-Size Log-Depth Negation-Limited Inverter for k-Tonic Binary Sequences 605
Hiroki Morizumi and Jun Tarui

The Existence of Unsatisfiable Formulas in k-LCNF for $k \geq 3$ 616
Qingshun Zhang and Daoyun Xu

Improved Exponential Time Lower Bound of Knapsack Problem Under BT Model 624
Xin Li, Tian Liu, Han Peng, Liyan Qian, Hongtao Sun, Jin Xu, Ke Xu, and Jiaqi Zhu

Phase Transition of Multivariate Polynomial Systems 632
Giordano Fusco and Eric Bach

Approximation Algorithms for Maximum Edge Coloring Problem 646
Wangsen Feng, Li'ang Zhang, Wanling Qu, and Hanpin Wang

Two Improved Range-Efficient Algorithms for F_0 Estimation 659
He Sun and Chung Keung Poon

Approximation to the Minimum Rooted Star Cover Problem 670
Wenbo Zhao and Peng Zhang

Approximability and Parameterized Complexity of Consecutive Ones Submatrix Problems 680
Michael Dom, Jiong Guo, and Rolf Niedermeier

Parameterized Algorithms for Weighted Matching and Packing Problems 692
Yunlong Liu, Jianer Chen, and Jianxin Wang

Kernelizations for Parameterized Counting Problems 703
Marc Thurley

Revisiting the Impossibility for Boosting Service Resilience 715
Xingwu Liu, Zhiwei Xu, and Juhua Pu

An Approximation Algorithm to the k-Steiner Forest Problem 728
Peng Zhang

A Distributed Algorithm of Fault Recovery for Stateful Failover 738
Indranil Saha and Debapriyay Mukhopadhyay

Path Embedding on Folded Hypercubes 750
Sun-Yuan Hsieh

An Approximation Algorithm Based on Chain Implication for Constrained Minimum Vertex Covers in Bipartite Graphs 760
Jianxin Wang, Xiaoshuang Xu, and Jianer Chen

Author Index 771

Detecting Sharp Drops in PageRank and a Simplified Local Partitioning Algorithm

Reid Andersen and Fan Chung

University of California at San Diego, La Jolla CA 92093, USA
fan@ucsd.edu, http://www.math.ucsd.edu/~fan/
randerse@ucsd.edu, http://www.math.ucsd.edu/~randerse/

Abstract. We show that whenever there is a sharp drop in the numerical rank defined by a personalized PageRank vector, the location of the drop reveals a cut with small conductance. We then show that for any cut in the graph, and for many starting vertices within that cut, an approximate personalized PageRank vector will have a sharp drop sufficient to produce a cut with conductance nearly as small as the original cut. Using this technique, we produce a nearly linear time local partitioning algorithm whose analysis is simpler than previous algorithms.

1 Introduction

When we are dealing with computational problems arising in complex networks with prohibitively large sizes, it is often desirable to perform computations whose cost can be bounded by a function of the size of their output, which may be quite small in comparison with the size of the whole graph. Such algorithms we call *local algorithms* (see [1]). For example, a local graph partitioning algorithm finds a cut near a specified starting vertex, with a running time that depends on the size of the small side of the cut, rather than the size of the input graph.

The first local graph partitioning algorithm was developed by Spielman and Teng [8], and produces a cut by computing a sequence of truncated random walk vectors. A more recent local partitioning algorithm achieves a better running time and approximation ratio by computing a single personalized PageRank vector [1]. Because a PageRank vector is defined recursively (as we will describe in the next section), a sweep over a single approximate PageRank vector can produce cuts with provably small conductance. Although this use of PageRank simplified the process of finding cuts, the analysis required to extend the basic cut-finding method into an efficient local partitioning algorithm remained complicated.

In this paper, we consider the following consequence of the personalized PageRank equation,

$$p = \alpha v + (1 - \alpha)pW,$$

where p is taken to be a row vector, v is the indicator vector for a single vertex v, and W is the probability transition matrix of a random walk on the graph (this

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 1–12, 2007.
© Springer-Verlag Berlin Heidelberg 2007

will be defined in more detail later). When a random walk step is applied to the personalized PageRank vector p, every vertex in the graph has more probability from pW than it has from p, except for the seed vertex v. This implies something strong about the ordering of the vertices produced by the PageRank vector p: there cannot be many links between any set of vertices with high probability in p and any set of vertices with low probability in p. More precisely, whenever there is a *sharp drop* in probability, where the kth highest ranked vertex has much more probability than the $k(1+\delta)$th vertex, there must be few links between the k highest ranked vertices and the vertices not ranked in the top $k(1+\delta)$.

We will make this observation rigorous in Lemma 1, which provides an intuitive proof that personalized PageRank identifies a set with small conductance. In the section that follows, we will prove a series of lemmas that describe necessary conditions for a sharp drop to exist. In the final section, we will use these techniques to produce an efficient local partitioning algorithm, which finds cuts in nearly linear time by detecting sharp drops in approximate PageRank vectors.

2 Preliminaries

PageRank was introduced by Brin and Page [3,7]. The PageRank vector $\mathrm{pr}(\alpha, s)$ is defined to be the unique solution of the equation

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, s)W, \tag{1}$$

where α is a constant in $(0,1)$ called the *teleportation constant* , s is a vector called the *starting vector*, and W is the random walk transition matrix $W = D^{-1}A$. Here D denotes the diagonal matrix whose diagonal entries are the degrees of the vertices, and A denotes the adjacency matrix of the graph.

The PageRank vector that is usually associated with search ranking has a starting vector equal to the uniform distribution $\frac{1}{n}\mathbf{1}$. PageRank vectors whose starting vectors are concentrated on a smaller set of vertices are often called *personalized PageRank vectors*. These were introduced by Haveliwala [5], and have been used to provide personalized search ranking and context-sensitive search [2,4,6]. We will consider PageRank vectors whose starting vectors are equal to the indicator function 1_v for a single vertex v. The vertex v will be called the *starting vertex*, and we will use the notation $\mathrm{pr}(\alpha, v) = \mathrm{pr}(\alpha, 1_v)$.

The *volume* of a subset $S \subseteq V$ of vertices is $\mathrm{Vol}(S) = \sum_{x \in S} d(x)$. We remark that $\mathrm{Vol}(V) = 2m$, and we will sometimes write $\mathrm{Vol}(G)$ in place of $\mathrm{Vol}(V)$. We write $e(S,T)$ to denote the number of edges between two disjoint sets of vertices S and T. The *conductance* of a set is

$$\Phi(S) = \frac{e(S, \bar{S})}{\min\left(\mathrm{Vol}(S), 2m - \mathrm{Vol}(S)\right)}.$$

The amount of probability from a vector p on a set S of vertices is written $p(S) = \sum_{x \in S} p(x)$. We will sometimes refer to the quantity $p(S)$ as an amount of probability even if $p(V)$ is not equal to 1. As an example of this notation, the amount of probability from the PageRank vector $\mathrm{pr}(\alpha, v)$ on a set S will be written $\mathrm{pr}(\alpha, \chi_v)(S)$. The support of a vector is the set of vertices on which it is nonzero, $\mathrm{Support}(p) = \{v \mid p(v) \neq 0\}$.

2.1 Approximate Personalized PageRank Vectors

Here are some useful properties of PageRank vectors (also see [5] and [6]).

Proposition 1. *For any starting vector s, and any constant α in $(0, 1]$, there is a unique vector $\mathrm{pr}(\alpha, s)$ satisfying $\mathrm{pr}(\alpha, s) = \alpha s + (1 - \alpha)\mathrm{pr}(\alpha, s)W$.*

Proposition 2. *For any fixed value of α in $(0, 1]$, there is a linear transformation R_α such that $\mathrm{pr}(\alpha, s) = sR_\alpha$. Furthermore, R_α is given by the matrix*

$$R_\alpha = \alpha I + \alpha \sum_{t=1}^{\infty} (1 - \alpha)^t W^t. \tag{2}$$

This implies that a PageRank vector $\mathrm{pr}(\alpha, s)$ is linear in its starting vector s.

Instead of computing the PageRank vector $\mathrm{pr}(\alpha, v)$ exactly, we will approximate it by another PageRank vector $\mathrm{pr}(\alpha, v - r)$ with a slightly different starting vector, where r is a vector with nonnegative entries. If $r(v) \leq \epsilon d(v)$ for every vertex in the graph, then we say $\mathrm{pr}(\alpha, v - r)$ is an ϵ-approximate PageRank vector for $\mathrm{pr}(\alpha, v)$.

Definition 1. *An* ϵ-approximate *PageRank vector for $\mathrm{pr}(\alpha, v)$ is a PageRank vector $\mathrm{pr}(\alpha, v - r)$ where the vector r is nonnegative and satisfies $r(u) \leq \epsilon d(u)$ for every vertex u in the graph.*

We will use the algorithm $\texttt{ApproximatePR}(v, \alpha, \epsilon)$ described in the following theorem to compute ϵ-approximate PageRank vectors with small support. The running time of the algorithm depends on ϵ and α, but is independent of the size of the graph.

Theorem 1. *For any vertex v, any constant $\alpha \in (0, 1]$, and any constant $\epsilon \in (0, 1]$, The algorithm $\texttt{ApproximatePR}(v, \alpha, \epsilon)$ computes an ϵ-approximate PageRank vector $p = \mathrm{pr}(\alpha, v - r)$ with support $\mathrm{Vol}(\mathrm{Support}(p)) \leq \frac{2}{(1-\alpha)\epsilon}$. The running time of the algorithm is $O(\frac{1}{\epsilon\alpha})$.*

The proof of this theorem, and the description of the algorithm, were given in [1]. We will use the algorithm as a black box.

3 A Sharp Drop in PageRank Implies a Good Cut

We describe a normalized rank function derived from a PageRank vector, and the ordering of the vertices induced by that rank function.

Definition 2. *Given a PageRank vector p, we define the following.*

- *Define the rank function q to be $q(u) = p(u)/d(u)$.*
- *Let π be a permutation that places the vertices in nonincreasing order of rank, so that*
$$q(\pi(1)) \geq q(\pi(2)) \geq \cdots \geq q(\pi(n)).$$
This is the ordering induced by the PageRank vector. An integer $j \in [1, n]$ will be called an index, and we will say that $\pi(j)$ is the vertex at index j.
- *Let $S_j = \{\pi(1), \ldots, \pi(j)\}$ be the set containing the j vertices of highest rank. The set S_j is called the jth level set of the PageRank vector.*
- *Define the shorthand notation $q(j) = q(\pi(j))$ and $V(j) = \mathrm{Vol}(S_j)$.*

We now prove the main lemma. If there is a sharp drop in rank at S_j, then the set S_j has small conductance. We will prove the contrapositive instead, because that is how we will eventually apply the lemma. Namely, we will prove that either the set S_j has small conductance, or else there is an index $k > j$ where the volume $\mathrm{Vol}(S_k)$ is significantly larger than $\mathrm{Vol}(S_j)$, and the rank $q(k)$ is not much smaller than $q(j)$.

Lemma 1 (Sharp Drop Lemma). *Let $p = \mathrm{pr}(\alpha, v - r)$ be an approximate PageRank vector. Let $\phi \in (0, 1)$ be a real number, and let j be any index in $[1, n]$. Either the number of edges leaving S_j satisfies $e(S_j, \bar{S}_j) < 2\phi\mathrm{Vol}(S_j)$, or else there is some index $k > j$ such that*

$$\mathrm{Vol}(S_k) \geq \mathrm{Vol}(S_j)(1 + \phi) \qquad \textit{and} \qquad q(k) \geq q(j) - \alpha/\phi\mathrm{Vol}(S_j).$$

Proof. For any set S of vertices,

$$pW(S) = p(S) - \sum_{(u,v)\in e(S,\bar{S})} q(u) - q(v).$$

Since $p = \mathrm{pr}(\alpha, v - r)$ and the vector r is nonnegative,

$$pW = (1 - \alpha)^{-1}(p - \alpha(v - r)) \geq p - \alpha v.$$

The two equations above imply that

$$\sum_{(u,v)\in e(S,\bar{S})} q(u) - q(v) \leq \alpha. \tag{3}$$

Now consider the level set S_j. If $\mathrm{Vol}(S_j)(1 + \phi) > \mathrm{Vol}(G)$, then

$$e(S_j \bar{S}_j) \leq \mathrm{Vol}(G)(1 - \frac{1}{1+\phi}) \leq \phi \mathrm{Vol}(S_j),$$

and the theorem holds trivially. If $\mathrm{Vol}(S_j)(1+\phi) \leq \mathrm{Vol}(G)$, then there is a unique index k such that

$$\mathrm{Vol}(S_{k-1}) \leq \mathrm{Vol}(S_j)(1+\phi) \leq \mathrm{Vol}(S_k).$$

If $e(S_j, \bar{S}_j) < 2\phi \mathrm{Vol}(S_j)$, we are done. If $e(S_j, \bar{S}_j) \geq 2\phi \mathrm{Vol}(S_j)$, then $e(S_j, \bar{S}_{k-1})$ is also large,

$$e(S_j, \bar{S}_{k-1}) \geq \partial(S_j) - \mathrm{Vol}(S_{k-1} \setminus S_j) \geq 2\phi \mathrm{Vol}(S_j) - \phi \mathrm{Vol}(S_j) = \phi \mathrm{Vol}(S_j).$$

Using equation (3),

$$\begin{aligned} \alpha \geq \sum_{(u,v) \in e(S_j, \bar{S}_j)} q(u) - q(v) &\geq \sum_{(u,v) \in e(S_j, \bar{S}_{k-1})} q(u) - q(v) \\ &\geq e(S_j, \bar{S}_{k-1})(q(j) - q(k)) \\ &\geq \phi \mathrm{Vol}(S_j) \cdot (q(j) - q(k)). \end{aligned}$$

This shows that $q(j) - q(k) \leq \alpha / \phi \mathrm{Vol}(S_j)$, completing the proof.

4 Ensuring That a Sharp Drop Exists

In this section, we will introduce several tools that will allow us to show that a sharp drop in rank exists for many personalized PageRank vectors. When we present the local partitioning algorithm in the next section, these tools will be used to prove its approximation guarantee.

Throughout this section and the next, we have two PageRank vectors to consider, the PageRank vector $p = \mathrm{pr}(\alpha, v)$, and the approximate PageRank vector $\tilde{p} = \mathrm{pr}(\alpha, v - r)$ that will be computed by the local partitioning algorithm. These two PageRank vectors induce two different orderings π and $\tilde{\pi}$, which lead to two different rank functions $q(k)$ and $\tilde{q}(k)$, which produce two collections of level sets S_k and $\tilde{S}_k$, which have different volumes $V(k) = \mathrm{Vol}(S_k)$ and $\tilde{V}(k) = \mathrm{Vol}(\tilde{S}_k)$.

We start by showing there is some index i where the rank $q(i)$ is not much smaller than $1/V(i)$. This lemma doesn't use any special properties of PageRank vectors, and is true for any nonnegative vector whose entries sum to 1.

Lemma 2 (Integration Lemma). *Let q be the rank function of any vector p for which $\|p\|_1 = 1$. Then, there exists an index i such that $q(i) \geq \frac{1}{H(2m)V(i)}$, where $H(2m) = \sum_{k=1}^{2m} 1/k = O(\log m)$.*

Proof. If we assume that $q(i) < c/V(i)$ for all $i \in [1, n]$, then

$$\begin{aligned}\sum_{i=1}^{n} q(i)d(i) &< c\sum_{i=1}^{n} \frac{d(i)}{V(i)} \\ &\leq c\sum_{k=1}^{2m} \frac{1}{k} \\ &= cH(2m).\end{aligned}$$

If $c = 1/H(2m)$, this would imply $\|p\|_1 = \sum_{i=1}^{n} q(i)d(i) < 1$, so we must have $q(i) \geq \frac{1}{H(2m)V(i)}$ for some index i.

We now give a lower bound on the rank function of an ϵ-approximate PageRank vector $\tilde{p} = \mathrm{pr}(\alpha, v-r)$ that depends on the rank function of the PageRank vector $p = \mathrm{pr}(\alpha, v)$ that is being approximated, and on the error parameter ϵ.

Lemma 3 (Approximation Error Lemma). *Let q be the rank function for a PageRank vector $p = \mathrm{pr}(\alpha, v)$, and let $\tilde{q}$ be the rank for an ϵ-approximate PageRank vector $\tilde{p} = \mathrm{pr}(\alpha, v - r)$. For any index i, there is an index j such that*

$$\tilde{q}(j) \geq q(i) - \epsilon \qquad \textit{and} \qquad \mathrm{Vol}(\tilde{S}_j) \geq \mathrm{Vol}(S_i).$$

Proof. If $v \in S_i$, then $p(v)/d(v) \geq q(i)$. Since $\tilde{p}$ is an ϵ-approximation of p,

$$\tilde{p}(v)/d(v) \geq p(v)/d(v) - \epsilon \geq q(i) - \epsilon.$$

Therefore, the set of vertices for which $\tilde{p}(v)/d(v) \geq q(i) - \epsilon$ has volume at least $\mathrm{Vol}(S_j)$, which proves the lemma.

The following lemma shows what happens when you repeatedly apply the Sharp Drop Lemma, but fail to find a cut with small conductance. You will find a sequence of larger and larger indices for which the rank does not drop very quickly. We give a lower bound on the rank of the final index in the sequence. We will eventually contradict this lower bound, which will show that one of the applications of the Sharp Drop Lemma finds a cut with small conductance.

Lemma 4 (Chaining Lemma). *Let $\{k_0 \ldots k_f\}$ be an increasing sequence of indices such that for each $i \in [0, f-1]$, the following holds.*

$$q(k_{i+1}) \geq q(k_i) - \alpha/\phi V(k_i) \qquad \textit{and} \qquad V(k_{i+1}) \geq (1+\phi)V(k_i).$$

Then, the last index k_f satisfies

$$q(k_f) \geq q(k_0) - 2\alpha/\phi^2 V(k_0).$$

Proof. The bound on the change in volume implies that $V(k_i) \geq (1+\phi)^i V(k_0)$ for all $i \in [0, f-1]$. Therefore,

$$\begin{aligned} q(k_f) &\geq q(k_0) - \frac{\alpha}{\phi V(k_0)} - \frac{\alpha}{\phi V(k_1)} - \cdots - \frac{\alpha}{\phi V(k_{f-1})} \\ &\geq q(k_0) - \frac{\alpha}{\phi V(k_0)} \left(1 - \frac{1}{(1+\phi)} - \cdots - \frac{1}{(1+\phi)^{f-1}}\right) \\ &\geq q(k_0) - \frac{\alpha}{\phi V(k_0)} (\frac{1+\phi}{\phi}) \\ &\geq q(k_0) - \frac{2\alpha}{\phi^2 V(k_0)}. \end{aligned}$$

This completes the proof of the Chaining Lemma.

To contradict the lower bound from the Chaining Lemma, we will place a lower bound on $\mathrm{pr}(\alpha, v)(C)$, that depends on the conductance of C. This bound will apply to many starting vertices v within C.

Definition 3. *Given a set C, let C_α be the set of vertices v within C such that $\mathrm{pr}(\alpha, v)(\bar{C})$ is at most $2\Phi(C)/\alpha$.*

Lemma 5 (Probability Capturing Lemma). *For any set C and value of α, we have $\mathrm{Vol}(C_\alpha) \geq (1/2)\mathrm{Vol}(C)$.*

Proof. Let π_C be the probability distribution obtained by sampling a vertex v from C with probability $d(v)/\mathrm{Vol}(C)$. It is not difficult to verify the following statement, which was proved in [1].

$$\mathrm{pr}(\alpha, \pi_C) W(\bar{C}) \leq \mathrm{pr}(\alpha, \pi_C)(\bar{C}) + \Phi(C).$$

We will apply this observation to the personalized PageRank equation.

$$\begin{aligned} \mathrm{pr}(\alpha, \pi_C)(\bar{C}) &= [\alpha \pi_C + (1-\alpha)\mathrm{pr}(\alpha, \pi_C) W](\bar{C}) \\ &= (1-\alpha)[\mathrm{pr}(\alpha, \pi_C) W](\bar{C}) \\ &\leq (1-\alpha)\mathrm{pr}(\alpha, \pi_C)(\bar{C}) + \Phi(C). \end{aligned}$$

This implies

$$\mathrm{pr}(\alpha, \pi_C)(\bar{C}) \leq \Phi(C)/\alpha.$$

If we sample a vertex v from the distribution π_C, then at least half of the time $\mathrm{pr}(\alpha, v)(\bar{C})$ is at most twice its expected value of $\mathrm{pr}(\alpha, \pi_C)(\bar{C})$, and hence at least half the time v is in C_α. This implies that the volume of the set C_α is at least half the volume of C.

5 Local Partitioning Algorithm

The local partitioning algorithm can be described as follows:

`Local Partition`(v, ϕ, x):
The input to the algorithm is a starting vertex v, a target conductance $\phi \in (0, 1/3)$, and a target volume $x \in [0, 2m]$.

PageRank computation:

1. Let $\gamma = H(2m)$, let $\alpha = \frac{\phi^2}{8\gamma}$, and let $\epsilon = \frac{1}{2\gamma x}$.
2. Compute an ϵ-approximate PageRank vector $\tilde{p} = \mathrm{pr}(\alpha, v - r)$, using `ApproximatePR`$(v, \alpha, \epsilon)$.
3. Order the vertices so that $\tilde{q}(1) \geq \tilde{q}(2) \geq \cdots \geq \tilde{q}(n)$.

Finding a cut:
The algorithm will now examine a sequence of vertices, looking for a sharp drop. We let $\{k_i\}$ be the indices of the vertices examined by the algorithm. The first index examined by the algorithm will be k_0, and the last index examined will be k_f. We will now describe how these indices are chosen.

1. Let the starting index k_0 be the largest index such that $\tilde{q}(k_0) \geq 1/2\gamma\tilde{V}(k_0)$. If no such index exists, halt and output FAIL: NO STARTING INDEX.
2. While the algorithm is still running:
   (a) If $(1 + \phi)\tilde{V}(k_i) > \mathrm{Vol}(G)$ or if $\tilde{V}(k_i) > \mathrm{Vol}(\mathrm{Support}(\tilde{p}))$, then let $k_f = k_i$, output FAIL: NO CUT FOUND and quit.
   (b) Otherwise, let k_{i+1} be the smallest index such that $\tilde{V}(k_{i+1}) \geq \tilde{V}(k_i)(1 + \phi)$.
   (c) If $\tilde{q}(k_{i+1}) \leq \tilde{q}(k_i) - \alpha/\phi\tilde{V}(k_i)$, then let $k_f = k_i$, output the set S_{k_i}, and quit. Otherwise, repeat the loop.

Remarks:
When we analyze the algorithm, we will need the following observations. Regardless of whether the algorithm fails or successfully outputs a cut, the sequence of indices $\{k_0, \ldots, k_f\}$ examined by the algorithm satisfies the conditions of Lemma 4. If the algorithm fails during the loop of step 2, then k_f satisfies either $(1 + \phi)\tilde{V}(k_f) > \mathrm{Vol}(G)$ or $\tilde{V}(k_f) > \mathrm{Vol}(\mathrm{Support}(\tilde{p}))$.

Theorem 2. *The running time of* `Local Partition`(v, ϕ, x) *is* $O(x\frac{\log^2 m}{\phi^2})$.

Proof. The running time of the algorithm is dominated by the time to compute and sort $\tilde{p}$. Computing $\tilde{p}$ can be done in time $O(1/\epsilon\alpha) = O(x\gamma/\alpha) = O(x\frac{\log m}{\alpha})$

using `ApproximatePR`. The support of this vector has volume $O(1/\epsilon) = O(\gamma x) = O(x \log m)$, so the time required to sort $\tilde{p}$ is

$$O(|\text{Support}(\tilde{p})| \log |\text{Support}(\tilde{p})|) = O(x \log^2 m).$$

Since we have set $\alpha = \Omega(\phi^2 / \log m)$, the total running time is

$$O(x \frac{\log m}{\alpha} + x \log^2 m) = O(x \frac{\log^2 m}{\phi^2}).$$

Theorem 3. *Consider a run of the algorithm* `Local Partition` *on the input values v, ϕ, and x. Let $\tilde{p} = \text{pr}(\alpha, v - r)$ be the ϵ-approximate PageRank vector computed by the algorithm. Note that $\alpha = \phi^2/8\gamma$ and $\epsilon = 1/2\gamma x$. The following statements are true.*

1. *Let q be the rank function of the PageRank vector $p = \text{pr}(\alpha, v)$. There exists an index K such that $q(K) \geq 1/\gamma \text{Vol}(S_K)$. Furthermore, if the target volume x satisfies $x \geq \text{Vol}(S_K)$, then the algorithm finds a starting index k_0 that satisfies $\text{Vol}(\tilde{S}_{k_0}) \geq \text{Vol}(S_K)$.*
2. *Assume there exists a set C whose volume satisfies $\text{Vol}(C) \leq \frac{1}{2}\text{Vol}(G)$, whose conductance satisfies $\Phi(C) \leq \alpha/80\gamma$, and for which the starting vertex v is contained in C_α. If the target volume x satisfies $x \geq \text{Vol}(S_K)$, then the algorithm successfully outputs a set. Furthermore, the set S output by the algorithm has the following properties.*
   (a) (Approximation guarantee) $\Phi(S) \leq 3\phi = 3\sqrt{8\gamma\alpha}$.
   (b) (Volume lower bound) $\text{Vol}(S) \geq \text{Vol}(S_K)$.
   (c) (Volume upper bound) $\text{Vol}(S) \leq (5/9)\text{Vol}(G)$.
   (d) (Intersection with C) $\text{Vol}(S \cap C) \geq (9/10)\text{Vol}(S)$.

Proof. We begin by considering the PageRank vector $p = \text{pr}(\alpha, v)$ in order to prove claim 1 of the theorem. Lemma 2 shows that there is some index K for which $q(K) \geq 1/\gamma \text{Vol}(S_K)$, which proves part of the claim. To prove the other part, we assume that $x \geq \text{Vol}(S_K)$, and show that the algorithm finds a starting index k_0 that satisfies $\text{Vol}(\tilde{S}_{k_0}) \geq \text{Vol}(S_K)$. Lemma 3 shows that there exists an index j such that $\text{Vol}(\tilde{S}_j) \geq \text{Vol}(S_K)$ and

$$\tilde{q}(j) \geq q(K) - \epsilon \geq \frac{1}{\gamma \text{Vol}(S_K)} - \epsilon.$$

Since $x \geq \text{Vol}(S_K)$, we have $\epsilon = 1/2\gamma x \leq 1/2\gamma \text{Vol}(S_K)$, which implies the following.

$$\tilde{q}(j) \geq \frac{1}{\gamma \text{Vol}(S_K)} - \frac{1}{2\gamma x} \geq \frac{1}{2\gamma \text{Vol}(S_K)} \geq \frac{1}{2\gamma \text{Vol}(\tilde{S}_j)}.$$

This shows that j may be chosen as a starting index, so the algorithm is assured of choosing some starting index $k_0 \geq j$, which we know will satisfy

$$\text{Vol}(\tilde{S}_{k_0}) \geq \text{Vol}(S_K) \qquad \text{and} \qquad \tilde{q}(k_0) \geq \frac{1}{2\gamma \text{Vol}(\tilde{S}_{k_0})}.$$

This proves Claim 1 of the theorem.

We now move on to the proof of Claim 2. Let k_f be the index of the last vertex considered by the algorithm. We will give a lower bound on $\tilde{q}(k_f)$. Because the rank $\tilde{q}(k_{i+1})$ is not much smaller than $\tilde{q}(k_i)$ at each step, Lemma 4 shows that

$$\tilde{q}(k_f) \geq \tilde{q}(k_0) - \frac{(2\alpha/\phi^2)}{\mathrm{Vol}(\tilde{S}_{k_0})}.$$

We have set $\alpha = \phi^2/8\gamma$ to ensure that $2\alpha/\phi^2 \leq 1/4\gamma$, and so

$$\begin{aligned}\tilde{q}(k_f) &\geq \tilde{q}(k_0) - \frac{(2\alpha/\phi^2)}{\mathrm{Vol}(\tilde{S}_{k_0})} \\ &\geq \frac{1}{2\gamma \mathrm{Vol}(\tilde{S}_{k_0})} - \frac{1}{4\gamma \mathrm{Vol}(\tilde{S}_{k_0})} \\ &\geq \frac{1}{4\gamma \mathrm{Vol}(\tilde{S}_{k_0})}.\end{aligned}$$

We now use the assumptions that $v \in C_\alpha$ and $\Phi(C) \leq \alpha/80\gamma$, and apply Lemma 5 to give the following bound on $\tilde{p}(\bar{C})$.

$$\tilde{p}(\bar{C}) \leq p(\bar{C}) \leq 2\Phi(C)/\alpha \leq 1/40\gamma.$$

Combining our lower bound on $\tilde{q}(k_f)$ with our upper bound on $\tilde{p}(\bar{C})$ gives the following bound on the intersection of $\tilde{S}_{k_f}$ with $\bar{C}$.

$$\begin{aligned}\mathrm{Vol}(\tilde{S}_{k_f} \cap \bar{C}) &\leq \frac{\tilde{p}(\bar{C})}{\tilde{q}(k_f)} \\ &\leq \frac{4\gamma \mathrm{Vol}(\tilde{S}_{k_0})}{40\gamma} \\ &\leq \frac{1}{10}\mathrm{Vol}(\tilde{S}_{k_f}).\end{aligned}$$

This implies

$$\mathrm{Vol}(\tilde{S}_{k_f}) \leq \mathrm{Vol}(C) + \mathrm{Vol}(\tilde{S}_{k_f} \cap \bar{C}) \leq \mathrm{Vol}(C) + \frac{1}{10}\mathrm{Vol}(\tilde{S}_{k_f}).$$

We now use the fact that $\mathrm{Vol}(C) \leq (1/2)\mathrm{Vol}(G)$,

$$\mathrm{Vol}(\tilde{S}_{k_f}) \leq (10/9)\mathrm{Vol}(C) \leq (5/9)\mathrm{Vol}(G) \leq \frac{\mathrm{Vol}(G)}{1+\phi}.$$

The last step follows by assuming that $\phi \leq 1/3$. We can do so without loss of generality because the approximation guarantee of the theorem is vacuous if $\phi \geq 1/3$.

The equation above shows that the algorithm will not experience a failure caused by $(1+\phi)\mathrm{Vol}(\tilde{S}_{k_f}) > \mathrm{Vol}(G)$, and our lower bound on $\tilde{q}(k_f)$ ensures that the algorithm will not experience a failure caused by $\mathrm{Vol}(\tilde{S}_{k_f}) > \mathrm{Vol}(\mathrm{Support}(\tilde{p}))$. This ensures that the algorithm does not fail and output FAIL: NO CUT FOUND. We have already ensured that the algorithm does not fail and output FAIL: NO STARTING INDEX. Since we have ruled out all of the possible failure conditions, the algorithm must successfully output a set.

We must still prove that the set output by the algorithm satisfies all the properties in claim 2. We have already proved that $\mathrm{Vol}(\tilde{S}_{k_f}) \leq (5/9)\mathrm{Vol}(G)$, which proves claim 2(b). We have proved $\mathrm{Vol}(\tilde{S}_{k_f} \cap \bar{C}) \leq \frac{1}{10}\mathrm{Vol}(\tilde{S}_{k_f})$, which proves claim (2d). We have proved $\mathrm{Vol}(\tilde{S}_{k_f}) \geq \mathrm{Vol}(\tilde{S}_{k_0}) \geq \mathrm{Vol}(S_K)$, which proves claim (2c).

For the coup de grâce, we will apply the Sharp Drop Lemma. Since the set $\tilde{S}_{k_f}$ was output by the algorithm, that set must have the following property: if k_{f+1} is the smallest index such that $\mathrm{Vol}(\tilde{S}_{k_{f+1}}) \geq \mathrm{Vol}(\tilde{S}_{k_f})(1+\phi)$, then $\tilde{q}(k_{f+1}) < \tilde{q}(k_f) - \alpha/\phi\mathrm{Vol}(\tilde{S}_{k_f})$. We can then apply the Sharp Drop Lemma to show that the number of edges leaving S_{k_f} satisfies $e(\tilde{S}_{k_f}, \bar{\tilde{S}}_{k_f}) < 2\phi\mathrm{Vol}(\tilde{S}_{k_f})$. Since $\mathrm{Vol}(\tilde{S}_{k_f}) \leq \frac{5}{9}\mathrm{Vol}(G)$, we have $\mathrm{Vol}(G) - \mathrm{Vol}(\tilde{S}_{k_f}) \geq \frac{4}{9}\mathrm{Vol}(G) \geq \frac{4}{5}\mathrm{Vol}(\tilde{S}_{k_f})$, and so

$$\begin{aligned}
\Phi(\tilde{S}_{k_f}) &= \frac{e(\tilde{S}_{k_f}, \bar{\tilde{S}}_{k_f})}{\min(\mathrm{Vol}(\tilde{S}_{k_f}), \mathrm{Vol}(G) - \mathrm{Vol}(\tilde{S}_{k_f}))} \\
&\leq \frac{2\phi\mathrm{Vol}(\tilde{S}_{k_f})}{(4/5)\mathrm{Vol}(\tilde{S}_{k_f})} \\
&\leq 3\phi.
\end{aligned}$$

This proves the approximation guarantee of claim (2a).

References

1. R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *Proc. 47th Annual Symposium on Foundations of Computer Science*, (2006).
2. P. Berkhin, Bookmark-Coloring Approach to Personalized PageRank Computing, *Internet Mathematics*, to appear.
3. S. Brin and L. Page, The anatomy of a large-scale hypertextual Web search engine, *Computer Networks and ISDN Systems*, (1998), 107–117.
4. D. Fogaras and B. Racz, Towards scaling fully personalized PageRank, *Proceedings of the 3rd Workshop on Algorithms and Models for the Web-Graph (WAW)*, (2004), 105–117.
5. T. H. Haveliwala, Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search, *IEEE Trans. Knowl. Data Eng.*, (2003), 784-796.

6. G. Jeh and J. Widom, Scaling personalized web search, *Proceedings of the 12th World Wide Web Conference (WWW)*, (2003), 271-279.
7. L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, *Stanford Digital Library Technologies Project*, 1998.
8. D. A. Spielman and S.-H. Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, *ACM STOC* (2004), 81–90.

Generalizations of the Compactness Theorem and Gödel's Completeness Theorem for Nonstandard Finite Structures

Miklós Ajtai

IBM Research, Almaden Research Center

Abstract. The compactness theorem and Gödel's completeness theorem are perhaps the most important tools of mathematical logic for creating extensions of an existing model of a given theory. Unfortunately none of these theorems hold if we restrict our attention to finite models. In this paper we give generalizations of these theorems which can be used to construct extensions of nonstandard versions of finite structures. Therefore, although the structures are infinite, some finiteness properties will be true both for the original and the extended structures. These types of model extensions are closely related to questions in complexity theory.

1 Introduction

1.1 The Formulation of the Results

The compactness theorem and Gödel's completeness theorems are fundamental tools of mathematical logic. Unfortunately if we restrict our attention to finite models (which is the natural choice if we study algorithmic questions), then these theorems are not valid any more, and they do not have natural analogues.

In this paper we describe generalizations (for the compactness and completeness theorems) that are, in some sense, applicable for the finite case as well. We do not deal directly with finite structures. The new generalized theorems hold for infinite structures that are finite in a nonstandard model of arithmetic (or equivalently finite set theory). Using specific properties of these structures, e.g. that the Axiom-scheme of Induction holds in them, which reflect the fact that they are "finite" in some sense, the new generalized theorems make possible to construct new models in a way that are not possible with the original versions of the same theorems.

We will formulate an analogue of the compactness theorem and describe its proof in detail. We also state the analogue of Gödel's completeness theorem but describe only the basic ideas of the proof. These two new theorems, at least in their present formulations, are not equivalent, although their proofs have a common core.

Our basic setup is the following. Suppose that we have a nonstandard model $\mathbf{M}$ of Peano Arithmetic and $\mathcal{M}$ is a structure whose universe consists of the

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 13–33, 2007.
© Springer-Verlag Berlin Heidelberg 2007

first n natural numbers in $\mathbf{M}$, that is, the set $\{0, 1, \ldots, n-1\}$, where n is a nonstandard element of $\mathbf{M}$. Assume further that $\mathcal{M}$ contains the restriction of the algebraic operations $+, \times$ and the ordering $\leq$ to $\mathtt{universe}(\mathcal{M})$. Apart from that, $\mathcal{M}$ may contain a finite number of other functions and relations with finite arities. We assume that all of these functions and relations are first-order definable in $\mathbf{M}$. Our basic question is whether $\mathcal{M}$ can be extended into another model of Peano Arithmetic, possibly with different properties, in a way that no new elements are introduced into the interval $[0, n-1]$, that is, the initial segment $\{0, 1, \ldots, n-1\}$ remains exactly what it was in the original model, together with the relations and functions given in $\mathcal{M}$. In general, we may be looking for other extensions of $\mathcal{M}$, which are not models of Peano Arithmetic, but e.g., models of Zermelo-Fraenkel set Theory, or any other system of axioms. The important point is that no new elements are introduced into the interval $[0, n-1]$ and the existing functions and relations remain unchanged. It is possible e.g., that $\mathbf{M}$, the original model of Peano Arithmetic cannot be extended into a model of ZF at all, but $\mathcal{M}$ can be extended. The basic question investigated in this paper is the following: given $\mathcal{M}$ how can we decide whether such an extension exists or not. We want to be able to decide this question remaining "inside $\mathcal{M}$", that is, in a way that we only have to deal with objects which are first-order definable in $\mathcal{M}$.

We show that, for a countable $\mathcal{M}$, we can answer this question if we know which first-order sentences are true in the structure $\mathcal{M}$. Moreover we will show that if we want to extend a countable $\mathcal{M}$ into a model of a theory G (where we have some reasonable conditions on the way as G is given, e.g., a recursive set of sentences like Peano Arithmetic or ZF will be good) then such an extension of $\mathcal{M}$ exists iff it exists for every finite subset of G. This statement is the analogue of the compactness theorem.

The analogue of the completeness theorem involves the same type of extensions of a countable structure $\mathcal{M}$. We have the same conditions on the theory G. We show that $\mathcal{M}$ has such an extension into a model of G, iff G and the diagram of the model $\mathcal{M}$ are consistent in a sense described below. (We define the diagram of $\mathcal{M}$ as follows. We introduce constant symbols for each elements of the universe of $\mathcal{M}$. The diagram of $\mathcal{M}$ is the set of all statements φ with the following properties: (i) φ may contain the newly introduced constants symbols and symbols corresponding to the relations and function of $\mathcal{M}$, (ii) φ holds in $\mathcal{M}$, and (iii) φ is either an atomic formula or the negation of an atomic formula.)

We show that an extension of $\mathcal{M}$ with the required properties does not exist iff there is a proof P in $\mathbf{M}$ formulated in Gentzen's proof system **LK**, which shows a contradiction in the theory $G+$"the diagram of $\mathcal{M}$" so that there is a standard natural number d with the property that in $\mathbf{M}$ the following assertions hold:

(a) In the proof P the length of each formula is at most d

(b) P is first-order definable in $\mathcal{M}^c$ for some standard c, with a formula whose length is at most d.

In other words the existence of an extension is equivalent to the statement that the corresponding theory is consistent provided that we restrict our attention to proofs which contain only short (standard size formulas) and which are first-order definable in $\mathcal{M}^c$. The latter statement implies that the overall size of the proof must be polynomial in n. This suggests a connection between questions of complexity theory and the theory of extensions of nonstandard models. (Several connections of this type have been made already as we will describe it in the next section).

We have used Gentzen type proofs for the description of the analogue of the Completeness Theorem since in this case the bounds on the size of the various parameters of the proofs can be formulated in a very simple and transparent way. The results can be modified for Hilbert type proof systems as well, but in that case the formulation of the bounds are more complicated because we have to allow formulas whose lengths are nonstandard natural numbers.

Unlike its original version, the new Completeness Theorem does not hold at all for Gentzen type proofs without the Cut Rule. The reason is that a cut-free proof can be greater by an exponential factor than the corresponding proof with cuts, so its size may exclude first-order definability in $\mathcal{M}^c$.

Finally we note that Gödel's Incompleteness Theorem also has a natural analogue in the context described above. We intend to return to this question in a separate paper.

1.2 Connection with Complexity Theory

There is a known close connection between complexity theory and the extensions of models of arithmetic or its fragments in the sense used in the present paper. We give the outline of a few results to illustrate this connection. Máté has shown in 1990 that the $P =? NP$ and $NP =?\text{co}-NP$ questions are closely connected to questions about model extensions (see [9]) and described many related results. In particular Theorem 1 of [9] formulates a statement about model extensions which implies $NP \neq \text{co}-NP$. Here we do not describe the theorem in its full generality, but only state an illuminating special case. Assume that M, N are nonstandard models of Peano Arithmetic which are True models of Arithmetic in the sense that a statement is true in them iff it is true in the standard model of arithmetic, which consists of the natural numbers. Assume further that there is an initial segment of the models, up to a nonstandard integer n which are identical, and N is an extension of M up to 2^{n^k} for all standard integers k. (The last condition means that every element of M less than 2^{n^k} is also an element of N and the arithmetic operations on such elements are the same in the two models.) Suppose now that there is a graph G which is in both M and N, and its nodes are the first n natural numbers. If G does not have a Hamiltonian cycle in M, but G has a Hamiltonian cycle in N then $NP \neq \text{co}-NP$. In other words if we are able to prove the existence of an extension of the model M which keeps $[1, n-1]$ and G unchanged, while only increases the powersets of all of the sets $\{0, 1, \ldots, n^k\}$, where k is standard, and at the same time adds a Hamiltonian cycle to the graph G, then we have proved $NP \neq \text{co}-NP$.

If in the statement of the described theorem, we drop the condition that M and N are True models (see Theorem 2 of [9]) then, roughly speaking, the conclusion of the theorem is that we cannot prove in Peano Arithmetic that $NP = \text{co}{-}NP$ by proving about a fixed nondeterministic Turing machine T and a fixed positive integer k that T decides in time n^k whether a property in $\text{co} - NP$ holds on an input of size n. (This does not exclude the possibility that we can prove $NP = \text{co}{-}NP$ in some other way.)

Another connection between complexity and model extensions is the following. For the theorem that parity cannot be computed by an AC_0 circuit (see Ajtai [1], Furst, Saxe, Sipser [4]), we have the following equivalent statement in model theory. If M is a countable model of Peano Arithmetic, then there exists a model N of Peano Arithmetic, which is identical to M up to n, where N is a nonstandard integer, and there exists a set $A \subseteq \{0, 1, \ldots, n-1\}$, so that the set A is both in M and N, but in M the integer $|A|$ is even, while in N the integer $|A|$ is odd. The proof of this equivalence is given in [1], and it contains the proof of a special case of the compactness theorem formulated in this paper. The improved subexponential lower bounds on constant depth parity circuits (see Yao [11] and Hastad [6]) have also equivalent reformulations in terms of model extensions. In this case the set $A \subseteq \{0, 1, \ldots, n-1\}$ is of size $(\log n)^c$, where c is an arbitrarily small but nonstandard positive integer.

The third example is the theorem that PHP_n, that is, the Pigeonhole Principle for sets of size n, formulated as a propositional statement, cannot be proved by a constant depth polynomial size Frege system, which do not introduce new variables. In this case the first proof was found using model theoretic motivation and the proof itself is using model theoretic arguments in addition to combinatorial ones, (see [2]). Later purely combinatorial improvements were given (see [3]), but it seems likely, that it would have been much more difficult to find a first proof without the model theoretic motivation.

In this case we start with a countable model M of Peano Arithmetic with a nonstandard integer n in it and a k-ary relation R on A, so that $R \in M$. Now the extended model N is not a model of Peano Arithmetic, it is the following structure. The universe of N is the set $\{0, 1, \ldots, n-1\}$, that is, an initial segment of M up to n. N also contains the arithmetic operations of M restricted to the set $\{0, 1, \ldots, n-1\}$ and the relation R. Finally N contains a one-to-one function f, which maps the set $\{0, 1, \ldots, n-1\}$ onto the set $\{0, 1, \ldots, n-2\}$, that is, it violates the Pigeonhole Principle. It is proved in [2], that if such an extension exists, so that in N the induction holds (that is, each first-order definable nonempty subset has a smallest element), then there is no polynomial size constant depth Frege proof for PHP_n, and in fact, the two statements are equivalent. In [2] such a model N is constructed partly with combinatorial techniques related to constant depth circuits and partly with a model theoretic technique which is the nonstandard analogue of Cohen's method of forcing. The method of forcing is applicable in general to make connections between problems of proof complexity and model extensions (see Krajicek [8]).

2 Set Theory

2.1 Finite Set Theory

We assume that the reader is familiar with the basic concepts of Zermelo-Fraenkel set theory (see e.g. [7]).

If x is a set then then let $I^1(x) = \bigcup x$ and $I^{n+1}(x) = I^1(I^n(x))$ for $n = 1, 2, \ldots$. The set $x \cup \bigcup_{i=1}^{\infty} I_n(x)$ is the transitive closure, or transitive hull of the set x. The transitive closure of the set x will be denoted by $\mathtt{cl}(x)$. A set is called hereditarily finite if its transitive closure is finite. (Or equivalently x is hereditarily finite if there is a natural number n so that $I^n(x)$ is finite and $I^{n(x)} = I^{n+1}(x)$). If we omit the axiom of infinite set from Zermelo-Fraenkel axiom system of set theory and add the axiom that every set is hereditarily finite, then the resulting system of axioms will be called Finite Set Theory. This system is equivalent to Peano Arithmetic in the sense, that the natural numbers (that is, ordinals) in a model of Finite Set Theory form a model of Peano Arithmetic, and conversely, if we start with a model of Peano Arithmetic, then we may define sets, e.g., by Gödel numbering, so that altogether they form a model of Finite Set Theory. Moreover if we do these steps one after the other, starting with any of the two models, then we get back the original model (up to isomorphism). In this paper we prefer Finite Set Theory to Peano Arithmetic, since it makes the statement and proofs of our results simpler.

2.2 Standard and Nonstandard Sets

Definition 1. Assume that $\mathbf{M}$ is a model of Finite Set Theory and x_0 is an element of $\mathtt{universe}(\mathbf{M})$. We say that x_0 is a standard element of $\mathbf{M}$ if there is no infinite sequence $x_1, x_2, \ldots$ with the property that for all $i = 0, 1, \ldots$, we have $x_i, x_{i+1} \in \mathtt{universe}(\mathbf{M})$, and $\mathbf{M} \models x_{i+1} \in x_i$. It is easy to see that the standard elements of a nonstandard model $\mathbf{M}$ of Finite Set Theory form a submodel which is isomorphic to the model of hereditarily finite sets (in the world) and there is a unique isomorphism ι between the two models. Therefore, unless we explicitly state otherwise, we will assume that $\mathbf{M}$ has the property that for each standard $x \in \mathbf{M}$ we have $x = \iota(x)$. In other words, the standard part of $\mathbf{M}$ is the set of hereditarily finite sets and on its elements the relation "$\in$" is the same as in the world. In particular, the natural numbers of the world form an initial segment of the natural numbers of $\mathbf{M}$, and on this segment the arithmetic operations and ordering in the world is the same as in $\mathbf{M}$.

Definition 2. Suppose that $\mathbf{M}$ is a model of Finite Set Theory and $B \in \mathbf{M}$. The set of all standard elements x of $\mathbf{M}$ with the property $\mathbf{M} \models x \in B$, will be called the standard part of B, and will be denoted by $\mathtt{standard}(B)$.

3 First-Order Formulas

In our terminology concerning first-order logic and Gentzen style proofs, we follow Takeuti's "Proof Theory" (see [10]), with the following exception. Since we

will not use variables for functions and relations, the meaning of the expressions "function symbol", "relation symbol" resp. "constant symbol", will be the same as "function constant", "predicate constant", resp. "individual constant" in [10].

Definition 3. If we say that $\mathcal{L}$ is a first-order language then we will always assume, unless we explicitly state otherwise, that $\mathcal{L}$ is a first-order language with equality and with a finite number of relation, function and constant symbols. By definition each constant symbol is a 0-ary function symbol as well. Therefore the expression "all function symbols of a language" includes the constant symbols of the language. Let $\mathbf{s}(\mathcal{L})$ be the set of symbols that we can use for the construction of a first-order formula of $\mathcal{L}$. That is, $\mathbf{s}(\mathcal{L})$ contains relation, function, and constant symbols, symbols for free and bound variables, symbols for the boolean logical operations, symbols for universal/existential quantifiers, left and right parenthesis, and the symbol ",". Each first-order formula is a finite sequence constructed from the elements of $\mathbf{s}(\mathcal{L})$ according to certain rules (see e.g. [10]). With this definition a language has an infinite number of symbols.

We will assume that each first-order language is represented in set theory, that is, its symbols are sets. The sets which are representing the various symbols can be arbitrarily chosen, but if we say that $\mathcal{L}$ is a first-order language then we assume that such a representation is fixed for $\mathcal{L}$. For another language $\mathcal{K}$ the representation can be different, e.g., the logical symbol "$\wedge$" can be represented by different sets in the languages $\mathcal{L}$ and $\mathcal{K}$. Usually it is assumed that a language has an infinite (perhaps countably infinite) number of free and bound variables, and it is not important how these variables are represented in set theory. Our situation is somewhat different, since we will want to consider a language as an element of a finite structure. Of course the infinite number of variables cannot be in a finite structure, but they can be in its Cartesian powers. To make this possible we will always assume that the variables are finite sequences made from the elements of the universe, as explained in the following definitions.

Definition 4. Assume that $\mathcal{L}$ is a first-order language. We introduce two new symbols, which will be denoted by $\mathbf{b}$ and $\mathbf{f}$. If we omit the free and bound variables from $\mathbf{s}(\mathcal{L})$ but add the symbols $\mathbf{b}$ and $\mathbf{f}$, the resulting set will be denoted by $\mathtt{Symb}(\mathcal{L})$.

Definition 5. Assume $\mathcal{L}$, $\mathcal{H}$ are first-order languages and $\mathcal{M}$ is a model of $\mathcal{H}$. We say that $\mathcal{L}$ is in $\mathcal{M}$ iff there is a positive integer k so that $\mathtt{Symb}(\mathcal{L}) \subseteq (\mathtt{universe}(\mathcal{M}))^k$, and the following conditions are satisfied:

(1) *x is a free variable of $\mathcal{L}$ iff there exists a natural number k and there exist $a_1, \ldots, a_k \in \mathtt{universe}(\mathcal{M})$ so that $x = \langle \mathbf{f}, a_1, \ldots, a_k \rangle$*

(2) *u is a bound variable of $\mathcal{L}$ iff there exists a natural number k and there exist $a_1, \ldots, a_k \in \mathtt{universe}(\mathcal{M})$ so that $u = \langle \mathbf{b}, a_1, \ldots, a_k \rangle$*

4 Extensions of Interpretations

Definition 6. Assume that $\mathcal{K}, \mathcal{L}$ are first-order languages and $\mathcal{K} \subseteq \mathcal{L}$, that is, every relation or function symbol of $\mathcal{K}$ is also a symbol of the same type (relation or function) in $\mathcal{L}$ and has the same arity. Suppose further that $\mathcal{L}$ has a unary relation symbol $\mathbf{U}$ not contained in $\mathcal{K}$. In this case we say that $\mathcal{L}$ is a $\mathbf{U}$-extension of $\mathcal{K}$.

Definition 7. Suppose that $\mathcal{L}$ is a $\mathbf{U}$-extension of $\mathcal{K}$. The interpretation λ of $\mathcal{L}$ is a $\mathbf{U}$-extension of the interpretation κ of $\mathcal{K}$ if the following conditions are satisfied:

(3) $\mathtt{universe}(\kappa) \subseteq \mathtt{universe}(\lambda)$ *and for all* $x \in \mathtt{universe}(\lambda)$ *we have that* $x \in \mathtt{universe}(\kappa)$ *iff* $\lambda \models \mathbf{U}(x)$

(4) *Assume that* R *is a* k*-ary relation symbol in* $\mathcal{K}$*. Then, for all* $a_1, \ldots, a_k \in \mathtt{universe}(\kappa)$*, we have* $\kappa \models R(a_1, \ldots, a_k)$ *iff* $\lambda \models R(a_1, \ldots, a_k)$

(5) *Assume that* f *is a* k*-ary function symbol in* $\mathcal{K}$*. Then, for all* $a_1, \ldots, a_k, b \in \mathtt{universe}(\kappa)$*, we have* $\kappa \models b = f(a_1, \ldots, a_k)$ *iff* $\lambda \models b = f(a_1, \ldots, a_k)$

Definition 8. Assume that $\mathcal{L}$ is a $\mathbf{U}$-extension of $\mathcal{K}$, κ is an interpretation of $\mathcal{K}$, and T is a theory in $\mathcal{L}$. We say that T has a model over κ if there exists an interpretation λ of $\mathcal{L}$, so that λ is a U-extension of κ and λ is a model of T.

5 First-Order Definability

In all of the definitions in this section we assume that $\mathcal{K}$ is a first-order language and κ is an interpretation of $\mathcal{K}$.

To formulate the compactness theorem we need the notion of first-order definability for relations and functions. In this section we provide the definitions of these concepts.

Definition 9. Suppose that k is a nonnegative integer and R is a k-ary relation on $\mathtt{universe}(\kappa)$. We say that R is first-order definable iff there is a nonnegative integer l, there are elements $a_1, \ldots, a_l$ of $\mathtt{universe}(\kappa)$ and there is a first-order formula $\varphi(x_1, \ldots, x_l, y_1, \ldots, y_k)$ of $\mathcal{K}$ so that

$$\kappa \models \forall y_1, \ldots, y_k,\ R(y_1, \ldots, y_k) \leftrightarrow \varphi(a_1, \ldots, a_l, y_1, \ldots, y_k)$$

In this case we say that R is first-order definable in κ with formula φ at $\langle a_1, \ldots, a_l \rangle$. If R is a k-ary relation on $(\mathtt{universe}(\kappa))^i$, than R is first-order definable in κ if the corresponding ik-ary relation is first-order definable in κ.

Definition 10. Assume now that $f(x_1, \ldots, x_k)$ is a k-ary function defined on $\mathtt{universe}(\kappa)$ and with values in $\mathtt{universe}(\kappa)$. We say that f is first-order definable in κ if the $k+1$-ary relation $y = f(x_1, \ldots, x_k)$ is first-order definable in κ. If f is a k-ary function on $(\mathtt{universe}(\kappa))^i$ with values in $(\mathtt{universe}(\kappa))^j$, then f is first-order definable in κ if the corresponding $j + ik$-ary relation is first-order definable in κ.

Definition 11. Suppose that k is a nonnegative integer and B is a subset of $(\mathtt{universe}(\kappa))^k$. B is first-order definable in κ if the k-ary relation, which holds exactly on the elements of B, is first-order definable in κ.

Definition 12. Assume that λ is the interpretation of a first-order language $\mathcal{L}$. We say that λ is first-order definable in κ, if there exists a positive integer k so that $\mathtt{universe}(\lambda) \subseteq (\mathtt{universe}(\kappa))^k$, moreover $\mathtt{universe}(\lambda)$ and all of the relations and functions of the structure defined by λ are first-order definable in κ.

Remark. Suppose that $\mathbf{M}$ is a model of Finite set Theory. We consider $\mathbf{M}$ as an interpretation of the language of set theory, therefore the definitions of this section are applicable for $\mathbf{M}$.

6 Compactness Theorem

We formulate now the analogue of the Compactness Theorem.

Theorem 13. *Assume that $\mathcal{L}$, $\mathcal{K}$ are first-order languages, $\mathcal{L}$ is a $\mathbf{U}$-extension of $\mathcal{K}$, $\mathbf{M}$ is a model of Finite Set Theory, and κ is an interpretation of $\mathcal{K}$ so that the following conditions are satisfied:*

(6) $\mathtt{universe}(\kappa)$ *is a countable set.*

(7) κ *is first-order definable in* $\mathbf{M}$.

(8) *There is an $X \in \mathbf{M}$ so that for all $x \in \mathbf{M}$ we have, $x \in \mathtt{universe}(\kappa)$ iff $\mathbf{M} \models x \in X$.*

(9) $\mathbf{M}$ *contains a nonstandard natural number.*

Assume further that G is a theory in $\mathcal{L}$ so that there is a set $B \in \mathbf{M}$ with the property that $G = \mathtt{standard}(B)$. For each positive integer k, let G_k be the set of those formulas from G whose length is less than k. Then, the theory G has a model over κ iff for each positive integer k, the theory G_k has a model over κ.

Remark. The assumption that $\mathbf{M}$ is a model of Finite Set Theory can be replaced by much weaker assumptions, e.g., $\mathbf{M}$ can be a suitably chosen segment of a model of Finite Set Theory. A sufficient condition on the segment, which guarantees that the theorem remains true, will be described in a more detailed version of the present paper.

7 First-Order Definability of Sets Sequences

In all of the definitions in this section we assume that $\mathcal{K}$ is a first-order language and κ is an interpretation of $\mathcal{K}$.

Definition 14. Assume that $s, r_1, \ldots, r_i$ are nonnegative integers, and H is a set, so that each element of H is a sequence $S = \langle S_1, \ldots, S_s \rangle$, where S_i is an r_i-ary relation on $\mathtt{universe}(\kappa)$ for $i = 1, \ldots, s$. The set H is first-order definable in κ if there exists a nonnegative integer l, there exists a set $B \subseteq (\mathtt{universe}(\kappa))^l$ first-order definable in κ, and for each $i = 1, \ldots, s$ there exists a first-order formula $\varphi_i(x_1, \ldots, x_l, y_1, \ldots, y_{r_i})$ of $\mathcal{K}$, so that the following two conditions are satisfied:

(10) *For each $\langle a_1, \ldots, a_l \rangle \in B$ there exists an $S = \langle S_1, \ldots, S_s \rangle \in H$ so that for each $i = 1, \ldots, s$, S_i is first-order definable in κ with formula φ_i at $\langle a_1, \ldots, a_l \rangle$.*

(11) *For each $S = \langle S_1, \ldots, S_s \rangle \in H$ there exists an $\langle a_1, \ldots, a_l \rangle \in B$ so that for each $i = 1, \ldots, s$, S_i is first-order definable in κ with formula φ_i at $\langle a_1, \ldots, a_l \rangle$.*

Definition 15. Assume that d is a positive integer and ξ is an interpretation of a language $\mathcal{L}$ so that $\mathtt{universe}(\xi) \subseteq (\mathtt{universe}(\kappa))^d$. Assume that the language $\mathcal{L}$ has t_1 relation symbols $R_1, \ldots, R_{t_1}$ with arities $r_1, \ldots, r_{t_i}$, and $\mathcal{L}$ has t_2 function symbols $f_1, \ldots, f_{t_2}$ with arities $s_1, \ldots, s_{t_2}$. Let $W_0, W_1, \ldots W_{t_1+t_2}$ be the following sequence of relations on $(\mathtt{universe}(\kappa))^d$. For all $x \in (\mathtt{universe}(\kappa))^d$, $W_0(x)$ iff $x \in \mathtt{universe}(\xi)$. $W_i = \xi(R_i)$ for $i = 1, \ldots, t_1$. For all $i = 1, \ldots, t_2$ W_{t_1+i} is an s_{i+1}-ary relation, and for all $y, x_1, \ldots x_{s_i} \in (\mathtt{universe}(\kappa))^d$, we have $W_{t_1+i}(y, x_1, \ldots, x_{s_i})$ iff $\xi \models y = f_i(x_1, \ldots, x_{s_i})$.

For each $i = 0, 1, \ldots, t_1 + t_2$ we define a relation W'_i on $\mathtt{universe}(\kappa)$. If the arity of W_i is q then the arity of W'_i is dq. For all $x_1, \ldots, x_{dq}$, we have $W'_i(x_1, \ldots, x_{dq})$ iff $W_i(\langle x_1, \ldots, x_d \rangle, \ldots, \langle x_{d(q-1)+1}, \ldots, x_{dq} \rangle)$. We will use the notation $S^{(\xi)} = \langle W'_0, \ldots, W'_{t_1+t_2} \rangle$.

The interpretation ξ is first-order definable in κ if all of the following relations are first-order definable in κ: $W'_0, \ldots, W'_{t_1+t_2}$.

Definition 16. Suppose that W is a set of interpretations of $\mathcal{L}$ and d is a positive integer so that for all ξ in W, $\mathtt{universe}(\xi) \subseteq (\mathtt{universe}(\kappa))^d$. We say that the set W is first-order definable in κ, if the set of sequences of relations $\{S^{(\xi)} \mid \xi \in W\}$ is first-order definable in κ, where $S^{(\xi)}$ is defined in the previous definition.

8 The Proof of the Compactness Theorem

We will use the following lemmas and definitions in the proof of the theorem.

Lemma 17. *For each positive integer k there exists a first-order formula of ψ_k of $\mathcal{L}$ so that $\models G_k \leftrightarrow \psi_k$. Moreover there is a $u \in \mathbf{M}$ so that $\mathbf{M} \models$ "u is a function" and for each (standard) positive integer k we have that $\mathbf{M} \models \psi_k = u(k)$.*

Proof of Lemma 17. Since the number of sentences of $\mathcal{L}$ with depth less than k is finite, their conjunction is a sentence equivalent to G_k. We will denote this sentence by ψ_k. By the assumptions of the theorem, G is the standard part of a set B in $\mathbf{M}$. Let g be the function which assigns to each formula of $\mathcal{L}$ its depth.

There is a function v in $\mathbf{M}$ which is an extension of g. Therefore the described construction of ψ_k can be carried out in $\mathbf{M}$ which gives the function u with the properties listed in the lemma. *Q.E.D.*(Lemma 17).

Definition 18. In the following ψ_k will denote a formula of $\mathcal{L}$ with the properties described in Lemma 17.

Definition 19. Let $\mathcal{K}_\kappa$ be the extension of $\mathcal{K}$ by a new constant symbol c_a for each $a \in K$ and let κ' be the extension of κ to $\mathcal{K}'$ with the property $\kappa'(c_a) = a$ for all $a \in K$. We assume that the new constant symbols are not contained in $\mathcal{L}$. We will use the notations $\mathcal{L}' = \mathcal{K}' \cup \mathcal{L}$. $\mathcal{L}'$ is a $\mathbf{U}$-extension of $\mathcal{K}'$. We will assume that these constant symbols are represented by the elements of K^c for some constant c, so the language $\mathcal{L}'$ is in K^c.

Definition 20. Let $\mathbf{d}(\kappa)$ be the theory in $\mathcal{K}'$ consisting of all satements σ of $\mathcal{K}'$ so that (a) σ is either an atomic formula or the negation of an atomic formula, and (b) $\kappa' \models \sigma$.

Definition 21. Assume that κ is an interpretation of the language $\mathcal{K}$ with the properties described in the theorem and $a_1, \dots, a_l \in \mathtt{universe}(\kappa)$. $\mathbf{d}_\kappa[a_1, \dots, a_l]$ will denote the set of all atomic formulas σ of $\mathcal{K}'$ with the following two properties:

(i) $\sigma \in \mathbf{d}(\kappa)$,
(ii) if $a \in \mathtt{universe}(\kappa)$ and c_a occurs in σ then $a \in \{a_1, \dots, a_l\}$

Lemma 22. *For all positive integers l, r we have that, for all propositional formulas $P(z_1, \dots, z_l)$ of the language $\mathcal{L}$ with the only free variables $z_1, \dots, z_l$, there exists a propositional formula $P'_r(z_1, \dots, z_l)$ of the language $\mathcal{K}$ so that for each interpretation κ of $\mathcal{K}$ with the properties listed in the theorem the following holds.*

For all $a_1, \dots, a_l \in \mathtt{universe}(\kappa)$, $\kappa \models P'_r(a_1, \dots, a_l)$ iff the following holds: in the theory $\mathbf{d}_\kappa[a_1, \dots, a_l] \cup \{P(c_{a_1}, \dots, c_{a_l})\}$ of the language $\mathcal{L}'$ there is no contradiction of length at most r.

Proof of Lemma 22. Suppose that $\langle a_1^{(j)}, \dots, a_l^{(j)} \rangle \in K^l$ for $j = 0, 1$. We say that $\langle a_1^{(0)}, \dots, a_l^{(0)} \rangle$, $\langle a_1^{(1)}, \dots, a_l^{(1)} \rangle$ are equivalent if for all t-ary relation symbols R and for all t-ary function symbols f of $\mathcal{K}$ we have that for all $i_0, i_1, \dots, i_t \in \{1, \dots, l\}$ the following two conditions hold:

(i) $\kappa \models R(a_{i_1}^{(0)}, \dots, a_{i_l}^{(0)})$ iff $\kappa \models R(a_{i_1}^{(1)}, \dots, a_{i_l}^{(1)})$ and
(ii) $\kappa \models a_{i_0}^{(0)} = f(a_{i_1}^{(0)}, \dots, a_{i_l}^{(0)})$ iff $\kappa \models a_{i_0}^{(1)} = f(a_{i_1}^{(1)}, \dots, a_{i_l}^{(1)})$.

This defines an equivalence relation on the ordered l tuples of K. The number of equivalence classes is clearly finite and remains below a bound which is a primitive recursive function of l. The truth value of statement $S \equiv$"in the theory $\mathbf{d}_\kappa[a_1, \dots, a_l] \cup \{P(c_{a_1}, \dots, c_{a_l})\}$ there is a contradiction of length at most r" depends only on the equivalence class of the l-tuple $\langle a_1, \dots, a_l \rangle$. Therefore the formula $P_r(z_1, \dots, z_r)$ simply describes those equivalence classes, in terms of $c_{a_1}, \dots, c_{a_l}$, where statement S holds. *Q.E.D.*(Lemma 22).

Definition 23. We assume that $\tau = \langle \tau_1, \tau_2, \ldots \rangle$ is a finite or infinite sequence of sentences of $\mathcal{L}'$ each in prenex form. For each $k = 1, 2, \ldots$, where τ_k is defined, we introduce skolem functions for the formula τ_k as new function symbols. The skolemized version of τ_k will be denoted by $\mathtt{skolem}(\tau_k)$. For each fixed positive integer k, the extension of $\mathcal{L}'$ with the newly introduced function symbols is a first-order language that we denote by $\mathbf{s}_k^{(\tau)}$. We assume that the new function symbols introduced for various values of k are distinct. We will denote by $\mathbf{S}_k^{(\tau)}$ the union of the languages $\mathbf{s}_1, \ldots, \mathbf{s}_k$.

Definition 24. We assume that for each $k = 1, 2, \ldots$ the formula ψ_k, as defined in the proof of Lemma 17, is in prenex form. We will use the definition of $\mathbf{S}_k^{(\tau)}$ mainly in the special case $\tau = \psi = \langle \psi_1, \psi_2, \ldots \rangle$. In this case we will write $\mathbf{S}_k$ as an abbreviation for $\mathbf{S}_k^{(\psi)}$.

Definition 25. Assume that $P(x_1, \ldots, x_s)$ is a propositional formula of a language with the only variables $x_1, \ldots, x_s$, and W is a finite set of closed terms of the same language. $\mathtt{subst}(P, W)$ will denote the conjunction of all of the formulas that we can get by substituting elements of W for each of the variables $x_1, \ldots, x_s$ in the formula P. (Note that the elements of $\mathtt{subst}(P, W)$ do not contain variables.)

Definition 26. Assume that $\tau = \langle \tau_1, \tau_2, \ldots \rangle$ is a finite or infinite sequence of sentences from $\mathcal{L}'$. For each positive integer k where τ_k is defined we will use the notation $\tau^{(k)} = \langle \tau_1, \ldots, \tau_k \rangle$. Suppose further that such an integer k is fixed and H is a set of terms of $\mathbf{S}_k^{(\tau)^{(k)}}$. For each j-ary function symbol g of $\mathbf{S}_k^{(\tau)}$, let $g^{(H)} = \{g(t_1, \ldots, t_j) | t_1, \ldots, t_j \in H\}$ and let $\overline{H^\tau} = H \cup \bigcup g^{(H)}$, where the union is taken for all function symbols g of $\mathbf{S}_k^{(\tau)}$. (In the case of $\tau = \psi$ we may omit the superscript ψ.)

Definition 27. We define a full information game between two players: Player 1 and Player 2. The game depends on a finite or infinite sequence $\tau = \langle \tau_1, \tau_2, \ldots \rangle$ whose elements are sentences from the language $\mathcal{L}$ and a positive integer k so that τ_k is defined. If τ is fixed then we will denote the corresponding game by $\mathcal{G}(\tau, k)$.

The players move alternately, starting with Player 1. There will be altogether k moves. The moves are numbered by the positive integers in increasing order. At move i first Player 1 is making her ith move then Player 2. We will define the position of the game after the ith move of Player j and will denote it by $Q_{i,j}$. The starting position is denoted by $Q_{0,2}$ (since intuitively it can be considered as the position after the "0th move" of the Player).

$Q_{i,j}$ is a pair $\langle V_{i,j}, f_{i,j} \rangle$.

By definition the starting position is $Q_{0,2} = \langle \emptyset, \emptyset \rangle$.

For all $i = 1, \ldots, k$, the ith move of Player 1 is an arbitrary element b_i of K. The new position $Q_{i,1}$ after this move is defined by

$$V_{i,1} = \overline{(V_{i-1,2} \cup \{b_i\})^{(\langle \tau_1, \ldots, \tau_i \rangle)}}, \ f_{i,1} = f_{i-1,2}$$

For all $i = 1, \ldots, k$, the ith move of Player 2 is an extension of the function $f_{i,1}$ to the set $V_{i,1}$ with the property that each new value of the function is in K. This extension is $f_{i,2}$ and $V_{i,2} = V_{i,1}$.

Player 1 wins, if there is a contradiction shorter than k in the theory $T_{\mathcal{G}(\tau,k)}$ which is the set of sentences:

$$\mathbf{d}_\kappa[b_1, \ldots, b_k] \cup$$
$$\bigcup_{i=1}^{k} \left\{ \left(\bigwedge_{q \in V_{i,2}} \mathbf{U}(q) \to q = c_{f_{k,2}(q)} \right) \wedge \mathtt{subst} \left(\bigwedge_{l=1}^{i} \mathtt{skolem}(\tau_l), V_{i,2} \right) \right\}$$

Lemma 28. *Assume that $1 \leq k < l$ are integers. If I_l is an instance of the game $\mathcal{G}(\tau, l)$ where Player 2 is the winner then Player 2 is also the winner in the instance I_k of the game $\mathcal{G}(\tau, k)$, where I_k consists of the first k moves of the game I_l.*

Proof. The definition of the games $\mathcal{G}(\tau, k)$, $\mathcal{G}(\tau, l)$ clearly imply that all of the moves of I_k are legal in the game $\mathcal{G}(\tau, k)$. This is an immediate consequence of the fact that the definition of the legality of the ith move in the game $\mathcal{G}(\tau, k)$ does not depend on k. The theory $T_{\mathcal{G}(\tau,k)}$ with respect to the game I_k is clearly a subset of the theory $T_{\mathcal{G}(\tau,l)}$ with respect to the game I_2. We know that there is no contradiction of length less than l in the theory $T_{\mathcal{G}(\tau,l)}$, therefore $k < l$ implies that there is no contradiction of length less than k in the theory $T_{\mathcal{G}(\tau,k)}$. *Q.E.D.*(Lemma 28).

Lemma 29. *Under the conditions of Theorem 13, for all positive integers k, Player 2 has a winning strategy in the game $\mathcal{G}(\psi, k)$.*

Proof. The assumptions of the theorem imply that there exists an interpretation ζ of $\mathcal{L}'$ so that $\zeta \models G_k$ and ζ is a $\mathbf{U}$-extension of κ'. By Lemma 17 we have that $\zeta \models \bigwedge_{j=1}^{k} \psi_j$

The strategy of Player 2, that we will denote by S_k, is the following. Assume that we are at move i. Player 2 has to extend the function $f_{i,1}$ to the set $V_{i,1}$. Each element of $V_{i,1}$ is a closed term of $\mathbf{S}_k$. Assume that q is an element of $V_{i,1}$ so that $f_{i,1}(q)$ is not defined. If $\zeta \models \neg\mathbf{U}(q)$ then $f_{i,2}(q)$ is an arbitrary element of K. If $\zeta \models \mathbf{U}(q)$ then, the fact that ζ is a $\mathbf{U}$-extension of κ' implies that $\zeta \models q = c_a$ for some $a \in K$. In this case let $f_{i,2}(q) = a$ for one of the elements a with this property.

The definition of the moves of Player 2 implies that ζ is a model of the theory $T_{\mathcal{G}}$. Since ζ is a model of theory $T_{\mathcal{G}}$ we have that $T_{\mathcal{G}}$ theory does not contain contradictions at all. This implies that the strategy S_k, that we have described, is a winning strategy for Player 2 in the game $\mathcal{G}(\psi, k)$. *Q.E.D.*(Lemma 22).

Now we may complete the proof of Theorem 13. We will work with games in the world and in $\mathbf{M}$. First consider an infinite game $\mathcal{G}(\psi, \infty)$, in the world, played by Player 1 and Player 2. The players move alternately and Player 1 starts the game. The kth move of Player i is a legal move if it would be a legal move in the game $\mathcal{G}(\psi, k)$. Player 2 wins, if for each positive integer $k = 1, 2, \ldots$, Player 2 wins in the game $\mathcal{G}(\psi, k)$ where the moves are the first k-moves of the infinite game $\mathcal{G}(\psi, \infty)$.

Now we consider the following instance of the game $\mathcal{G}(\psi, \infty)$. Player 1 plays according to the following strategy. K is countable, therefore there exists an enumeration $a_1, a_2, \ldots$ of all of the elements of K. The ith move of Player 1 is the element a_i (that is $b_i = a_i$ using the notation of the definition of the game $\mathcal{G}(\tau, k)$).

For the definition of the strategy of the second player we consider the games $\psi(\mathcal{G}, k)$, $k = 1, 2, \ldots$ in $\mathbf{M}$. According to Lemma 29 for each fixed standard positive integer k the following holds: $\mathbf{M} \models$ " Player 2 has a winning strategy in the game $\mathcal{G}(\psi, k)$". This together with the facts that $\mathbf{M}$ is a model of Finite Set Theory and it contains a nonstandard natural number imply that there is a nonstandard $k_0 \in \mathbf{M}$ so that $\mathbf{M} \models$ "Player 2 has a winning strategy S_{k_0} in game $\mathcal{G}(\psi, k_0)$". Now we can define the strategy of Player 2 in the game $\mathcal{G}(\psi, \infty)$. For each (standard) positive integer i Player 2 makes his i-th move according to strategy S_{k_0}.

Assume that in the game $\mathcal{G}(\psi, \infty)$ the players are playing according to the described strategies. Lemma 28 implies that Player 2 wins in all of the games $\mathcal{G}(\psi, k)$ for $k = 1, 2, \ldots$. Therefore for each $k = 1, 2, \ldots$ in the theory

$$\mathbf{d}_\kappa[a_1, \ldots, a_k] \cup$$
$$\bigcup_{i=1}^{k} \left\{ \left(\bigwedge_{q \in V_{i,2}} \mathbf{U}(q) \to q = c_{f_{k,2}(q)} \right) \wedge \mathtt{subst}\left(\bigwedge_{l=1}^{i} \mathtt{skolem}(\tau_l), V_{i,2} \right) \right\}$$

there is no contradiction of length less than k. The various functions $f_{k,2}$ are compatible, so if f is their common extension then for all $k = 1, 2, \ldots$ in the theory

$$T^{(k)} = \mathbf{d}_\kappa[a_1, \ldots, a_k] \cup$$
$$\bigcup_{i=1}^{k} \left\{ \left(\bigwedge_{q \in V_{i,2}} \mathbf{U}(q) \to q = c_{f(q)} \right) \wedge \mathtt{subst}\left(\bigwedge_{l=1}^{i} \mathtt{skolem}(\tau_l), V_{i,2} \right) \right\}$$

there is no contradiction of length shorter than k. As a consequence of this and the fact that $\mathtt{universe}(\kappa) = \{a_1, a_2, \ldots\}$, we have that the theory

$$T = \mathbf{d}(\kappa) \cup$$
$$\bigcup_{k=1}^{\infty} \bigcup_{i=1}^{k} \left\{ \left(\bigwedge_{q \in V_{i,2}} \mathbf{U}(q) \to q = c_{f(q)} \right) \wedge \mathtt{subst}\left(\bigwedge_{l=1}^{i} \mathtt{skolem}(\tau_l), V_{i,2} \right) \right\}$$

does not contain a contradiction. By the compactness theorem there exists a model D of T. Since T does not contain quantifiers, D has a minimal submodel D_0, containing the values of all closed terms. T contains the skolemized version of the theory $\{\psi_k \mid k = 1, 2, \ldots\}$ therefore D_0 is a model of $G = \bigcup_{k=1}^{\infty} G_k$. We claim the D_0 is a $\mathbf{U}$-extension of κ. Indeed D_0 consists of the values of the terms in the set $\bigcup_{i=1}^{\infty} V_{i,2}$. Moreover for each term q in this set, the theory T contains a sentence which implies $\mathbf{U}(q) \to q = c_{f(q)}$. Consequently all of the elements of

D_0 where the relation **U** holds belong to K. On the other hand for each element of K there is q so that $f(q) = a$. Indeed, the choice of the strategy of Player 1 implies that $a = b_i$ for some positive integer i, and so $c_{a \in V_{i,1}}$ which implies $a \in \mathtt{range}(f_{i,2})$ and o $K \subseteq D_0$. Finally, since $\mathbf{d}(\kappa) \subseteq T$, all of the relation and function symbols of $\mathcal{K}$ has the interpretation in D_0 that is required by the definition of a **U**-extension. *Q.E.D.*(Theorem 13)

Lemma 30. *Assume that $\mathcal{L}$, $\mathcal{K}$ are first-order languages, $\mathcal{L}$ is a* **U***-extension of $\mathcal{L}$, k is a positive integer, $\tau = \langle \tau_1, \ldots, \tau_k \rangle$ is a sequence of sentences of $\mathcal{L}$. Then there is a sentence η of the language $\mathcal{K}$ such that for each interpretation κ we have the following*

$\kappa \models \eta$ iff Player 2 has a winning strategy in game $\mathcal{G}(\tau, k)$.

Proof. η will be the sentence whose Ehrenfeucht-Frechet game is the game $\mathcal{G}(\tau, k)$. Each move of Player 1 is an element of $K = \mathtt{universe}(\kappa)$, and each move of Player 2 is a sequent of elements from K. The choices of Player 1 correspond to variables bound by the universal quantifiers and for Player 2 the variables bound by the existential quantifier. The number of elements of K that Player 2 has to give in a single move, may depend on the earlier moves of Player 1. However this is not a problem since there is a finite upper bound on this number. (This bound is a primitive recursive function of $\langle \tau_1, \ldots, \tau_l \rangle$.) Suppose that the moves of the players, that is, the corresponding sequence of elements are fixed. Lemma 22 shows that Player 2 is the winner if a propositional formula of $\mathcal{K}$ is true in κ if we substitute the moves of the players for its variables. *Q.E.D.*(Lemma T1.8)

9 The Completeness Theorem

9.1 Proofs in LK

We will formulate and prove a generalization of Gödel's completeness theorem for the proof system introduced by Gentzen, namely the "logistischer klassischer Kalkül" or **LK** described in [5] or [10].

We assume that the reader is familiar with **LK**. Here we give only a brief sketch of its definition without formulating its rules of inference.

Definition 31. Assume that $\langle T, \leq \rangle$ is a partially ordered set and $x, y \in T$. x is a predecessor of y if $x < y$ and there is no $z \in T$ with $x < z < y$. In this case y is called the successor of x.

Definition 32. A triplet $\langle T, \leq_T, C \rangle$ is a binary PO-set if the following conditions are satisfied.

(12) *T is a nonempty set, $\leq_T$ is a partial ordering of T with a smallest element 0_T.*

(13) *Each element of T has at most two successors.*

(14) *C is a binary relation on T with the property that if $b, c \in T$, $b \neq c$ are both successors of a $t \in T$, then $C(b,t) \equiv \neg C(c,t)$.*

According to its definition, an **LK** proof in the first-order language $\mathcal{H}$ is a pair of functions g_0, g_1, defined on the set of nodes of a finite binary PO-set $\langle T, \leq_T, C\rangle$, so that for each $t \in T$, both $g_0(t)$ and $g_1(t)$, are finite sequences whose elements are formulas of $\mathcal{H}$. The sequence $g_0(t)$ is the antecedent and the sequence $g_1(t)$ is the succedent. The pair $\langle g_0(t), g_1(t)\rangle$ is the sequent assigned to the node t. Moreover the sequents assigned to a node t and to its successors in the PO-set must be formed according to an inference rule of **LK**, and the sequents assigned to source nodes must be initial sequents of **LK**.

For the definition of inference rules and initial sequents of **LK** see e.g., [10].

We will denote by I the set of symbols of the language $\mathcal{H}$ occurring in the proof.

Definition 33. In the above definition of a proof in **LK** we have used the notion of sequences at two different places. Namely formulas are sequences, the antecedents and succedents are sequences of formulas. For the present purposes we define a sequence in the following way. We fix an arbitrary finite totally ordered set Λ and we represent the sequence $\langle a_0, a_1, \ldots, a_{k-1}\rangle$ by the function f which is defined on the smallest k elements $\lambda_0 < \lambda_1 < \ldots < \lambda_{k-1}$ of Λ by $f(\lambda_i) = a_i$. Of course only sequences of length at most $|\Lambda|$ can be represented this way. The element of the sequence which is assigned to the element a of Λ will be called the a-placed element of the sequence.

Using this definition of a sequence we represent the functions g_0, g_1 by relations R_0, R_1 on the set $T \times \Lambda \times \Lambda \times I$, namely $R_0(t, a, b, c)$ holds iff the a-placed element of the antecedent $g_0(t)$ is a formula whose b-placed element is c. The definition of R_1 is the same using the succedent $g_1(t)$.

Therefore an **LK** proof P in the language $\mathcal{H}$ consists of a finite binary PO-set $\mathcal{T}$, a finite totally ordered set Λ, a finite set $I \subseteq \mathbf{s}(\mathcal{H})$ and relations R_0, R_1 on the set $T \times \Lambda \times \Lambda \times I$, which has to satisfy the conditions of the definition of **LK**. (Clearly these conditions are formulated in Finite Set Theory.) If the proof P has this structure we will write $P = \langle \mathcal{T}, \Lambda, I, R_1, R_2\rangle$.

Assume that τ is an interpretation of a first-order language $\mathcal{L}$, and $P = \langle \mathcal{T}, \Lambda, I, R_1, R_2\rangle$ is an **LK** proof in $\mathcal{H}$, where $\mathcal{T} = \langle T, \leq, C\rangle$. We say that P is first-order definable in τ if $T, \leq, C, \Lambda, I, R_1, R_2$ are all first-order definable in τ. This clearly implies that the sizes of the sets T, Λ, and I are smaller than $|\mathtt{universe}(\tau)|^c$, where c depends only on the lengths of the first-order formulas used in the definitions of $T, \leq, C, \Lambda, I, R_1, R_2$ in τ.

Definition 34. Assume that $\mathcal{L}$ is a first-order language, H is a theory in $\mathcal{L}$, and P is an **LK** proof in $\mathcal{L}$. We say that the proof reaches a contradiction in the theory H, if all of the formulas of the antecedent of the sequent assigned to the root of the PO-set belong to H, and the succedent of this sequent is empty.

9.2 Diagrams, and Induction Axioms

Definition 35. In this definition it is important that we consider equality as a relation symbol of each first-order language. Assume $\mathcal{K}$ is a first-order language and κ is an interpretation of $\mathcal{K}$. We extend the language $\mathcal{K}$, by a new constant symbol c_a for each element of $\mathtt{universe}(\kappa)$. The diagram of κ, which will be denoted by $\mathtt{diag}(\kappa)$, consists of all of the formulas of the type:

(15) *$R(c_{a_1}, \ldots, c_{a_k})$ for all k-ary relation symbols of $\mathcal{K}$ and for all $a_1, .., a_k \in \mathtt{universe}(\kappa)$ so that $\kappa \models R(a_1, \ldots, a_k)$, for $k = 1, 2, \ldots$*

(16) *$\neg R(c_{a_1}, \ldots, c_{a_k})$ for all k-ary relation symbols of $\mathcal{K}$ and for all $a_1, .., a_k \in \mathtt{universe}(\kappa)$ so that $\kappa \models \neg R(a_1, \ldots, a_k)$, for $k = 1, 2, \ldots$*

(17) *$c_{a_0} = f(c_{a_1}, \ldots, c_{a_k})$, for all k-ary function symbols of $\mathcal{K}$ (including constant symbols if $k = 0$) and for all $a_0, a_1, .., a_k \in \mathtt{universe}(\kappa)$ so that $\kappa \models a_0 = f(a_1, \ldots, a_k)$ for $k = 0, 1, 2, \ldots$.*

Definition 36. Suppose that the first-order language $\mathcal{L}$ is a **U**-extension of the first-order language $\mathcal{K}$. $\mathcal{L}'$ will denote the language that we get from $\mathcal{L}$ by adding the new constant symbols introduced in the definition of $\mathtt{diag}(\kappa)$. (Note that $\mathcal{L}'$ may have an infinite number of constant symbols if κ has an infinite universe.) We will always assume that the language $\mathcal{L}'$ is in κ in the sense defined earlier. Moreover, those symbols of $\mathcal{L}'$ that are not in $\mathcal{L}$, are represented in κ so that each symbol c_a of $\mathcal{L}'$ is the element a itself.

Definition 37. 1. Assume that $\mathcal{H}$ is a first-order language and $\leq$ is a binary relation in $\mathcal{H}$. The axiom-scheme of Induction for the language $\mathcal{H}$ with respect to the binary relation "$\leq$", consists of the axiom "$\leq$ is a total ordering of the universe" and for each positive integer k and first-order formula $\varphi(x_0, x_1, \ldots, x_k)$ of $\mathcal{H}$, with the only free variables $x_0, x_1, \ldots, x_k$, the sentence:

$$\forall \underline{t}, \Big(\exists x, \varphi(x, \underline{t})\Big) \rightarrow \Big(\exists y \forall z, \varphi(y, \underline{t}) \wedge \Big(\varphi(z, \underline{t}) \rightarrow y \leq z\Big)\Big)$$

where $\underline{t}$ stands for $t_1, \ldots, t_k$.

2. If the language $\mathcal{H}$, in addition to the binary relation "$\leq$", also contains a unary relation symbol **U**, then the Axiom-scheme of Induction for the language $\mathcal{H}$, with respect to "$\leq$", and restricted to **U**, consists of the statement that "$\leq$ is a total ordering of the set where **U** holds", and for each positive integer k and first-order formula $\varphi(x_0, x_1, \ldots, x_k)$ of $\mathcal{H}$, with the only free variables $x_0, x_1, \ldots, x_k$, the sentence given in the previous definition, with the modification that the $\exists x$, $\exists y$, and $\forall z$ quantifiers are all restricted to the set of elements where **U** holds.

Remark. We may also say, that the induction scheme with respect to $\leq$ is the set of statements expressing, that the universe is well-ordered by $\leq$, provided that we consider only first-order definable sets in the definition of a well-ordering. The induction scheme with respect to $\leq$ and restricted to **U** expresses the statement, that the set where **U** holds is well-ordered in the same sense.

9.3 The Statement of the Completeness Theorem

Definition 38. Assume that $\mathbf{M}$ is a model of finite set theory. If $X \in \mathbf{M}$ then $X\wr_{\mathbf{M}}$ will denote the set of all $x \in \mathbf{M}$ with $\mathbf{M} \models x \in X$. (The difference between X and $X\wr_{\mathbf{M}}$ is that X is a set in $\mathbf{M}$, while $X\wr_{\mathbf{M}}$ is a set in the world. X as a set in the world may have quite different elements.) In a similar way if $\mathbf{M} \models$ "R is a relation on the set X" then $R\wr_{\mathbf{M}}$ will be a relation in the world defined on the set $X\wr_{\mathbf{M}}$ by $R\wr_{\mathbf{M}}(x)$ iff $\mathbf{M} \models R(x)$. If $f \in \mathbf{M}$ is a function in $\mathbf{M}$ then $f\wr_{\mathbf{M}}$ will denote the unique function g with $\mathtt{domain}(f)\wr_{\mathbf{M}} = \mathtt{domain}(g)$ and for all $x, y \in \mathbf{M}$, $y = g(x)$ iff $\mathbf{M} \models y = f(x)$. We define the same notion for relations and functions with arities larger than 1 in a similar way.

Definition 39. If $\mathcal{L}$ is a language so that $\mathtt{Symb}(\mathcal{L})$ is a hereditarily finite set then we will say that $\mathcal{L}$ is a hereditarily finite language. If $\mathcal{L}$ is such a language and $\mathbf{M}$ is a model of Finite Set Theory then $\mathcal{L}$ is also a language in $\mathbf{M}$, (since we assumed that such a model always contains the hereditarily finite sets). Suppose now that $\sigma \in \mathbf{M}$ so that $\mathbf{M} \models$"σ is a interpretation of $\mathcal{L}$". Then $\sigma\wr_{\mathbf{M}}$ will denote the unique interpretation τ of $\mathcal{L}$ in the world so that $\mathbf{M} \models$"X is the universe of σ" implies $X\wr_{\mathbf{M}} = \mathtt{universe}(\tau)$, and for each relation or function symbol Q of $\mathcal{L}$ we have $\mathbf{M} \models Y = \sigma(Q)$ iff $Y\wr_{\mathbf{M}} = \tau(Q)$.

The statement of the following lemma is an immediate consequence of the definitions.

Lemma 40. *Assume that $\mathcal{K}$ is a hereditarily finite language $\mathbf{M}$ is a model of finite set theory, and κ is a interpretation of $\mathcal{K}$ which is first-order definable in $\mathbf{M}$. Suppose further that there exists an $X \in \mathbf{M}$ so that $\mathtt{universe}(\kappa) \subseteq X\wr_{\mathbf{M}}$. Then there exists a unique element $\bar{\kappa} \in \mathbf{M}$ so that $\kappa = \bar{\kappa}\wr_{\mathbf{M}}$.*

Definition 41. Under the assumption of Lemma 40, the unique interpretation $\bar{\kappa}$, whose existence is stated in the lemma, will be denoted by $\bar{\kappa}^{(\mathbf{M})}$ and we will omit the superscript if the choice of $\mathbf{M}$ is clear from the context. Suppose $\mathcal{L}$ is a hereditarily finite language as well, and it is a $\mathbf{U}$-extension of $\mathcal{K}$. If an interpretation κ is fixed then we have defined the language $\mathcal{L}'$. Suppose that κ satisfies the assumptions of Lemma 40. In this case we may define $\mathcal{L}'$ in $\mathbf{M}$ with respect to $\bar{\kappa}$. This language will be denoted by $\bar{\mathcal{L}}'$.

Notation. Under the assumptions of the previous definition if we write a statement $\mathbf{M} \models \varphi$, then in the formula φ we will write κ instead of $\bar{\kappa}$ and we will write $\mathcal{L}'$ instead of $\bar{\mathcal{L}}'$ to make the statement more transparent.

Remark. In the formulation of the theorem we will use the fact, that in a model of Finite Set Theory each natural number q is the set of all natural numbers less than q.

Theorem 42. *Assume that $\mathcal{L}$, $\mathcal{K}$ are hereditarily finite first-order languages, $\mathcal{L}$ is a $\mathbf{U}$-extension of $\mathcal{K}$, $\mathbf{M}$ is a model of Finite Set Theory, and κ is an interpretation of $\mathcal{K}$ so that the following conditions are satisfied:*

(18) $\mathtt{universe}(\kappa)$ *is a countable set.*

(19) κ *is first-order definable in* $\mathbf{M}$.

(20) *There exists a* q *so that* $\mathbf{M} \models$ *"*q *is a natural number and* $\mathtt{universe}(\kappa) = q$*".*

(21) *The language* $\mathcal{K}$ *contains a binary relation "*$\leq$*" and* $\kappa(\leq)$ *is the natural ordering of the natural numbers in* $\mathbf{M}$ *on* $\mathtt{universe}(\kappa)$.

(22) *If we restrict the* $+$ *and* $\times$ *operations, of the natural numbers in* $\mathbf{M}$ *to the set* $\mathtt{universe}(\kappa)$*, then both of them, as ternary relations, are first-order definable in* κ.

Assume further that G *is a theory in* $\mathcal{L}$ *so that there is a set* $B \in \mathbf{M}$ *with the property that* $G = \mathtt{standard}(B)$*, and in* G *all of the statements are provable, which belong to the the Axiom-scheme of Induction for the language* $\mathcal{L}$ *with respect to the relation* $\leq_{\mathcal{K}}$*, and restricted to* $\mathbf{U}$*. Then, the theory* G *has a model over* κ *iff there exists a (standard) natural number* d *and a* $P \in \mathbf{M}$*, so that the following three requirements are met*

(23) $\mathbf{M} \models$ *"*P *is a proof in* $\mathbf{LK}$ *in the language* $\mathcal{L}'$*, and* P *reaches a contradiction in the theory* $G_d + \mathtt{diag}(\kappa)$*"*

(24) $\mathbf{M} \models$ *"every formula contained in* P *is of length at most* d*".*

(25) $\mathbf{M} \models$ *"*P *is first-order definable in* κ *with formulas of length less than* d.

9.4 The Sketch of the Proof of the Completeness Theorem

In the proof of the theorem we use the following lemma.

Lemma 43. *Suppose that* $\mathcal{M}$ *is a structure with a total ordering of the universe with a smallest element* 0 *and largest element* 1*. Assume further that each nonempty first-order definable subset of the universe has a smallest element, and each nonzero element has a predecessor. Then each nonempty partially ordered set, which is first-order definable in* $\mathcal{M}$*, has a minimal element.*

Proof. The assumptions of the lemma imply that each nonempty first-order definable subset U of the universe has a largest element in the total ordering. Indeed if U does not contain the largest element 1 of the universe then let Y be the set of those elements of the universe which are strictly larger than each element of U. According to the assumptions of the lemma Y has a smallest element y and it has a predecessor u. Clearly u is the largest element of U.

Assume that the statement is not true and P is a first-order definable partially ordered set without a minimal element. $\leq_P$ denotes the partial ordering of P and $\leq$ denotes the total ordering of the universe. Let X be the set of all elements x of P with the following property:

(26) *for all* $y \in P$ *there exists a* $z \in P$ *so that* $z <_P y$ *and* $z < x$

X is nonempty since, the indirect assumption implies that, 1 is in X. Let v be the smallest element of X with respect to the total ordering. v cannot be 0, (the smallest element of the total ordering), since 0 does not satisfy condition (26). By the assumptions of the lemma, v has a predecessor u. We claim that in contradiction to the minimality of v in X, the element u also satisfies condition (26) with $x \to u$. Indeed, let $y \in P$ be arbitrary. Since $v \in X$, there is a $z \in P$ so that $z <_P y$ and $z < v$. If $z \neq u$ then $z < u$ and so condition (26) is satisfied. Assume now that $z = u$. Using again that $v \in X$ we get that there exists a $t \in P$ with $t <_P z = u$ and $t < x$. Now $t <_P u$ implies that $t \neq u$, so by $t < v$ we have $t < u$. On the other hand $t <_P z <_P< y$ and so $t <_P< y$. Therefore condition (26) is satisfied by $x \to u$ and so $u \in X$ in contradiction to the minimality of v. *Q.E.D.*(Lemma 43)

Sketch of the proof of Theorem 42. In this sketch we sometimes will refer to the sets X and $X\wr_{\mathbf{M}}$ with the same symbol. The choice of the right set however will be clear from the context.

Assume first that there is an **LK** proof P in **M** which shows a contradiction in the theory $G_d + \mathtt{diag}(\kappa)$ and first-order definable in κ with formulas of length at most d, but contrary to the assertion of the theorem there is a model **G** of $G + \mathtt{diag}(\kappa)$ over κ. (Note that **G** is not necessarily first-order definable in **M**.) First we note that, since the proof P is first-order definable in $\bar{\kappa}$ with a formula of standard size, there is a standard integer i so that the set of nodes of the binary PO-set T is a subset of $D = (\mathtt{universe}(\kappa))^i$. D is a totally ordered set with respect to the lexicographic ordering. $\mathtt{universe}(\kappa)$ is well ordered in **G** with respect to first-order definable sets, since G contains the induction scheme restricted to **U**. Therefore D is also well-ordered in **G** in the same sense. Clearly **G** also has a largest and smallest element, and every element of D has both a predecessor and a successor. Applying Lemma 43 we get that in every first-order definable partially ordered set $X \subseteq D$, $X \in \mathbf{G}$, has the property that every subset which is first-order definable in **G** has a smallest element.

Therefore the binary *PO*-set T satisfies this condition as well. Since the lengths of the formulas of the proof P remain below a standard bound d we have that the truth values of these formulas in the model **G** can be defined in **G**, moreover the function which assigns to each formulas its truth value is first-order definable in **G**. Therefore we may assign a truth value to each sequent of the proof namely if $g_{0(t)}$ the antecedent contains the set of formulas Γ and $g_1(t)$ the succedent contains the set of formulas Θ then the value assigned to t will be $\tau(t) \equiv \bigwedge_{\gamma \in \Gamma} \tau(\gamma) \to \bigvee_{\vartheta \in \Theta} \tau(\vartheta)$. It is easy to see that the function τ is first-order definable in **G**. By our assumptions about T, the value assigned by τ to the root of the tree is "$FALSE$" while the values assigned by τ to the maximal nodes are all "TRUE". Since τ is first-order definable there is a maximal node of m of T so that $\tau(m)$ is "FALSE". m cannot be a maximal node in the whole set T, since there τ is "TRUE". Therefore the sequents assigned to m and to its successors by the pair $\langle g_0, g_1 \rangle$ must satisfy one of the inference rules. It is easy

to show that this is impossible if $\tau(m)$ is "$FALSE$" but $\tau(x)$ is true for all of the successors of m. This completes the proof of the theorem in one direction.

Assume now that the theory $H = G + \mathtt{diag}(\kappa)$ has no model over κ. First note that it is sufficient to prove the theorem for the special case when the theory H contains only prenex universal formulas. (We will say that such a theory is universal.) Indeed introducing Skolem functions as new function symbols, as it was done in the proof of the generalized Compactness Theorem we get an equivalent universal theory. It is not clear whether, in general, a **LK** proof showing a contradiction in the universal theory can be transformed, in a first-order definable way, into a **LK** proof in the original theory. In our case however, this is possible. We will tell more about this later.

We suppose now that the theory H is universal and it has no model over κ. In the proof of the generalized Compactness Theorem we have seen that if in **M** it is true that Player 2 has a winning strategy in the game $\mathcal{G}(\psi, k_0)$, for a nonstandard k_0, then H has a model over κ. Therefore our present assumption implies that Player 1 has a winning strategy in all of these games. Therefore the smallest k, so that Player 1 has a winning strategy in game $\mathcal{G}(\psi, k)$, is a standard natural number. Suppose a standard k is fixed so that Player 1 has a winning strategy in the game $\mathcal{G}(\psi, k)$.

This game is of the following type. The players alternately have to tell elements a_i (perhaps more than one at a time) of $\mathtt{universe}(\kappa)$. The total number of these elements remain standard. Player 2 essentially has to evaluate skolem functions at places d_j which depend on the elements a_i which already occurred in the game. The evaluation must be done only under the assumption that the value of the skolem function is in $\mathtt{universe}(\kappa)$. The elements d_j are given as terms of elements a_i which has occurred earlier. At the end Player 1 wins if some of the universal axioms of H turned into propositional formulas at evaluating the quantified variables at various terms (without variables), and some of the declarations of Player 2 about the values of the Skolem functions are contradictory (in the propositional calculus) and there is such a contradiction of length at most k.

If X is the sequence of moves in the game, then let $H_{\mathcal{G}}(X)$ be the theory that we get from H if we add to it all of the propositional statements about the values of Skolem functions made by Player 2 provided that X was the sequence of moves. Our assumptions imply that Player 1 has a strategy so that there is an **LK** proof (satisfying the conditions of the theorem) that shows a contradiction in the theory $H_{\mathcal{G}}(X)$. Indeed the mentioned propositional contradiction can be easily transformed into such a proof.

Using induction we show, that for each $i = 0, 1, \ldots, k$, there is a similar game $\mathcal{G}'$ so that the length of the game is at most $k - i$ and Player 1 has a strategy which guarantees that there is an **LK** proof P which shows a contradiction in the theory $H_{\mathcal{G}'}(X)$. At the end, with a game of length 0 we get an **LK** proof which shows a contradiction in the theory H. This inductive proof leads to a recursive construction of the **LK** proof P. With the help of the fact that the number of recursive steps is a standard integer we can prove that P satisfies all of our requirements, in particular it is first-order definable in **M**, and the length

of the formulas in it remain below a standard bound. Following this recursion we may also transform a proof in the universal theory to a proof in the original theory.

We do not describe here the inductive step, only note that it involves the use of the cut rule n times where $n = |\mathtt{universe}(\kappa)|$.

References

1. M. Ajtai, Σ^1_1-*formulae on finite structures*, Annals of Pure and Applied Logic, 24 (1983), 1-48.
2. M. Ajtai, *The Complexity of the Pigeonhole Principle*, Combinatorica 14 (4), (1994) 417-433. 1993.
3. P. Beame, R. Impagliazzo, J. Krajicek, T. Pitassi, P.Pudlák and A. Woods, *Exponential Lower Bound for the Pigeonhole Principle* (extended abstract), in: Proc. ACM Symp. on Theory of Computing, ACM Press, (1992), pp.200-220.
4. M. Furst, J.B. Saxe and M. Sipser, *Parity, Circuits and the Polynomial Time Hierarchy*, Mathematical Systems Theory, (17):13-17, 1984.
5. G. Gentzen *Unterschungen über das logische Schliessen*, Matematische Zeitschrift 39 (1934) 176-210, 405-431
6. J. Hastad *Almost optimal lower bounds for small depth circuits* Proc. 18th Annu. ACM Symp. Theory Computing 6-20 (1986).
7. T. Jech, *Axiomatic Set Theory*, Academic Press, San Diego, 1978.
8. J. Krajicek, *Forcing with random variables and proof complexity*, eds. A.Beckmann, U.Berger, B.Löwe, and J.V.Tucker: Logical Approaches to Computational Barriers, 2nd Conference on Computability in Europe, CiE 2006, Swansea, UK, July 2006, Lecture Notes in Computer Science, Springer, (2006), pp.277-278.
9. A. Máté, *Nondeterministic polynomial-time computation and models of Arithmetic*, Journal of the ACM, Vol. 37, (1), January 1990, pp. 175-193.
10. G. Takeuti, *Proof Theory*, North-Holland, Studies in Logic and the Foundations of Mathematics, Vol. 81, Second edition, 1987.
11. A. Yao *Separating the polynomial time hierarchy by oracles* Proc. 26th Annu. IEEE Symp. Found. Comp. Sci. 1-10 (1985).

Approximation Algorithms for 3D Orthogonal Knapsack*

Florian Diedrich[1], Rolf Harren[2], Klaus Jansen[1], Ralf Thöle[1], and Henning Thomas[1]

[1] Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24098 Kiel, Germany
[2] Fachbereich Informatik, Universität Dortmund, Otto-Hahn-Str. 14, 44227 Dortmund, Germany

Abstract. We study non-overlapping axis-parallel packings of 3D boxes with profits into a dedicated bigger box where rotation is forbidden; we wish to maximize the total profit. Since this optimization problem is NP-hard, we focus on approximation algorithms. We obtain fast and simple algorithms with approximation ratios $9+\epsilon$ and $8+\epsilon$ as well as an algorithm with approximation ratio $7+\epsilon$ that uses more sophisticated techniques; these are the smallest approximation ratios known for this problem. *Topics:* Algorithms, computational and structural complexity.

1 Introduction

Given a list $L = \{R_1, \ldots, R_n\}$ of boxes with sizes $R_i = (x_i, y_i, z_i)$ and positive profits p_i for each $i \in \{1, \ldots, n\}$ and a dedicated box $Q = (a, b, c)$, we study non-overlapping axis-parallel packings of sublists of L into Q which we call *feasible*. For simplicity call Q a *bin*. We wish to select a sublist that permits a packing and maximizes the profit. This problem will be called the *orthogonal three-dimensional knapsack problem* or *OKP-3* for short and we denote the optimal profit by OPT. It is a natural generalization of the knapsack problem (KP) which is known to be NP-hard. This makes an exact algorithm with a polynomial worst-case runtime bound impossible unless P = NP holds. W.l.o.g. we assume $a = b = c = 1$ and that each $R_i \in L$ can be packed by otherwise removing *infeasible* boxes and scaling in $O(n)$ time.

Related problems. Different geometrically constrained two- and three-dimensional packing problems were studied, resulting in three main directions. In *strip packing* the target area is a strip of infinite height; the objective is to minimize

* Research supported in part by DFG Project, "Entwicklung und Analyse von Approximativen Algorithmen für Gemischte und Verallgemeinerte Packungs- und Überdeckungsprobleme, JA 612/10-1", in part by the German Academic Exchange Service DAAD, in part by project AEOLUS, EU contract number 015964, and in part by a grant "DAAD Doktorandenstipendium" of the German Academic Exchange Service DAAD. Part of this work was done while visiting the ID-IMAG, ENSIMAG, Grenoble.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 34–45, 2007.
© Springer-Verlag Berlin Heidelberg 2007

the height of the packing. For the 2D case, [22] yields an approximation ratio of 5/2; in [1], an asymptotic approximation ratio of 5/4 was obtained. The best known absolute approximation ratio of 2 was obtained with different techniques in [21,23]. In [13], an asymptotic fully polynomial time approximation scheme (AFPTAS – see [24] for a definition) was presented. For the 3D case, research has focused mainly on the asymptotic approximation ratio [20]. An asymptotic ratio of $2+\epsilon$ was obtained in [10]; this was improved to 1.691 by Bansal et al. [3]. In [17] the *on-line* version was studied, resulting in a competitive ratio of 29/10. In *bin packing* the objective is to minimize the number of identical bins. For the 1D case, an asymptotic polynomial time approximation scheme (APTAS – see [24]) was presented in [7], while in [18] the currently best known asymptotic approximation ratio of 11/9 for the popular FFD algorithm is proved. For the 2D case, an asymptotic approximation ratio of 1.691 was obtained in [4]. In [2] it was proved that d-dimensional bin packing does not admit an APTAS for $d \geq 2$ and therefore no FPTAS, but an APTAS for packing d-dimensional cubes into the minimum number of unit cubes was presented. In the *knapsack* scenario the number of bins is a fixed constant [5], usually 1. For the 2D case, [11] yields an approximation ratio of $2+\epsilon$. Classical 1D knapsack problems are relatively well understood, see [12,19] for surveys. Although the problems are closely related, results cannot be transferred directly. One main difference between bin/strip packing and knapsack packing is that in the first setting all boxes of the instance must be packed but in the latter a selection of items is needed.

Previous results and applications. Harren [8] obtained a ratio of $9/8+\epsilon$ for the special case of packing cubes into a cube and proved the APX-completeness of the general case [9]. A *cutting stock* application is cutting blocks with given profits from larger pieces of material to maximize the profit; another application is the problem of selecting boxes to be transported in a container. Besides these, the problem is motivated from multiprocessor scheduling on grid topology. In this perspective, for a time slice of fixed duration, a set of jobs to be executed must be chosen and each job requires a subgrid of prespecified rectangular shape. For a special case of this application, in [25] an on-line algorithm is presented; See [6] for a study of similar problems.

New results. Our contribution is a fast and simple $(9+\epsilon)$-approximation algorithm based on strip packing (Section 2) which is refined to an $(8+\epsilon)$-approximation algorithm in Section 3. Both of these have practical running times. In Section 4 we obtain a $(7+\epsilon)$-approximation algorithm using more costly techniques before concluding with open problems in Section 5.

2 An Algorithm Based on Strip Packing

We approximately solve a relaxation by selecting $L' \subseteq L$ that is at least near-optimal and has a total volume of at most 1 which is partitioned into 9 sublists. For each of these a packing into the bin will be generated. Out of these one with maximum profit is chosen, resulting in a $(9+\epsilon)$-approximation algorithm.

More precisely L' will be packed into a strip $[0,1] \times [0,1] \times [0,\infty)$ by a level-oriented algorithm, improving a result from [16]. We partition the strip into packings of sublists of L' and among these return one with maximum profit. For each box R_i the rectangle (x_i, y_i) is called the *base rectangle* of R_i, denoted as $br(R_i)$. Such a rectangle (x_i, y_i) is called *big* $:\Leftrightarrow x_i \in (1/2, 1] \wedge y_i \in (1/2, 1]$, *long* $:\Leftrightarrow x_i \in (1/2, 1] \wedge y_i \in (0, 1/2]$, *wide* $:\Leftrightarrow x_i \in (0, 1/2] \wedge y_i \in (1/2, 1]$, and *small* $:\Leftrightarrow x_i \in (0, 1/2] \wedge y_i \in (0, 1/2]$. For each list L of boxes use $V(L) := \sum_{R_i \in L} x_i y_i z_i$ to denote the total volume of L and for each list L of rectangles $r_i = (x_i, y_i)$ use $A(L) := \sum_{r_i \in L} x_i y_i$ to denote the total area of L. Furthermore, $P(L) := \sum_{R_i \in L} p_i$ denotes the total profit of L. Finally, for each list L of boxes use $H(L)$ to denote the height of a packing of L where the packing itself will be clear from the context. We use the following theorem from [11] which is a refinement of the main result from [23].

Theorem 1. *Let L be a list of n rectangles such that $A(L) \leq 1/2$ holds and no long rectangles or no wide rectangles occur in L. Then L permits a feasible packing into the unit square which can be generated in time $O(n \log^2 n / \log\log n)$.*

First we apply the modified strip packing algorithm, then we construct the partition of the strip. The strip packing algorithm uses Theorem 1 to obtain an *area guarantee* for each but the last level, improving a result from [16].

Algorithm A

1. Partition L into two sublists $L_1 := \{R_i | br(R_i) \text{ is long}\}$ and $L_2 := L \setminus L_1$. W.l.o.g. let $L_1 = \{R_1, \ldots, R_m\}$ and $L_2 = \{R_{m+1}, \ldots, R_n\}$.
2. Generate the packing for L_1 as follows.
   2.1. Find the boxes R_i in L_1 for which the area of $br(R_i)$ is greater than $1/4$ which are $R_{p+1}, \ldots, R_m$ w.l.o.g. Stack these on top of one another in direction z, each on its own level.
   2.2. Sort the remaining boxes $R_1, \ldots, R_p$ in non-increasing order of z_i, resulting in a list L'_1.
   2.3. Partition L'_1 into consecutive sublists $L''_1, \ldots, L''_v$ where the total base area of each sublist is as close to $1/2$ as possible but not greater. Pack each of these sublists on a level by itself using Theorem 1. Stack all of these levels on top of one another in direction z.
3. Generate the packing for L_2 in a similar way as for L_1 by Theorem 1. The resulting steps are called Steps 3.1 – 3.3.
4. Concatenate the packings of L_1 and L_2 to obtain a packing of L.

Theorem 2. *For each list L of n boxes Algorithm A generates a packing of height at most $4V(L) + Z_1 + Z_2$ where Z_1 and Z_2 are the heights of the first levels generated in Steps 2.3 and 3.3. The construction can be carried out in time $O(n \log^2 n / \log\log n)$.*

The proof is omitted due to page limitations; the result can be obtained by replacing the area bound 7/32 by 1/4 in the proof of Theorem 4 from [16]. The second part is a partition applied to the output of Algorithm A; see Figure 1.

Algorithm B

1. Set $\delta := \epsilon/(9+\epsilon)$. Use an FPTAS for KP from [12,14] to select $L' \subseteq L$ such that $V(L') \leq 1$ and $P(L') \geq (1-\delta)\mathrm{OPT}$ holds, where OPT denotes the optimum of the generated KP instance.
2. Use Algorithm A to generate a packing of L' into the strip but separate the first levels generated in Steps 2.3 and 3.3. Pack these into a bin each.
3. By Theorem 2 the remaining strip has a height of at most $4V(L') \leq 4$. Consider the three cutting unit squares $[0,1] \times [0,1] \times \{i\}$ for $i \in \{1,2,3\}$. Generate a partition of the region $[0,1] \times [0,1] \times [0,4]$ into 7 subsets, namely 4 subsets which are each positioned in the regions $[0,1] \times [0,1] \times [i-1,i]$ for $i \in \{1,\ldots,4\}$ but not intersecting any of the unit squares and 3 subsets of boxes which each intersect with one of the three cutting unit squares.
4. Out of the sets generated in Steps 2 and 3 return one with maximum profit.

Each set generated in Steps 2 and 3 permits a feasible packing into the unit cube which is available as a byproduct of Algorithm A. L' is partitioned into at most 9 subsets by Algorithm B, as illustrated in Figure 1.

Theorem 3. *Algorithm B is a $(9+\epsilon)$-approximation algorithm for OKP-3 with running time $O(\mathrm{T_{KP}}(n,\epsilon) + n\log^2 n/\log\log n)$, where $\mathrm{T_{KP}}(n,\epsilon)$ is the running time of the FPTAS used for KP from [12,14]. Furthermore this bound is tight.*

Proof. Clearly $9+\epsilon$ is an upper bound for the ratio and the running time is dominated by solving the knapsack instance and by Algorithm A. For the following instance this bound can be attained. We have 10 boxes $R_1 := (1/2, 1/2, 2/15)$, $R_2 := (3/4, 1/3, 2/15)$, $R_3 := (1, 2/7, 3/4)$, $R_4 := \ldots := R_7 := (1, 2/7, 1/2)$, $R_8 := (1, 2/7, 1/4+2/15)$, $R_9 := (1, 2/7, 2/15)$ and $R_{10} := (1,1,1)$. Furthermore $p_1 := \ldots := p_9 := 1/9 - \epsilon/[9(9+\epsilon)]$ and $p_{10} := 1$. Let $S_1 := \{R_1, \ldots, R_9\}$ and $S_2 := \{R_{10}\}$. It is clear that S_2 is an optimal solution; elementary calculation shows $V(S_1) = 1$ and $P(S_1) = 1-\delta$, hence S_1 may be selected in Step 1 of Algorithm B. Applying Algorithm B and assuming that the boxes are stacked in increasing order of index in Step 2.1 of Algorithm A, we obtain 9 bins each containing an item with profit $1/(9+\epsilon)$. □

Note that only the subset that is returned needs to be packed level-wise using the algorithm from Theorem 1 while the discarded subsets need not to be arranged.

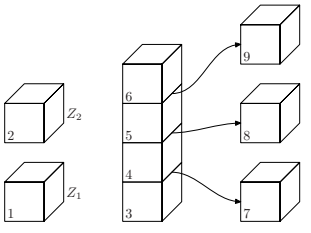


Fig. 1. At most 9 bins are generated by Algorithm B

Algorithm B can be used to solve the special cases where we wish to maximize the number of selected boxes or the volume by setting $p_i := 1$ or $p_i := x_i y_i z_i$ for each $i \in \{1, \ldots, n\}$. This also holds for the other two algorithms that we present. In [14] approximation algorithms for various knapsack problems are found. Using these, Algorithm B can be generalized by replacing the KP solver in Step 1, yielding algorithms for *unbounded* OKP-3 and *multiple-choice* OKP-3; see [14] for notions and details. Algorithm B can be modified to yield a ratio of 18 with a much better running time by using a 2-approximation algorithm for classical KP. Call R_i *small* $:\Leftrightarrow x_i \in (0, 1/2] \wedge y_i \in (0, 1/2] \wedge z_i \in (0, 1/2]$. We obtain a criterion for packability of a list of boxes; the proof is omitted due to space restrictions.

Lemma 1. *Each list L of small boxes for which $V(L) \leq 1/8$ holds can be packed into the unit cube in time $O(n \log^2 n / \log \log n)$.*

3 A Refined Construction

In Algorithm A and the proof of Theorem 2, the area bound 1/2 from Theorem 1 was used. We separate boxes with base area greater than 1/4, resulting in the area guarantee $1/2 - 1/4 = 1/4$ for each level generated in Steps 2.2 and 2.3 except the last ones. The height bound can be improved if this area guarantee is improved. We have arbitrarily chosen direction z to be the axis for level generation, but any direction $d \in \{x, y, z\}$ will do. The two levels of height Z_1 and Z_2 are the result of partitioning the instance. We study the packing of small boxes more closely; Algorithm C is applied only to lists of small boxes.

Algorithm C

1. Find the boxes R_i in L for which the area of $br(R_i)$ is greater than 1/10 which are $R_1, \ldots, R_m$ w.l.o.g. Sort these in non-increasing order of z_i, resulting in a list L_1. Arrange these in groups of 4 boxes each, except for the last group. Each group can be put on a seperate level by placing the boxes into the corners of the level. Stack these levels on top of one another in direction z.
2. Sort the remaining boxes $R_{m+1}, \ldots, R_n$ in non-increasing order of z_i, resulting in a list L_2.
3. Partition L_2 into consecutive sublists $L_1'', \ldots, L_v''$ where the total base area of each sublist is as close to 1/2 as possible but not greater. Pack each of these sublists on a level by itself using Theorem 1. Stack all of these levels on top of one another in direction z.
4. Concatenate the packings of L_1 and L_2 to obtain a packing of L.

Note that we formed two groups and obtain an area guarantee of 2/5 for each layer except the last ones generated in Steps 1 and 3. To avoid confusion we point out that the area guarantee does not hold for the *last* generated layers, while the summands Z_1 and Z_2 in Theorem 2 are the heights of the respective *first* layers. Similar to the proof of Theorem 2, we obtain the following results.

Theorem 4. *For each list L of n small boxes Algorithm C generates a feasible packing of height at most $5/2V(L) + Z_1 + Z_2$ where $Z_1 \leq 1/2$ and $Z_2 \leq 1/2$ are the heights of the first levels generated in Steps 1 and 3. The construction can be carried out in time $O(n \log^2 n / \log\log n)$.*

Lemma 2. *Each list L of n small boxes with $V(L) \leq 1$ permits a feasible packing into at most 5 bins. The construction can be carried out algorithmically in time $O(n \log^2 n / \log\log n)$; the bound of 5 is tight for the used construction.*

Proof. Use Algorithm C to arrange L in a strip, but separate the first levels generated in Steps 1 and 3. Since L contains only small boxes, these two levels can be packed together into a bin. By Theorem 4, the remaining strip has a height of at most $5/2$. Consider the two cutting unit squares $[0,1]\times[0,1]\times\{i\}$ for $i \in \{1,2\}$. Generate a partition of the region $[0,1] \times [0,1] \times [0,5/2]$ into 5 subsets, namely first 3 subsets which are each positioned in the regions $[0,1] \times [0,1] \times [i-1,i]$ for $i \in \{1,2\}$ as well as the region $[0,1]\times[0,1]\times[2,5/2]$ but not intersecting any of the unit squares, and furthermore 2 subsets of boxes which each intersect with one of the two cutting unit squares. The first three sets can be packed into one bin each. Since L contains only small boxes, the last two sets can be arranged together into one additional bin. We have at most 5 bins; see Figure 2. The running time is dominated by Algorithm C and thus bounded by $O(n \log^2 n / \log\log n)$. To show the tightness of the bound let $\gamma := 1/500$ and consider the instance L consisting of $R_1 := \ldots := R_{29} := (1/2, 1/5 + \gamma, 1/3 + \gamma)$, $R_{30} := (\gamma, \gamma, 1/2)$, $R_{31} := \ldots := R_{33} := (1/2, 1/5, \gamma)$ and $R_{34} := (1/2, 1/5 - 2\gamma^2, \gamma)$. Note that $V(L) = 29/30 + 122/15\gamma + 15\gamma^2 - \gamma^3 < 29/30 + 1/60$. Application of Algorithm C results in 9 layers with height greater than $1/3$, which means that the layers cannot be arranged in less than 5 bins. □

Now we refine Algorithm B to yield a better approximation ratio, but we need to solve a stronger relaxation. For any direction $d \in \{x, y, z\}$ a box R_i is called *d-big* $:\Leftrightarrow d_i \in (1/2, 1]$ and we use X, Y and Z to denote the set of boxes that are d-big for the corresponding direction. Any box that is d-big for every direction $d \in \{x, y, z\}$ will be called a *big* box.

Lemma 3. *Let L be a list of n boxes in which no small boxes and at most 3 big boxes occur. Then L can be partitioned into sets X', Y' and Z' in time $O(n)$, such that each of these contains at most one big box and the x-projections of boxes in X', the y-projections of boxes in Y' and the z-projections of boxes in Z' contain no long or no wide rectangles.*

Proof. Remove the at most 3 big boxes from L and distribute them in X', Y' and Z' such that in each of these sets at most one big rectangle occurs. Set

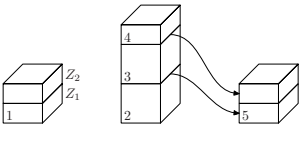


Fig. 2. The small boxes can be packed into at most 5 bins

$X' := X' \cup \{R_i \in L | x_i > 1/2, z_i \leq 1/2\}$, $Y' := Y' \cup \{R_i \in L | y_i > 1/2, x_i \leq 1/2\}$ and finally $Z' := Z' \cup \{R_i \in L | z_i > 1/2, y_i \leq 1/2\}$ to obtain the claim. □

To avoid repetition, we enumerate the cases in the analysis only.

Algorithm D

1. Set $\delta := \epsilon/(8+\epsilon)$. Use a PTAS for non-geometric 4D KP from [12,14] to select $L' \subseteq L$ such that $P(L') \geq (1-\delta)\mathrm{OPT}$ where OPT denotes the optimum of the integral linear program

$$\text{maximize } \sum_{i=1}^{n} p_i R_i \text{ subject to } R \in P$$

   where R_i is an indicator variable for the box of the same name and the polytope P of nonnegative integers is defined by the constraints

$$\sum_{i=1}^{n} x_i y_i z_i R_i \leq 1, \quad \sum_{R_i \in X} y_i z_i R_i \leq 1, \quad \sum_{R_i \in Y} x_i z_i R_i \leq 1, \quad \sum_{R_i \in Z} x_i y_i R_i \leq 1.$$

2. Partition L' into at most 8 subsets which permit a feasible packing as described below. Out of these, return one with maximum profit.

Theorem 5. *Algorithm D is an $(8+\epsilon)$-approximation algorithm for OKP-3 with running time $O(\mathrm{T}_{4\mathrm{DKP}}(n,\epsilon) + n\log^2 n/\log\log n)$, where $\mathrm{T}_{4\mathrm{DKP}}(n,\epsilon)$ is the running time of the PTAS used for 4D KP from [12,14]. Furthermore this bound is tight.*

Proof. We have not imposed a bound on the number of big boxes in the relaxation; due to the area conditions there are at most 3 big boxes in the selected set. *Case 1:* There is a direction $d \in \{x, y, z\}$ such that the d-projection area of all d-big boxes in L' is larger than or equal to 1/2. In this case all d-big boxes can be packed into at most 3 bins with a construction from [11], which can be carried out in time $O(n\log^2 n/\log\log n)$, resulting in a volume of at least 1/4 being packed. The total volume of the remaining boxes is bounded by 3/4 and each remaining box has a d-height of at most 1/2. We apply Algorithm A in direction d which results in a strip of d-height at most 3 and two additional levels of d-height at most 1/2 each. All of these sets can be packed into at most 5 bins, generating at most 8 bins in total. *Case 2:* For all $d \in \{x, y, z\}$ the total projection area of all d-big boxes is smaller than 1/2. By Lemma 3 we partition the set $\{R_i \in L' | R_i \text{ is not small}\}$ into sets X', Y' and Z' such that the total projection area of X', Y' and Z' for the corresponding direction is not greater than 1/2 and the x-projections of boxes in X', the y-projections of boxes in Y' and the z-projection of boxes in Z' contain no long or no wide rectangles, respectively, and each of these sets contains at most one big box. By Theorem 1 the sets X', Y' and Z' can be packed into at most one bin each, resulting in at most 3 bins in total. Let S denote the set of small boxes;

these are not yet packed. Clearly $V(S) \leq 1$ holds, so by Lemma 2 the set S can be packed into at most 5 bins, which results in at most 8 bins in total. The runtime bound follows from the fact that we can distinguish between the two cases in time $O(n)$. For the tightness of the bound, consider the instance L in which $R_1, \ldots, R_{34}$ are as in the proof of Lemma 2, $R_{35} := (1, 1, 1/180)$, $R_{36} := (1, 1/180, 1)$, $R_{37} := (1/180, 1, 1)$, and $R_{38} := (1, 1, 1)$. The profits are defined by $p_i := 1/[9(8+\epsilon)]$ for $i \in \{1, \ldots, 4, 30, \ldots, 34\}$, $p_i := 1/[8(8+\epsilon)]$ for $i \in \{5, \ldots, 28\}$, $p_i := 1/(8+\epsilon)$ for $i \in \{29, 35, 36, 37\}$ and $p_{38} := 1$. Let $S_1 := L \backslash \{R_{38}\}$ and $S_2 := \{R_{38}\}$. Since $P(S_1) = 8/(8+\epsilon) = (1-\delta) < 1 = P(S_2)$, S_2 is an optimal solution. Elementary calculation verifies that S_1 may be chosen in Step 1 of Algorithm D. Application of Algorithm D leads to *Case 2* in the analysis above, where $X' = \{R_{35}\}$, $Y' = \{R_{37}\}$ and $Z' = \{R_{36}\}$. Each of these sets is packed into a separate bin. The remaining items are small and are packed into 5 bins by the proof of Lemma 2. In total, 8 bins are generated; the profits are chosen such that each bin yields a profit of exactly $1/(8+\epsilon)$. □

4 Enumerations and a Shifting Technique

The algorithms above generate cutting areas in the strip, resulting in subsets that have to be re-packed. We permit further loss of profit by discarding more boxes to remove inconvenient layers; the loss will be suitably bounded. The improvement will be at the cost of a considerably larger running time due to a large enumeration. Since the running time is not practicable anyway we omit the run time analysis of this approach. First we remove sets intersecting the cutting areas and the additional layers with a shifting technique.

Lemma 4. *Let $L = \{R_1, \ldots, R_n\}$ be a list of boxes with $z_i \leq \epsilon$ for each $R_i \in L$. Suppose L admits a packing into a strip of height at most h and let m be a positive integer. Then we can create m gaps of shape $[0,1] \times [0,1] \times [0,\epsilon]$ in the packing by deleting boxes such that for the remaining list $L' \subseteq L$ the inequality $P(L') \geq (1 - 2(m+1)\epsilon/h)P(L)$ holds. The construction can be done in time polynomial in n.*

Proof. We partition the strip into regions of height ϵ and eventually one region of smaller height. More precisely we define $p := \lceil h/\epsilon \rceil$ and partition the strip of height h into p regions $S_1, \ldots, S_p$ of shape $[0,1] \times [0,1] \times [0,\epsilon]$ where the uppermost region is of possibly smaller height. Then for each $i \in \{1, \ldots, p\}$ let $T_i = \{R_j \in L | R_j \cap S_i \neq \emptyset\}$ and let $U_1, \ldots, U_{m+1}$ be the $m+1$ sets out of $T_1, \ldots, T_p$ which have the smallest profit. Removing these from the packing causes a loss of profit which is $2(m+1)/pP(L)$ at most; we remove $m+1$ sets since we might select the uppermost region; in this way we assert that we have at least m regions of height ϵ. Let L' be the set of remaining boxes; finally $2(m+1)/pP(L) \leq 2(m+1)(h/\epsilon)^{-1}P(L) = 2(m+1)\epsilon/hP(L)$ holds. □

Note that the construction above can be carried out in any direction.

Theorem 6. *Let $L = \{R_1, \ldots, R_n\}$ be a list of boxes with $z_i \leq \epsilon$ for each $R_i \in L$ and $V(L) \leq \alpha$ with $\alpha \in [1/4, 1]$ holds. Then it is possible to select $L'' \subseteq L$ such that $P(L'') \geq (1 - 12\epsilon)P(L)$ holds and L'' admits a feasible packing into at most $\lceil 4\alpha \rceil$ bins. The construction can be carried out in time polynomial in n.*

Proof. Use Algorithm A to pack L into a strip of height at most $h := 4\alpha$ and two additional layers L_1 and L_2 by Theorem 2. Let L' be the set of boxes in L arranged in the strip; the following construction is illustrated in Figure 3. Use Lemma 4 to generate at most 5 suitable gaps in the strip, resulting in a loss of profit of at most $12\epsilon/hP(L') \leq 12\epsilon/hP(L)$; since $\alpha \in [1/4, 1]$, this loss is bounded by $12\epsilon P(L)$. The remaining set of boxes in the strip and L_1 and L_2 is denoted as L''. Consider the 3 cutting unit squares $[0, 1] \times [0, 1] \times \{i\}$ for $i \in \{1, 2, 3\}$ and L_3, L_4 and L_5 be the sets of boxes in the strip that intersect with these unit squares, respectively. W.l.o.g. none of the sets $L_1, \ldots, L_5$ is empty; otherwise it is removed from consideration. Note that each of the sets $L_1, \ldots, L_5$ can be arranged on a layer of height at most ϵ, so we generate a feasible packing by arranging them into the 5 gaps. In the resulting packing, the 3 cutting unit squares $[0, 1] \times [0, 1] \times \{i\}$ for $i \in \{1, 2, 3\}$ do not intersect with any box. Furthermore, all layers $L_1, \ldots, L_5$ are merged in the strip; the packing can be rearranged into $\lceil 4\alpha \rceil$ bins. □

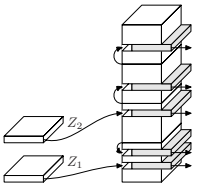


Fig. 3. The shifting technique described in Theorem 6

Similar as before for any $d \in \{x, y, z\}$ a box R_i is called *d-ϵ-big* $:\Leftrightarrow d_i \in (\epsilon, 1]$, and *$d$-$\epsilon$-small* $:\Leftrightarrow d_i \in (0, \epsilon]$. We explain the details in the proof only.

Algorithm E

1. Set $\delta := \epsilon/[35(7 + \epsilon)]$, let $L_1 := \{R_i | R_i$ is d-δ-big for each $d \in \{x, y, z\}\}$ and $L_2 := L \setminus L_1$.
2. For each $L_3 \subseteq L_1$ such that $|L_3| \leq \lfloor 1/\delta^3 \rfloor$ use an exact algorithm to verify whether L_3 is feasible. Store feasible L_3 of maximum total profit.
3. Use an FPTAS for classical KP from [12,14] to select $L_4 \subseteq L_2$ such that $V(L_4) \leq 1$ and $P(L_4) \geq (1 - \delta)\mathrm{OPT}$ holds.
4. Use the construction described below to select $L_5 \subseteq L_4$ which can be packed into at most 6 bins under a small loss of profit.
5. Out of the at most 7 sets generated in Step 2 and Step 4 return one with maximum profit.

Theorem 7. *Algorithm E is a $(7+\epsilon)$-approximation algorithm for OKP-3. Furthermore, this bound is asymptotically tight in the sense that it cannot be improved for ϵ arbitrary small.*

Proof. Note that $\lfloor 1/\delta^3 \rfloor$ is an upper bound for the number of boxes from L_1 in a feasible solution since δ^3 is a lower bound for the volume of each $R_i \in L_1$. Step 2 can be carried out in time polynomial in δ and thus polynomial in $1/\epsilon$ using an exact optimization algorithm as in [2]. We show that in Step 4 at most 6 sets are generated, resulting in at most 7 bins in total. Partition L_2 into 3 subsets X', Y' and Z' such that in each of these all boxes R_i are d-ϵ-small for the correspondig direction; note that $V(X') + V(Y') + V(Z') \leq 1$ holds. We apply the construction from Theorem 6 in each of the three directions. Study the following cases, where $V(X') \geq V(Y') \geq V(Z')$ holds w.l.o.g. *Case 1:* $V(X') \in (3/4, 1]$. The boxes in X' can be packed into at most 4 bins. We have $V(Y') + V(Z') \leq 1/4$. This means $V(Y') \leq 1/4$ and $V(Z') \leq 1/4$ holds. Consequently Y' and Z' can be packed into one bin each, resulting in at most 7 bins in total. *Case 2:* $V(X') \in (1/2, 3/4]$. The boxes in X' can be packed into at most three bins. Furthermore $V(Y') + V(Z') < 1/2$, which means that $V(Y') < 1/2$ holds. Consequently the boxes in Y' can be packed into at most 2 bins. Furthermore $V(Z') < 1/4$ holds and finally the boxes in Z' can be packed into 1 bin; this generates at most 7 bins in total. *Case 3:* We have $V(X') \in [0, 1/2]$. The boxes in X' can be packed into at most two additional bins. Furthermore $V(Y') \leq 1/2$ and $V(Z') \leq 1/2$ holds. This means that the boxes in Y' and Z' can be packed into at most two bins each. In total at most 7 bins are generated. In each of these cases at most 7 bins are generated; now we prove the ratio. Fix an optimal solution S and let P_1^* be the profit of boxes in $S \cap L_1$ and let P_2^* be the profit of boxes in $S \cap L_2$. Consequently $P_1^* + P_2^* \geq \mathrm{OPT}$ holds. Let P_1 be the profit of the set that is stored in Step 2 and let P_2 be the profit of the set that is selected in Step 3. By construction we have $P_1 \geq P_1^*$ and $P_2 \geq (1-\delta)P_2^*$. Furthermore, by threefold application of the construction from Theorem 6 the loss of profit in P_2 is bounded by $36\delta P_2$. The profit of the set returned in Step 5 is at least

$$\begin{aligned}(P_1 + P_2)/7 &\geq (P_1^* + (1-\delta)(1-36\delta)P_2^*)/7 \\ &\geq (P_1^* + P_2^*)(1-\delta)(1-36\delta)/7 \\ &= \mathrm{OPT}(1-\delta)(1-36\delta)/7 \geq \mathrm{OPT}/(7+\epsilon)\end{aligned}$$

which proves the claimed approximation ratio. The instance for the tightness of the bound is omitted for space reasons. □

5 Conclusion

We contributed approximation algorithms for an NP-hard combinatorial optimization problem, where the runtimes of the simpler algorithms are practical. It is an open problem whether here an algorithm with a ratio less than $7+\epsilon$ exists. We are interested in a reduction of the running time, especially for Algorithm E. In [15] it was proved that it is NP-complete to decide whether a set of squares can be packed into the unit square. However, it is an open problem whether checking the feasibility of cubes into the unit cube is NP-complete. Lemma 4

reminds of the main result from [23], but is less flexible and structural. Further research is necessary to generalize the main result from [23] to the 3D case.

Acknowledgements. The authors thank the anonymous referees for valuable comments. Florian Diedrich thanks Denis Naddef for hospitality during the preparation of this article.

References

1. B. S. Baker, D. J. Brown, and H. P. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2(4):348–368, 1981.
2. N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31:31–49, 2006.
3. N. Bansal, X. Han, K. Iwama, M. Sviridenko, and G. Zhang. Harmonic algorithm for 3-dimensional strip packing problem. accepted at the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.
4. A. Caprara. Packing two-dimensional bins in harmony. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 490–499, 2005.
5. F. Diedrich. Approximative Algorithmen für Rucksackprobleme. Diploma thesis, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, 2004.
6. A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science (Special Issue on Dynamic and On-line Algorithms)*, 130(1):49–72, 1994.
7. W. Fernandez de la Vega and G. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
8. R. Harren. Approximating the orthogonal knapsack problem for hypercubes. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 238–249, 2006.
9. R. Harren. Approximation Mehrdimensionaler Packungsprobleme. Diploma thesis, Universität Dortmund, 2006.
10. K. Jansen and R. Solis-Oba. An asymptotic approximation algorithm for 3d-strip packing. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 143–152, 2006.
11. K. Jansen and G. Zhang. Maximizing the total profit of rectangles packed into a rectangle. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 197–206, 2004.
12. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
13. C. Kenyon and E. Rémila. A near-optimal solution to a two dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.
14. E. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4:339–356, 1979.
15. J. Y.-T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10:271–275, 1990.
16. K. Li and K.-H. Cheng. On three-dimensional packing. *SIAM Journal of Computation*, 19(5):847–867, 1990.

17. K. Li and K.-H. Cheng. Heuristic algorithms for on-line packing in three dimensions. *Journal of Algorithms*, 13:589–605, 1992.
18. R. H. Li and M. Y. Yue. The proof of FFD(L) $\leq$ (11/9)OPT(L) + (7/9). *Chinese Science Bulletin*, 42:1262–1265, 1997.
19. S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.
20. F. K. Miyazawa and Y. Wakabayashi. An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica*, 18:122–144, 1997.
21. I. Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *Proceedings of the Second Annual European Symposium on Algorithms (ESA)*, pages 290–299, 1994.
22. D. D. K. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, 1980.
23. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal of Computation*, 26(2):401–409, 1997.
24. V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.
25. D. Ye and G. Zhang. Online scheduling of parallel jobs with dependencies on 2-dimensional meshes. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 329–338, 2003.

A Comparative Study of Efficient Algorithms for Partitioning a Sequence into Monotone Subsequences

Bing Yang[1], Jing Chen[2], Enyue Lu[3], and S.Q. Zheng[2,4]

[1] Cisco Systems, 2200 East President George Bush Highway, Richardson, TX 75082
[2] Telecom. Engineering Program, University of Texas at Dallas, Richardson, TX 75083
[3] Math. and Computer Science Dept., Salisbury University, Salisbury, MD 21801
[4] Dept. of Computer Science, University of Texas at Dallas, Richardson, TX 75083

Abstract. Tradeoffs between time complexities and solution optimalities are important when selecting algorithms for an NP-hard problem in different applications. Also, the distinction between theoretical upper bound and actual solution optimality for realistic instances of an NP-hard problem is a factor in selecting algorithms in practice. We consider the problem of partitioning a sequence of n distinct numbers into minimum number of monotone (increasing or decreasing) subsequences. This problem is NP-hard and the number of monotone subsequences can reach $\left\lfloor \sqrt{2n+\frac{1}{4}} - \frac{1}{2} \right\rfloor$ in the worst case. We introduce a new algorithm, the modified version of the Yehuda-Fogel algorithm, that computes a solution of no more than $\left\lfloor \sqrt{2n+\frac{1}{4}} - \frac{1}{2} \right\rfloor$ monotone subsequences in $O(n^{1.5})$ time. Then we perform a comparative experimental study on three algorithms, a known approximation algorithm of approximation ratio 1.71 and time complexity $O(n^3)$, a known greedy algorithm of time complexity $O(n^{1.5}\log n)$, and our new modified Yehuda-Fogel algorithm. Our results show that the solutions computed by the greedy algorithm and the modified Yehuda-Fogel algorithm are close to that computed by the approximation algorithm even though the theoretical worst-case error bounds of these two algorithms are not proved to be within a constant times of the optimal solution. Our study indicates that for practical use the greedy algorithm and the modified Yehuda-Fogel algorithm can be good choices if the running time is a major concern.

Keywords: monotone, subsequence, permutation, algorithm, NP-complete, approximation, complexity.

1 Introduction

A subsequence of a sequence of distinct numbers is monotone if it is increasing or decreasing. Partitioning a sequence into monotone subsequences is a problem that has many applications. This problem has attracted attention for many years (e.g. [1]-[17]). As early as 1935, Erdös and Szekeres [1] proved that every sequence

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 46–57, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of n numbers has a monotone subsequence of size $\lceil \sqrt{n} \rceil$. In 1950, Dilworth proved his famous *Dilworth Theorem*, which says that for any sequence with elements in a partially ordered set, the size of a longest increasing (decreasing) subsequence equals to the minimum number of decreasing (increasing) subsequences that the sequence can be partitioned into [12]. There were several extensions of Dilworth's results on the partition problems on partially ordered sets, including [13]-[17].

Though the problem of partitioning a sequence into minimum number of increasing (decreasing) subsequences can be easily solved, the problem of partitioning a sequence into minimum number of monotone subsequences is difficult. In 1994, Wagner [5] proved that this problem is NP-hard. In 1998, Yehuda and Fogel [2] gave a $O(n^{1.5})$-time algorithm to partition a sequence into $2\lfloor\sqrt{n}\rfloor$ monotone subsequences. In 2002, Fomin, Kratsch and Novelli [3] gave an $O(n^3)$-time approximation algorithm (which is be named *the Fomin-Kratsch-Novelli algorithm* in this paper) of approximation ratio 1.71 for this problem. In [3], an $O(N^{1.5} \log n)$-time greedy algorithm is also presented. It is shown that the solutions computed by this algorithm have solution values no more than $\ln n$ times the optimal solution values.

A related topic is finding a tight upper bound for the minimum monotone subsequences that a sequence can be divided into. The results of Erdös *et al.* [1] and Dilworth's [12] led to a bound of $2\lfloor\sqrt{n}\rfloor$. In 1986, Brandstädt and Kratsch [4] gave a smaller bound of $\left\lfloor\sqrt{2n+\frac{1}{4}}-\frac{1}{2}\right\rfloor$ and proved it is existentially tight. A generalized result was stated by Erdös *et al.* [6] in 1991.

In this paper, we first present a new $O(n^{1.5})$-time algorithm based on the speedup techniques of [2]. This algorithm, which is named the *the modified Yehuda-Fogel algorithm*, uses the upper bound $\left\lfloor\sqrt{2n+\frac{1}{4}}-\frac{1}{2}\right\rfloor$ as a heuristic and guarantees a solution of no more than $\left\lfloor\sqrt{2n+\frac{1}{4}}-\frac{1}{2}\right\rfloor$ monotone subsequences. Noticing that the Fomin-Kratsch-Novelli algorithm, the greedy algorithm and the modified Yehuda-Fogel algorithm have different time complexities, we compare their performances in terms of solution optimality by running them on randomly generated data sets. Our experiments show that their actual performances are quite close. Since tradeoffs between time complexities and solution optimalities and the distinction between theoretical upper bound and actual solution optimality for realistic instances of an NP-hard problem are important when selecting algorithms for an NP-hard problem in different applications, our study provides evidence that for random inputs the partition algorithms of lower time complexities are desirable for time-critical applications.

2 Modified Yehuda-Fogel Algorithm

Denote the set of all permutations $\pi = (\pi(1), \pi(2), \cdots, \pi(n))$ of integers $\{1, 2, \cdots, n\}$ by $\Pi(n)$. A permutation in $\Pi(n)$ is called (x, y)-*partitionable* if it can be partitioned into x increasing and y decreasing subsequences, and such a partition is called a (x, y)-*partition*. Let $p(x, y)$ denote the subset of

permutations in $\Pi(n)$ that are (x, y)-partitionable. For a permutation π, we define $P(\pi) = min\{x + y : \pi \in p(x, y)\}$ and $P_n = \max_{\pi \in \Pi(n)} P(\pi)$.

Theorem 1 ([4]). $P_n \leq \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$.

In the greedy algorithm of [3], the most time-consuming portion is iteratively finding the longest increasing subsequences. In each iteration, the search for a longest increasing sequence starts freshly without considering the remaining data structure at the end of the previous iteration. Yehuda and Fogel [2] considered using a layered data structure to reduce running time. Their algorithm guarantees at most $2\lfloor\sqrt{n}\rfloor$ monotone subsequences, and runs in $O(n^{1.5})$ time, which is faster than the greedy algorithm. We modify the original Yehuda-Fogel algorithm to produce fewer monotone subsequences while keeping the same time complexity.

Let $\pi = (\pi(1), \pi(2), \cdots, \pi(n)) \in \Pi(n)$ be any permutation, and P be a subsequence of π. For an element $p = \pi(i)$ in P, define $p.x$ and $p.y$ as i and $\pi(i)$, respectively. Element q dominates element p if $q.x > p.x$ and $q.y > p.y$. An element p in P is called a *maximal element* in P if there is no element in P dominating p. The set of all maximal elements in P is denoted as $M(P)$. Yehuda and Fogel defined a layer structure on P as

$$\mathscr{L}(P) = \Big(L_1(P), L_2(P), \cdots, L_l(P)\Big),$$

where $L_i(P)$, called the ith-layer of P, is defined recursively as follows:

$$\begin{aligned} L_1(P) &= M(P) \\ L_i(P) &= M\Big(P - \bigcup_{1 \leq j < i} L_j(P)\Big) \end{aligned}$$

For example, the layer structure on a permutation π = (9, 5, 11, 4, 1, 3, 2, 10, 7, 6, 8) is: $\mathscr{L}(\pi) = (L_1, L_2, L_3, L_4)$, where $L_1 = (11, 10, 8)$, $L_2 = (9, 7, 6)$, $L_3 = (5, 4, 3, 2)$ and $L_4 = (1)$, as shown in Figure 1.

Clearly, each layer $L_i(P)$ is a decreasing subsequence. Yehuda and Fogel showed that $|\mathscr{L}(P)|$, the number of layers, is the size of a longest increasing subsequence in P, and the layer structure of P can be computed in $O(n \log n)$ time[2]. We name such an algorithm the *Layer-Construction* algorithm.

It is shown in [2] that, given $\mathscr{L}(P)$ and an integer $k \leq |\mathscr{L}(P)|$, an increasing subsequence of k elements from k consecutive layers of $\mathscr{L}(P)$ can be found in $O(|P|)$ time. We name such an algorithm the *Increasing Subsequence* algorithm.

Let S be an increasing subsequence of P computed by the Increasing Subsequence algorithm. Yehuda and Fogel gave an algorithm that computes the layer structure $\mathscr{L}(P - S)$ of $P - S$ from $\mathscr{L}(P)$ in $O(n + |S|^2)$ time, where $|P| \leq n$. We name this algorithm the *Layer Update* algorithm.

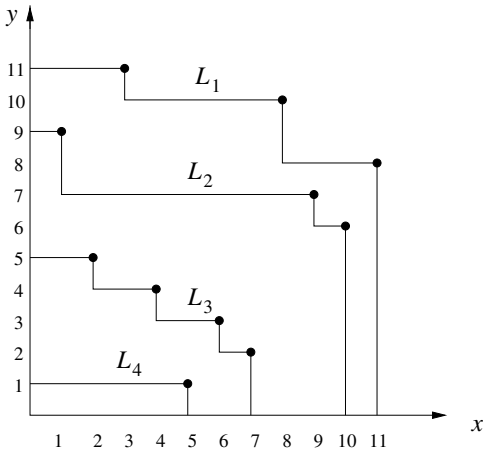


Fig. 1. Layer structure of $\pi = (9, 5, 11, 4, 1, 3, 2, 10, 7, 6, 8)$

Now, we present a monotone subsequence partition algorithm based on the Yehuda-Fogel algorithm[2] with improved solution optimality. We call our algorithm the *modified Yehuda-Fogel algorithm*.

Input: A permutation $\pi = (\pi(1), \pi(2), \cdots, \pi(n))$.
Output: A list of monotone subsequences $S = (S_1, S_2, \cdots, S_k)$ with $k \leq \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$.

1. Let S be an empty set for monotone subsequences.
2. Let $K := \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$, $P := \pi$ and $S := \emptyset$.
3. Use the Layer Construction algorithm to compute $\mathscr{L}(P) = (L_1, L_2, \cdots, L_l)$, the layer structure of P.
4. For $k = K$ down to 1 by -1 do
   (a) Let $l := |\mathscr{L}(P)|$.
   (b) If $l \leq k$ then include L_j, $1 \leq j \leq l$, as S_{K-k+j}, and go to Step 5.
   (c) If $l > k$ then construct $\mathscr{L}(P') = (L_{l-k+1}, L_2, \cdots, L_l)$ from $\mathscr{L}(P) = (L_1, L_2, \cdots, L_l)$, with P' being the set of elements in $(L_{l-k+1}, L_2, \cdots, L_l)$.
   (d) Use the Increasing Subsequence to find a increasing subsequence D of P' and include D into S as S_{K-k+1}.
   (e) Use the Layer Update algorithm to get $\mathscr{L}(P' - D) = (L'_1, \cdots, L'_{k'})$.
   (f) Let $P := P - D$. If $P = \emptyset$ then go to Step 5.
   (g) For $j = 1$ to k' do: rename L'_j as L_{l-k+j}.
   (h) Let $\mathscr{L}(P) := (L_1, L_2, \cdots, L_{l-k+k'})$.
5. Return S.

The difference between the original Yehuda-Fogel algorithm and the modified Yehuda-Fogel algorithm lies in the selection of k in Step 4. In the original

Yehuda-Fogel algorithm, k is the fixed value of $\lceil \sqrt{n} \rceil$. Instead of using a fixed number in Step 4, our modified algorithm uses a flexible control mechanism to achieve a smaller number of resulting monotone subsequences, which is at least about $\sqrt{2}$ times smaller than the solution computed by the original Yehuda-Fogel algorithm.

The correctness proof of the Modified Yehuda-Fogel algorithm is similar to the correctness proof of the Yehuda-Fogel algorithm[2]. The time complexity of the modified Yehuda-Fogel algorithm is determined by the performance of Step 4. Each substep takes $O(n)$ time, with only Step 4.e requiring explanation. It takes $O(n + k^2) = O(n)$ time, since $k \leq \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$. The for-loop of Step 4 runs for at most $\left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor = O(n^{0.5})$ iterations, resulting total $O(n^{1.5})$ time complexity of Step 4. In summary, we have the following claim.

Theorem 2. *Any permutation π in $\Pi(n)$ can be partitioned into at most $\sqrt{2n + \frac{1}{4}} - \frac{1}{2}$ monotone subsequences by the modified Yehuda-Fogel algorithm in $O(n^{1.5})$ time.*

3 Average Performance

The problem of partitioning a sequence into minimum number of monotone subsequences was shown NP-hard. The Fomin-Kratsch-Novelli algorithm (F-K-N for short), the greedy algorithm (Greedy for short), the Yehuda-Fogel algorithm, (Y-F for short) and the modified Yehuda-Fogel algorithm (M.Y-F for short) are approximation/heuristic algorithms for this problem. Their time complexities and solution optimalities (in terms of the worst-case theoretical ratios of the obtained solution values and the optimal solution values), are summerized in the following table.

	Time	Provable worst ratio to $P(\pi)$
F-K-N	$O(n^3)$	$1 + \frac{1}{\sqrt{2}} \approx 1.71$
Greedy	$O(n^{1.5} \log n)$	$\ln(n)$
Y-F	$O(n^{1.5})$	$2\lfloor \sqrt{n} \rfloor / P(\pi)$
M.Y-F	$O(n^{1.5})$	$\left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor / P(\pi)$

There is an issue of tradeoffs between algorithm time complexities and the solution optimalities. For example, a natural question arises: what are the average performances of these algorithms? The answer to this question is important in determining time/optimality tradeoffs for practical uses. One way of answering this question is by theoretical analysis, which can be very difficult. Another way is by experiments, which is much easier and more meaningful for practical purposes. Evaluating the performances of approximation/heuristic algorithms by experiments can provide strong evidence on the effectiveness of these algorithms,

and this approach has been widely used in practice. Since the modified Yehuda-Fogel algorithm always outperforms the Yehuda-Fogel algorithm, we conduced extensive experiments only on the Fomin-Kratsch-Novelli algorithm, the greedy algorithm and the modified Yehuda-Fogel algorithm.

We implemented these three algorithms and applied them to randomly generated permutations (i.e. sequences) with size n equals 50, 100, 500, 1000, 2000, 3000, and 4000, respectively. The numbers m of random permutations used are summerized in the following table. Since the algorithms take much longer time to execute when the permutation size increases, a smaller m is chosen for a larger n.

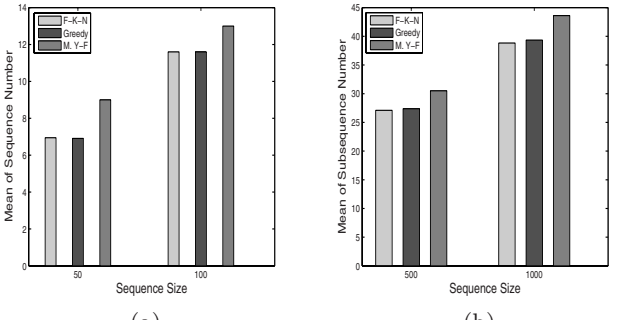
n	50	100	500	1000	2000	3000	4000
m	10000	10000	2000	500	200	100	50

The effectiveness of our new modified Yehuda-Fogel algorithm is derived from comparing the average numbers of monotone subsequences computed by the three algorithms.

Let $P_m(n) = \{\Pi_1(n), \Pi_1(n), \cdots, \Pi_m(n)\}$ be a set of m random permutations of size n, and $N_A(\Pi_i(n))$ the number of monotone subsequences produced by applying algorithm A to permutation $\Pi_i(n)$. Define

$$N_{A,P_m(n)} = \frac{\sum_{\Pi_i(n) \in P_m(n)} N_A(\Pi_i(n))}{m}.$$

That is, $N_{A,P_m(n)}$ is the average number of monotone subsequences obtained by applying algorithm A to the random permutations of $P_m(n)$. Figures 2 and Figure 3 show the average numbers of monotone subsequences (i.e. $N_{F-K-N,P_m(n)}$, $N_{Greedy,P_m(n)}$, and $N_{M.Y-F,P_m(n)}$) computed by the three algorithms with random permutations (sequences) of various sizes. Figure 4 shows the performance differences $\frac{N_{A,P_m(n)}}{N_{F-K-N,P_m(n)}}$ of the three algorithms, using the average subsequences numbers computed by the Fomin-Kratsch-Novelli algorithm as basis (i.e. 100%).



(a) (b)

Fig. 2. $N_{A,P_m(n)}$, mean number of monotone subsequences, with n = 50, 100, 500, and 1000

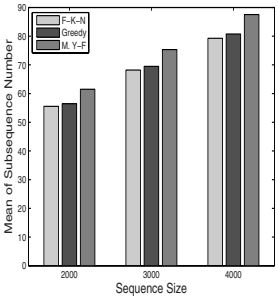


Fig. 3. $N_{A,P_m(n)}$, mean number of monotone subsequences, with $n = 2000$, 3000, and 4000

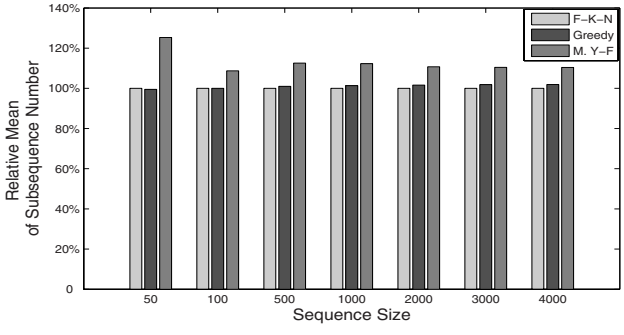


Fig. 4. Mean number of monotone subsequences compared with the mean number of monotone subsequences computed by the Fomin-Kratsch-Novelli algorithm

These figures show that the average solution values computed by the modified Yehuda-Fogel algorithm are only about 10% larger than the average solution values computed by the other two algorithms (except $n = 50$). This indicates that, in average, the solutions computed by the modified Yehuda-Fogel algorithm within about $1.71 \times 1.10 = 1.88$ times the optimal solutions. As expected the modified Yehuda-Fogel algorithm is the fastest among the three, but it produces less accurate results. If the running time is the major concern in some applications, the modified Yehuda-Fogel algorithm can be used.

Somewhat surprisingly, the average solution values computed by the greedy algorithm are about the same as (and for $n = 50$, even better than) the average solution values computed by the Fomin-Kratsch-Novelli algorithm, even though its $O(n^{1.5} \log n)$ time complexity is significantly lower than the $O(n^3)$ complexity of the Fomin-Kratsch-Novelli algorithm.

4 A Closer Look at Greedy Algorithm

Our experiment show that, not only the average performance, the monotone subsequence numbers on individual permutations computed by the greedy algorithm are also very close to those computed by the Fomin-Kratsch-Novelli algorithm (which is a constant ratio approximation algorithm).

Let

$$O_{A,k}(\Pi_i(n)) = \begin{cases} 1, \text{ if } N_A(\Pi_i(n)) = k; \\ 0, \text{ otherwise.} \end{cases}$$

Define

$$OR_{A,P_m(n)}(k) = \frac{\sum_{\Pi_i(n) \in P_m(n)} O_{A,k}(\Pi_i(n))}{m}.$$

$OR_{A,P_m(n)}(k)$ is the *occurrence ratio* of the exactly k monotone subsequences obtained by applying algorithm A to the set $P_m(n)$ of m random permutations of size n. Subfigure (a) of each of Figures 5 to 11 compares the $OR_{F-K-N,P_m(n)}(k)$ and $OR_{Greedy,P_m(n)}(k)$ for the same $P_m(n)$s.

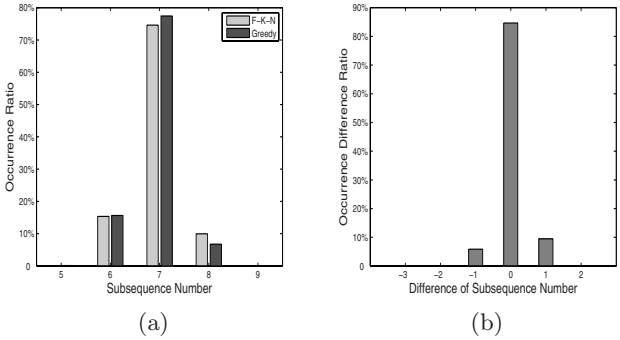
Let

$$OD_k(\Pi_i(n)) = \begin{cases} 1, \text{ if } N_{F-K-N}(\Pi_i(n)) \\ \quad -N_{Greedy}(\Pi_i(n)) = k; \\ 0, \text{ otherwise.} \end{cases}$$

Define

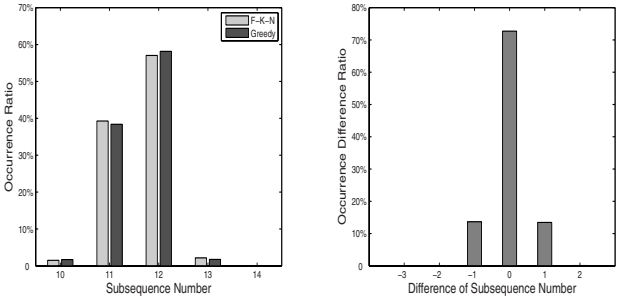
$$ODR_{P_m(n)}(k) = \frac{\sum_{\Pi_i(n) \in P_m(n)} OD_k(\Pi_i(n))}{m}.$$

$ODR_{P_m(n)}(k)$ is the *occurrence difference ratio* of the difference of exactly k between the monotone subsequences obtained by applying the Fomin-Kratsch-Novelli algorithm and the greedy algorithm to the set $P_m(n)$ of m random permutations of size n. Subfigure (b) of each of Figures 5 to 11 displays the distribution of $ODR_{P_m(n)}(k)$. For example, Figure 5(b) shows that the number of subsequences computed by the greedy algorithm is at most 1 more than the



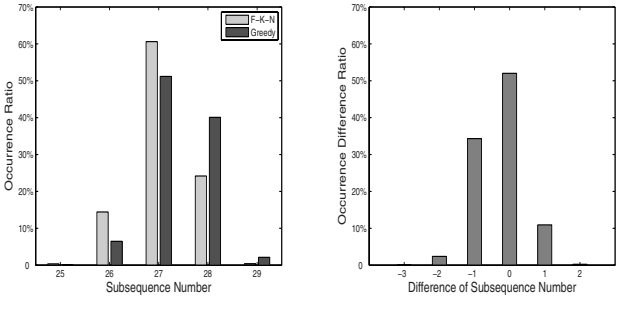
(a) (b)

Fig. 5. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 50$



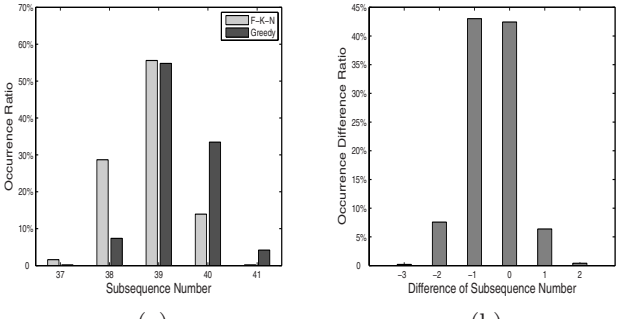
(a) (b)

Fig. 6. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 100$



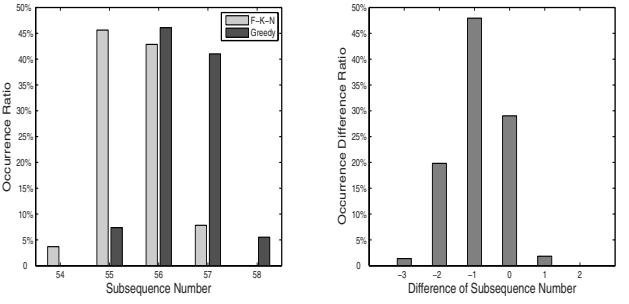
(a) (b)

Fig. 7. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 500$



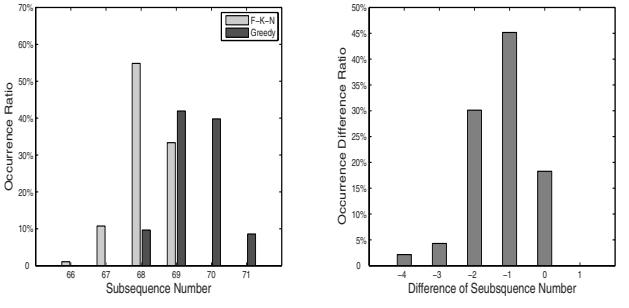
(a) (b)

Fig. 8. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 1000$



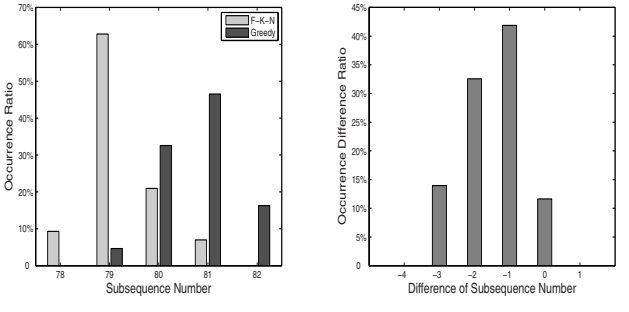
(a) (b)

Fig. 9. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 2000$



(a) (b)

Fig. 10. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 3000$



(a) (b)

Fig. 11. (a) $OR_{A,P_m(n)}(k)$ and (b) $ODR_{P_m(n)}(k)$, $n = 4000$

number of subsequences computed by the Fomin-Kratsch-Novelli algorithm for $n = 50$ in our experiments; furthermore, the number of subsequences computed by the greedy algorithm is 1 less than the number of subsequences computed by the Fomin-Kratsch-Novelli algorithm in about 10% of tested instances.

By comparing the subsequence numbers and subsequence numbers differences, we can easily see that the Fomin-Kratsch-Novelli algorithm and the greedy algorithm compute roughly identical number of subsequences. Since the Fomin-Kratsch-Novelli algorithm has a constant approximation ratio, our experiments show strong evidence that the greedy algorithm is likely to be an approximation algorithm with a constant approximation ratio. Since the greedy algorithm is much faster than the Fomin-Kratsch-Novelli algorithm, the greedy algorithm should be given preference in real-world usage.

5 Concluding Remarks

In this paper we proposed a modified Yehuda-Fogel algorithm for partitioning any sequence with size n into at most $\left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ monotone subsequences. This algorithm guarantees a solution better than the one computed by the original Yehuda-Fogel algorithm without taking more time. Then we compared this algorithm with other two known algorithms (the Fomin-Kratsch-Novelli algorithm and the greedy algorithm) of higher time complexities by experiments. Our experiments show that in average the modified Yehuda-Fogel algorithm computes comparable solutions. The input paterns, if any, for which the greedy algorithm and the modified Yehuda-Fogel algorithm compute results that are significantly worse than the Fomin-Kratsch-Novelli algorithm may be very rare.

By experiments, we also compared the performances of the Fomin-Kratsch-Novelli algorithm and the greedy algorithm. Our experiments show that the solutions computed by these two algorithms are almost identical. Our study shows that for practical use both of the modified Yehuda-Fogel algorithm and the greedy algorithm are good choices. For time-critical applications that tolerate certain degree of inaccuracy, the modified Yehuda-Fogel algorithm because of its lower time complexity.

In [3], Fomin *et al.* posed a question of whether or not the greedy algorithm is an approximation algorithm with a constant approximation ratio. Our experiments show that it is very likely that the answer to this question is affirmative. However, proving or disproving this conjecture remains to be an outstanding open problem.

References

1. P. Erdös and G. Szekeres, "A Combinatorial Problem in Geometry", *Compositio Mathematica* 2 (1935) 463-470.
2. R. B. Yehuda and S. Fogel, "Partitioning a Sequence into Few Monotone Subsequences", *Acta Informatica* 35 (1998) 421-440.

3. F. V. Fomin, D. Kratsch and J. Novelli, "Approximating Minimum Cocolorings", *Information Processing Letters* 84 (2002) 285-290.
4. A. Brandstädt and D. Kratsch, "On Partitions of Permutations into Increasing and Decreasing Subsequences", *Elektron. Inf. Verarb. Kybern.* 22 (1986) 263-273.
5. K. Wagner, "Monotonic Coverings of Finite Sets", *Elektron. Inf. Verarb. Kybern.* 20 (1984) 633-639.
6. P. Erdös, J. Gimbel and D. Kratsch, "Some Extremal Results in Cochromatic and Dichromatic Theory", *Journal of Graph Theory* 15 (1991) 579-585.
7. J. S. Myers, "The Minimum Number of Monotone Subsequences", *Electronic Journal of Combinatorics* 9(2) (2002) R4.
8. C. A. Tracy and H. Widom, "On the Distributions of the Lengths of the Longest Monotone Subsequences in Random Words", *Probab. Theory Relat. Fields* 119 (2001) 350-380.
9. R. Siders, "Monotone Subsequences in Any Dimension", *Journal of Combinatorial Theory, Series A* 85 (1999) 243-253.
10. J. Matoušek and E. Welzl, "Good splitters for counting points in triangles", *The 5th Ann. ACM Conf. On Computational Geometry* (1989) 124-130.
11. M. L. Fredman, "On Computing the Length of Longest Increasing Subsequences", *Discrete Mathematics* 11 (1975) 29-35.
12. R.P. Dilworth, "A Decomposition Theorem for Partially Ordered Sets", *Annals of Mathematics* 51(1) (1950) 161-166.
13. A. Frank, "On Chain and Antichain Families of a Partially Ordered Set", *Journal of Combinatorial Theory, Series B* 29 (1980) 176-184.
14. A. J. Hoffman and D. E. Schwartz, "On Partitions of a Partially Ordered Set", *Journal of Combinatorial Theory, Series B* 23 (1977) 3-13.
15. C. Greene and D. J. Kleitman, "The Structure of Sperner k-Families", *Journal of Combinatorial Theory, Series A* 20 (1976) 41-68.
16. C. Greene, "Some Partitions Associated with a Partially Ordered Set", *Journal of Combinatorial Theory, Series A* 20 (1976) 69-79.
17. C. Greene and D. J. Kleitman, "Strong Versions of Sperner's Theorem", *Journal of Combinatorial Theory, Series A* 20 (1976) 80-88.

The Hardness of Selective Network Design for Bottleneck Routing Games*

Haiyang Hou and Guochuan Zhang

Department of Mathematics, Zhejiang University, Hangzhou 310027, China
{yang629,zgc}@zju.edu.cn

Abstract. In this paper, motivated by the work of Azar et al. [3] we consider selective network design on bottleneck routing games. Assuming $P \neq NP$ we achieve the following results. For the *unsplittable bottleneck games* the trivial algorithm is a best possible approximation algorithm. For the *k-splittable unweighted bottleneck games* it is *NP*-hard to compute a best *pure-strategy Nash equilibrium*. Moreover no polynomial time algorithms can have a constant approximation ratio if the edge latency functions are continuous and non-decreasing.

1 Introduction

A lot of research has recently gone into the study of communication networks within the framework of game theoretic analysis. Because of the lack of a central authority that manages traffic, network users are free to act according to their own performance without regard to the social optimum. It is a well known fact that if each user chooses an action to optimize his own utility, then the selfish behavior of users would lead to a globally inefficient solution.

Koutsopias and Papadimitriou [10] and Papadimitriou [13] proposed to study of the *price of anarchy* (also referred to as the *coordination ratio*): How much worse is the cost of selfish behavior compared to a hypothetical centralized solution? This question has been studied in various different models (e.g. [2], [7], [16]). Another prominent measure is the *price of stability* [1], which is the best-case cost of selfish behavior compared to a hypothetical centralized solution. It captures how efficient stability can get.

The concept of *k-splittable* flows was first studied in Baier et al. [4], as an intermediate problem between splittable and unsplittable flows. In such a situation, each commodity may split its flow along a finite number k of different paths. Meyers [11] applied the idea of *k-splittable* flows to network congestion games.

Banner et al. [5] considered bottleneck routing games, in which we are given a network, finitely many (selfish) users, each associated with a positive flow demand and a load-dependent latency function for each edge; the cost of a user

* Research supported in part by NSFC (60573020) and NSF of Zhejiang Province (Y605353).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 58–66, 2007.
© Springer-Verlag Berlin Heidelberg 2007

is the performance of the worst edge that he employs; the social (i.e. system) objective is to optimize the performance of the worst element in the network (i.e. the network bottleneck).

Braess's Paradox is the counterintuitive phenomenon that adding additional edges to a network with selfish routing can make all of the traffic worse off. This paradox was first discovered by Braess [6] and later reported by Murchland [12]. Braess's paradox motivates the following *network design problem* for improving the performance of a network with selfish users: given a network, which edges should be removed to obtain the best social cost in a *pure-strategy Nash equilibrium*? A natural variant of the problem is the *selective network design problem*: given a network $G = (V, E)$ and $E_1 \subseteq E$, how do we find a subgraph H of G containing the edges of E_1 that obtain the best social cost in *pure-strategy Nash equilibrium*?

Previous Work. Roughgarden [14] studied the problem of *network design problem* for *splittable* routing games by presenting a detailed analysis. It was proved that the best possible inapproximability results are $4/3$ for linear edge latency functions, $\Theta(d/\ln d)$ for polynomials of degree d edge latency functions, and $n/2$ for general continuous non-decreasing edge latency functions, respectively. Moreover the trivial algorithm is best possible in terms of approximation.

Azar et al. [3] investigated the *network design problem* and the *selective network design problem* for *unsplittable* routing games in which the users can have general demands and each user must choose a single path between his source and his destination. For linear edge latency functions they proved that $(3+\sqrt{5})/2$ is the best possible bound for the *network design problem* and the *selective network design problem*. For polynomials of degree d edge latency functions and weighted demands they showed a lower bound of $\Omega(d^{d/4})$ for any approximation algorithm for the *network design problem* and the *selective network design problem*. For general continuous non-decreasing edge latency functions no approximation algorithm for *network design problem* has a constant approximation ratio.

Banner et al. [5] dealt with bottleneck routing games for two routing scenarios, one of which is the *unsplittable bottleneck game* (a user only traffic over one path) and another of which is the *splittable bottleneck game*. They proved *the existence, convergence, price of anarchy and price of stability of pure Nash equilibria* in such games.

Our Results. In this paper, we focus the *selective network design problem* on the *bottleneck routing games*. We consider two routing scenarios, namely the *unsplittable bottleneck game* and the *k-splittable unweighted bottleneck game* (each player has identical weight, and split its traffic equally into k parts. Each part must choose a path, and thus each user uses at most k paths).

– If the edge latency functions are polynomial with a degree p, there is a lower bound of $\Omega(|E|^p)$ on the approximation ratio of any polynomial time algorithm for *unsplittable bottleneck Selective Network Design Problem*, unless $P = NP$. Meanwhile we show that the trivial algorithm is best possible in this sense.

- We show that a *k-splittable unweighted bottleneck game* admits a *pure-strategy* Nash equilibrium, but computing the best *pure-strategy Nash equilibrium* is *NP-hard*.
- For general continuous, non-decreasing edge latency functions, assuming $P \neq NP$, the approximation ratio of any polynomial time approximation algorithm for *k-splittable unweighted bottleneck Selective Network Design problem* is unbounded.

2 Preliminaries

Bottleneck Routing Games. There is a directed graph $G = (V, E)$. Each edge $e \in E$ is given a load-dependent performance function $f_e : R^+ \longrightarrow R^+$. We assume that, for all $e \in E$, f_e is continuous and non-decreasing. Given n users, each user j has a positive bandwidth request defined by a triple (s_j, t_j, ω_j), where $s_j, t_j \in V$ are the source/destination pair, and $\omega_j \in R^+$ corresponds to the required bandwidth. Denote by Λ_j the set of all paths from the source s_j to the destination t_j and by Λ the set of all paths in the network. We further denote by l_P^j the flow of user j on a path $P \in \Lambda_j$. User j can assign any value to l_P^j, as long as $l_P^j \geq 0$ (nonnegative constraint) and $\sum_{P \in \Lambda_j} l_P^j = \omega_j$ (demand constraint). This assignment of traffic to paths shall also be referred to as the user strategy. Each user is aware of the strategies made by all other users when making his decision. The set of all possible strategies of a user is referred to as the user strategy space. The product of all user strategy spaces is termed the joint strategy space; each element in the joint strategy space is termed a (flow vector) profile; effectively it is a global assignment of traffic to paths that satisfies the demands of all users.

Given a profile $l = \{l_P^j\}$ and a path $P \in \Lambda$, denote by l_P the total flow that is carried over P, i.e. $l_P = \sum_{j=1}^{n} l_P^j$. Let l_e^j be the total flow that user j transfers though e, i.e. $l_e^j = \sum_{P|e \in P} l_P^j$. Finally, for a profile l and an edge $e \in E$, denote the total flow carried by e as l_e. We define the *network bottleneck* $B(l)$ of a flow l as the performance of the worst edge in the network, i.e. $B(l) \triangleq \max_{e \in E}\{f_e(l_e)\}$. Analogously, we define the *bottleneck of user* j as the performance of the worst edge that j employs, i.e. $b_j(l) \triangleq \max_{e \in E | l_e^j > 0}\{f_e(l_e)\}$.

We assume that the users are non-cooperative, who wish to minimize their own bottlenecks giving no regard to the global optimum. In this paper, we consider two games. The triple $\langle G, N, \{f_e\}\rangle$ is termed an *unsplittable bottleneck game* if each user can employ only a single path, while it is termed *k-splittable unweighted bottleneck game* if each player has identical weight and can split its traffic equally into k parts, each of which can choose a path.

A profile is said to be at *pure-strategy Nash equilibrium* (PNE) if there is no incentive for any user to change its strategy. More formally, considering an

unsplittable (k-splittable unweighted) bottleneck game, $l = \{l_P^j\}$ is an unsplittable (k-splittable unweighted) *Nash flow* if, for each user j and every flow vector $\tilde{l} = \{\tilde{l}_P^j\}$ that satisfies $l_P^i = \tilde{l}_P^i$ for each $i \in N \setminus \{j\}$, we get $b_j(l) \leq b_j(\tilde{l})$.

The *price of anarchy* (PoA) of a bottleneck routing game is defined as the largest ratio between the cost of a PNE of the game and the cost of the optimal (minimum network bottleneck $B(l)$) outcome. The PoA of a game can be viewed as the worst-case loss in efficiency due to uncoordinated behavior by selfish users.

Formalizing the Network Design Problem. Let $B(H, l)$ be the network bottleneck incurred by a given profile l in a PNE for subgraph H of G. If there is a user j such that $\Lambda_j = \emptyset$ in H, then $B(H, l) = \infty$. We denote by $B(H)$ the maximum network bottleneck obtained for the graph H, where the maximum is taken over all profiles l in the PNE for the graph H. *The Selective Network Design Problem* is stated as follows. Given a bottleneck routing game with directed graph $G = (V, E)$ and $E_1 \subseteq E$, find a subgraph H of G containing the edges of E_1 that minimizes $B(H)$. Let H^* be the subgraph containing E_1 produced by an optimal algorithm and H be the subgraph containing E_1 obtained by an approximation algorithm A. The approximation ratio of algorithm A is thus defined to be the worst case ratio between $B(H)$ and $B(H^*)$ over all instances.

3 Main Results

We first consider unsplittable bottleneck games. Banner et al. [5] proved that a PNE always exists and the PoA is unbounded. More specifically, they showed

Proposition 1. *Given an* unsplittable bottleneck game $\langle G, N, \{f_e\}\rangle$, *where* $f_e(l_e) = a \cdot (l_e)^p$ *for each* $e \in E$, *the PoA is* $\Theta(|E|^p)$.

We aim at finding a subgraph H of G such that $B(H)$ is minimum. A trivial algorithm for the problem simply outputs the entire network G without checking with any proper subgraph of G. The following corollary can be easily observed with the similar analysis in [3].

Corollary 1. *Given an* unsplittable bottleneck game $\langle G, N, \{f_e\}\rangle$, *where* $f_e(l_e) = a \cdot (l_e)^p$ *for each* $e \in E$, *the trivial algorithm for The Selective Network Design Problem is an* $O(|E|^p)$*-approximation algorithm.*

Naturally the trivial algorithm may not produce a good solution. But, "unfortunately", it can be shown that the trivial algorithm is already best possible in terms of approximation for *The Selective Network Design Problem*.

Theorem 1. *For* unsplittable bottleneck game $\langle G, N, \{f_e\}\rangle$, *where* $f_e(l_e) = a \cdot (l_e)^p$ *for each* $e \in E$, *assuming* P $\neq$ NP, *the trivial algorithm is the best approximation algorithm for* The Selective Network Design Problem.

Proof. To prove this theorem we will use a reduction from the 2 Directed Disjoint Paths problem (2DDP), that was shown to be *NP*-complete in [8]. Given a

directed graph $G = (V, E)$ and distinct vertices s_1, s_2, t_1, $t_2 \in V$, the 2DDP asks if there exists an s_1-t_1 path P_1 and an s_2-t_2 path P_2 such that P_1 and P_2 are vertex disjoint. We are going to show that for an unsplittable bottleneck game $\langle G, N, \{f_e\}\rangle$, where $f_e(l_e) = a \cdot (l_e)^p$ for each $e \in E$, any algorithm with approximation ratio better than $\Omega(|E|^p)$ for *The Selective Network Design Problem* can be used to distinguish "yes" and "no" instances of 2DDP in polynomial time.

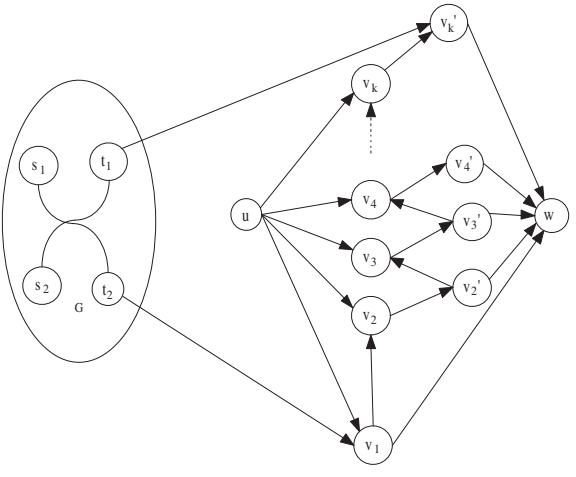


Fig. 1. Proof of Theorem 1

Let I be an instance of 2DDP, where the directed graph is $G = (V, E)$. Assume $|E| = k$. We expand graph G by adding vertices and edges as shown in Figure 1. The new network is denoted by $G' = (V', E')$. Let $E_1 = E' - E$ be the set of edges which the subgraph H of G' must contain. All the edges in G' have performance function $f_e(l_e) = a \cdot (l_e)^p$. We consider an unsplittable bottleneck game with $k + 4$ players who use the network G'. Let $\gamma > \varepsilon > 0$. Player 1 has a bandwidth request (s_1, t_1, ε) (player 1 has to move ϵ units of bandwidth from s_1 to t_1). Player 2 has a bandwidth request (s_2, t_2, ε). Player 3 has a bandwidth request (s_1, w, γ). Player 4 has a bandwidth request (s_2, w, γ). Player i $(4 < i \leq k + 4)$ has a bandwidth request (u, w, γ). It is easy to see that the new instance I' can be constructed from I in polynomial time. To complete the proof, it suffices to show the following two statements.

(1) If I is a "yes" instance of 2DDP, then G' contains a subgraph H of G' with $B(H) = a(2\gamma)^p$.
(2) If I is a "no" instance of 2DDP, then $B(H) \geq a(k\gamma)^p$ for all subgraphs H of G'.

Recall that the subgraph H of G' must contain all edges of E_1. To prove (1), let P_1 and P_2 be the vertex-disjoint paths in G, respectively. Construct H by

deleting all edges of G involved in neither P_1 nor P_2. Then, H is a subgraph of $G^{'}$ that contains the paths $s_1 \longrightarrow t_1$, $s_2 \longrightarrow t_2$, $s_1 \longrightarrow t_1 \longrightarrow v_k^{'} \longrightarrow w$, $s_2 \longrightarrow t_2 \longrightarrow v_1 \longrightarrow w$, $u \longrightarrow v_1 \longrightarrow w$, $u \longrightarrow v_2 \longrightarrow v_2^{'} \longrightarrow w$, $\cdots$, $u \longrightarrow v_k \longrightarrow v_k^{'} \longrightarrow w$. These paths are the direct paths of players $1, 2, \ldots, (k+4)$, respectively. The optimal solution is obtained when each player chooses his direct path and this solution is a PNE for $I^{'}$ in which the individual costs of the players are $a(\gamma+\epsilon)^p$, $a(\gamma+\epsilon)^p$, $a(2\gamma)^p$, $a(2\gamma)^p$, $a(2\gamma)^p$, $a(\gamma)^p$, $\cdots$, $a(\gamma)^p$, and $a(2\gamma)^p$ respectively. The network bottleneck $B(H) = a(2\gamma)^p$ and it is the unique value for any PNE.

For (2), we may assume that H contains $s_1 \longrightarrow t_1$, $s_2 \longrightarrow t_2$ paths. In this case the two paths are not disjoint and thus H must contain $s_1 \longrightarrow t_2$ path. Consider a profile as follows: Player 1 uses its direct path $s_1 \longrightarrow t_1$; player 2 uses its direct path $s_2 \longrightarrow t_2$; player 3 uses its indirect path $s_1 \longrightarrow t_2 \longrightarrow v_1 \longrightarrow w$; player 4 uses its direct path $s_2 \longrightarrow t_2 \longrightarrow v_1 \longrightarrow w$; all other players use the indirect path $u \longrightarrow v_1 \longrightarrow v_2 \longrightarrow v_2^{'} \longrightarrow v_3 \longrightarrow v_3^{'} \longrightarrow \cdots \longrightarrow v_k \longrightarrow v_k^{'} \longrightarrow w$. Then this is a PNE and the cost of players $3, 4, \ldots, (k+4)$ are $a(2\gamma)^p$, $a(k\gamma)^p$, $\cdots$, and $a(k\gamma)^p$, respectively. The network bottleneck $B(H) \geq a(k\gamma)^p$.

Note that $k = |E|$ and $k = \Omega(|E^{'}|)$. The proof is completed. □

In the following we turn to k-splittable unweighted bottleneck games with general continuous, non-decreasing edge performance function. It is observed that the proof of the existence of PNE for unsplittable bottleneck games in [5] can be applied to k-splittable unweighted bottleneck games. Thus the following theorem holds.

Theorem 2. *A* k-splittable unweighted bottleneck game *admits a* pure-strategy Nash equilibrium.

A k-splittable unweighted bottleneck game may have more than one PNE. A PNE is called the *best* PNE if amongst the set of PNE, it minimizes the maximum network bottleneck. We show that computing the *best* PNE is *NP*-hard.

Theorem 3. *Given a* k-splittable unweighted bottleneck game $\langle G, N, \{f_e\}\rangle$ *and a value* B*, it is* NP*-hard to determine if the game has a PNE with a* bottleneck *of at most* B.

Proof. The reduction is from the Disjoint Connecting Paths Problem [9]. In the problem, given a network $G = (V, E)$ and k distinct source-destination nodes $(s_1, t_1), (s_2, t_2), \cdots, (s_k, t_k)$, we want to determine if we can find k mutually link-disjoint paths connecting the k node pairs. For any instance I of the Disjoint Connecting Paths Problem, where the corresponding graph is G, we create a new network presented in Figure 2. Make k copies of the graph G. Add super sources $S_1, S_2, \cdots, S_k$ and connect them to all the k copies of s_i in the original graph. Similarly, add super sink $T_1, T_2, \cdots, T_k$, that connect to all copies of t_i node. Denote the new network by $G^{'} = (V^{'}, E^{'})$. We further define the following link performance functions: the edges in the copied graphs have the same performance function $f_e(l_e) = kBl_e$, all other new added edges with performance function

$l_e = 0$. We consider a k-splittable unweighted bottleneck game with k players who use the network $G^{'}$. Each player j needs to send a unit of flow from S_j to T_j. Every unit flow is split equally into k parts. Each part must choose a path, and thus each user can use st most k paths.

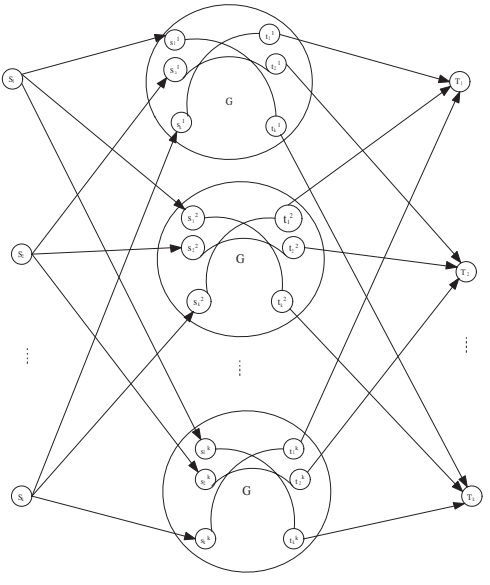


Fig. 2. Proof of Theorem 3

It is easy to show that there exist k mutually disjoint paths connecting the k node pairs if and only if there is a Nash flow with a network bottleneck of at most B. □

We show that the any approximation algorithm has an unbounded approximation ratio. The proof adopts the idea of [3].

Theorem 4. *For a general continuous, non-decreasing function, assuming* P $\neq$ NP*, the approximation ratio of any polynomial time algorithm for* k-splittable unweighted bottleneck Selective Network Design problem *is unbounded.*

Proof. We make a reduction from the problem 2DDP, similarly as the proof of Theorem 1. Let G be the graph in an instance I of of 2DDP. Create a new network denoted by $G^{'} = (V^{'}, E^{'})$ in Figure 3. Let $E_1 = E^{'} - E$ be the set of edges that the subgraph H of $G^{'}$ should contain. We define the following performance functions for the edges of $E^{'}$: the edges (w_i, v_1), (v_1, u_i), (u_i, v_2), (v_2, u_i), (u_i, v_1), $(w_i^{'}, v_2)$ $(i = 1, \cdots, k)$, are given the latency function $f(x) = 0$ for $x \leq 1/k$ and $f(x) = x - 1/k$ for $x \geq 1/k$. And all other agents are given performance function $f(x) = 0$. We consider a k-splittable unweighted bottleneck game with six players

that uses the network G'. Player 1 has a bandwidth request $(s_1, t_1, 1)$ (player 1 has to move a unit of bandwidth from s_1 to t_1). Player 2 has a bandwidth request $(s_2, t_2, 1)$. Player 3 has a bandwidth request $(s_1, v_1, 1)$. Player 4 has a bandwidth request $(s_2, v_2, 1)$. Player 5 has a bandwidth request $(v_1, v_2, 1)$. Player 6 has a bandwidth request $(v_2, v_1, 1)$. The new instance I' can be constructed from I in polynomial time. We can show the following two statements and thus complete the proof.

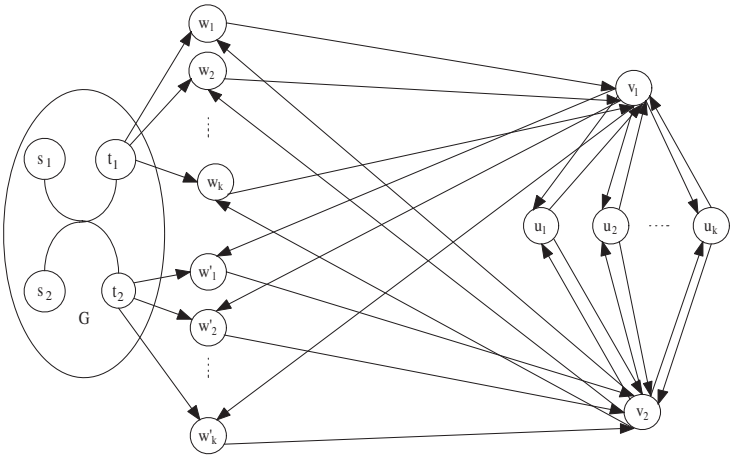


Fig. 3. Proof of Theorem 4

(1) If I is a "yes" instance of 2DDP, then G' contains a subgraph H of G' with $B(H) = 0$.
(2) If I is a "no" instance of 2DDP, then $B(H) > 0$ for all subgraphs H of G'. □

References

1. E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 59-73, 2004.
2. B. Awerbuch, Y. Azar, and A. Epstein. The price of routing unsplittable flow. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 331-337, 2005.
3. Y. Azar and A. Epstein. The hardness of network design for unsplittable flow with selfish users. *Proceedings of the 3rd Workshop of Approximation and Online Algoeithms (WAOA)*, pages 41-54, 2005.
4. G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 101-113, 2002.

5. R. Banner and A. Orda. Bottleneck Routing Games in Communication Networks. *Proceedings of the 25th Conference on Computer Communications (IEEE INFOCOM)*, 2006.
6. D. Braess. Uber ein paradoxon der verkehrsplanung. *Unternehmensforschung*, 12: 258-268, 1968.
7. G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67-73, 2005.
8. S. Fortune, J.E.Hopcroft, and J.C. Wyllie. The directed subgrph homeomorphism problem. *Theoretical Computer Science*, 10(2): 111-121, 1980.
9. M.R. Garey and D.S. Johnson. *Computers and Intractability*, W.H. Freeman and Co., 1979.
10. E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 404-413, 1999.
11. C. Meyers, Ph.D thesis, 2006, MIT.
12. J.D. Murchland. Braess'paradox of traffic flow. *Transportation Research*, 4: 391-394, 1970.
13. C.H. Papadimitriou. Algorithms, games, and the Internet. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing* , pages 749-753, 2001.
14. T. Roughgarden. Designing networks for selfish users is hard. *Proceedings of the 42th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* pages 472-481, 2001.
15. T. Roughgarden. The price of anarchy is independent of the network topology. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 428-437, 2002.
16. T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM* 49(2): 236-259, 2002.

A Polynomial Time Algorithm for Finding Linear Interval Graph Patterns

Hitoshi Yamasaki and Takayoshi Shoudai

Department of Informatics, Kyushu University
Motooka 744, Fukuoka 819-0395, Japan
{h-yama,shoudai}@i.kyushu-u.ac.jp

Abstract. A graph is an interval graph if and only if each vertex in the graph can be associated with an interval on the real line such that any two vertices are adjacent in the graph exactly when the corresponding intervals have a nonempty intersection. A number of interesting applications for interval graphs have been found in the literature. In order to find structural features common to structural data which can be represented by intervals, this paper proposes new interval graph structured patterns, called linear interval graph patterns, and a polynomial time algorithm for finding a minimally generalized linear interval graph pattern explaining a given finite set of interval graphs.

1 Introduction

A graph $G = (V, E)$ is an *interval graph* if and only if for each vertex $v \in V$, a closed interval I_v in the real line can be associated such that for each pair of vertices $u, v \in V$ $(u \neq v)$, $(u, v) \in E$ if and only if $I_u \cap I_v \neq \emptyset$. For example, in Fig. 1, G is an interval graph which has its interval representation $R(G)$. One important application of interval graphs is a physical mapping in genome research, that is, to reconstruct the relative positions of fragments of DNA along the genome from certain pairwise overlap information [12]. Reliable and complete overlap information is very costly and practically not available. Probe interval graphs were introduced by Zhang et al.[7,13] to represent only partial overlap information. As another application, the mutual exclusion scheduling problem is known to be formalized by a subclass of interval graphs [3].

It is known that the general problem of deciding graph isomorphism appears to be hard. However for some special classes of graphs, isomorphism can be decided efficiently. The class of interval graphs is the case. Lueker and Booth [6] gave a linear time algorithm for interval graph isomorphism. The Lueker and Booth's isomorphism algorithm uses a data structure called a *labeled PQ-tree* which is an extension of *PQ-tree* [2]. PQ-trees are used to represent the permutations of a set in which various subsets of the set occur consecutively.

In order to represent interval patterns common to interval structured data, we propose *interval graph patterns* which consist of interval graph structures and simplicial variables. The formal definition is described in Section 2. An interval

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 67–78, 2007.
© Springer-Verlag Berlin Heidelberg 2007

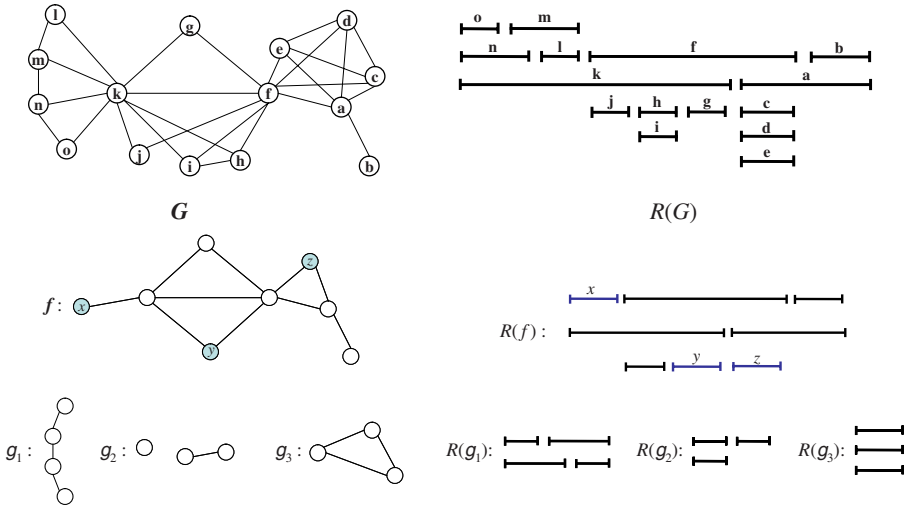


Fig. 1. An interval graph G (its interval representation $R(G)$) and an interval graph pattern f (its interval representation $R(f)$). The interval graph G is the instance of f by θ where $\theta = [x/g_1, y/g_2, z/g_3]$.

graph pattern is called *linear* if all variables in it have mutually distinct variable labels. For an interval graph pattern g, the interval graph language of g, denoted by $L(g)$, is the set of all interval graphs which are obtained from g by substituting arbitrary interval graphs for all variables in g. In Fig. 1, f is a linear interval graph pattern with three variables of variable labels x, y and z, and $R(f)$ is an interval representation of f. The interval graph G is obtained from f by replacing x, y and z with g_1, g_2 and g_3, respectively. Then $G \in L(f)$.

For a finite set of interval graphs S, a *minimally generalized linear interval graph pattern* explaining S is defined as a linear interval graph pattern g such that $S \subseteq L(g)$ and there exists no linear interval graph pattern g' satisfying $S \subseteq L(g') \subsetneq L(g)$. In this paper, we give a polynomial time algorithm for finding a minimally generalized linear interval graph pattern explaining a given finite set of interval graphs. As related works, Horváth et al. [5] developed a frequent subgraph mining algorithm in outerplanar graphs that works in incremental polynomial time. We gave polynomial time learning algorithms for linear tree patterns with internal structured variables [9,10] and two-terminal series-parallel graph patterns [11] in the framework of inductive inference.

2 Interval Graph Patterns

For a graph $G = (V, E)$ and a vertex $u \in V$, the set of vertices adjacent to u, called the *neighborhood* of u, is denoted by $N_G(u)$. Occasionally we call $N_G(u)$ the *open* neighborhood of u and $N_G(u) \cup \{u\}$ the *closed* neighborhood of u.

We denote the *closed* neighborhood of u by $N_G[u]$. A clique of a graph G is a complete subgraph of G. A vertex u is *simplicial* if the subgraph induced from $N_G(u)$ is a clique.

Definition 1. Let $G = (V, E)$ be an interval graph. Let V_g and H_g be a partition of V such that $V_g \cup H_g = V$ and $V_g \cap H_g = \emptyset$. A triplet $g = (V_g, E, H_g)$ is called an *interval graph pattern* if all vertices in H_g are simplicial and no two vertices in H_g are adjacent. Let X be an infinite alphabet. We call an element in X a *variable label*. Each variable in an interval graph pattern is labeled with a variable label in X. We call elements in V_g and H_g a vertex and a *variable*, respectively.

For any interval graph pattern g, we denote the sets of vertices, edges, and variables by $V(g)$, $E(g)$, and $H(g)$, respectively. The *size* of g is defined as $|V(g)| + |H(g)|$. For any $u \in H(g)$, we denote the variable label of u by $x(u)$. An interval graph pattern g is called *linear* if all variables in $H(g)$ have mutually distinct variable labels in X, that is, for any two variables $u, v \in H(g)$, $x(u) \neq x(v)$. We denote the set of all interval graphs by $\mathcal{IG}$, the set of all interval graph patterns by $\mathcal{IGP}$, and the set of all linear interval graph patterns by $\mathcal{LIGP}$.

Definition 2. Let f and g be two interval graph patterns in $\mathcal{IGP}$. We say that f and g are *isomorphic*, denoted by $f \cong g$, if there exists a bijection φ from $V(f) \cup H(f)$ to $V(g) \cup H(g)$ such that (1) for any $u, v \in V(f) \cup H(f)$, $(u, v) \in E(f)$ if and only if $(\varphi(u), \varphi(v)) \in E(g)$, (2) $\varphi(V(f)) = V(g)$, and (3) for any $u, v \in H(f)$, $x(u) = x(v)$ if and only if $x(\varphi(u)) = x(\varphi(u))$.

Definition 3. Let g_1 and g_2 be interval graph patterns in $\mathcal{IGP}$. For a variable label $x \in X$, the form x/g_2 is called a *binding* for x. A new interval graph pattern $g_1[x/g_2]$ is obtained from g_1 and x/g_2 by connecting all vertices and variables in $V(g_2) \cup H(g_2)$ to all vertices in $N_{g_1}(h)$ for each variable h such that $x(h) = x$, and then removing h from g_1. Formally $g_3 = g_1[x/g_2]$ is defined as $V(g_3) = V(g_1) \cup V(g_2)$, $E(g_3) = E(g_1) \cup E(g_2) \cup \{(u, v) \mid u \in N_{g_1}(h), v \in V(g_2) \cup H(g_2)\} - \{(u, h) \mid u \in N_{g_1}(h)\}$, and $H(g_3) = H(g_1) \cup H(g_2) - \{h\}$. A *substitution* θ is a finite collection of bindings $[x_1/g_1, \ldots, x_n/g_n]$, where x_i's are mutually distinct variable labels in X.

The interval graph pattern $f\theta$, called the *instance* of f by θ, is obtained by applying all the bindings x_i/g_i on f simultaneously. We give an example of a substitution in Fig. 1. For an interval graph G and an interval graph pattern g, we say that g *matches* G if there exists a substitution θ such that $G \cong g\theta$.

Definition 4 ([6]). A *labeled PQ-tree* is a node-labeled ordered tree whose internal nodes consist of two classes, namely P-nodes and Q-nodes. Each leaf and P-node is labeled with a nonnegative integer ℓ and each Q-node with m children is labeled with a lexicographically sorted sequence of m' pairs of integers $(i_1, j_1), \ldots, (i_{m'}, j_{m'})$ where $m' \geq 1$ and $1 \leq i_k \leq j_k \leq m$ for $k = 1, 2, \ldots, m'$. We denote the label of a node a by $Label(a)$. We say that two labeled PQ-trees T_1 and T_2 are equivalent, denoted by $T_1 \equiv T_2$, if T_2 is obtained from T_1 by applying any combination of the following transformations:

(a) arbitrarily reordering the children of a P-node, and
(b) reversing the ordering of the m children of a Q-node and replacing the label $(i_1, j_1), \ldots, (i_{m'}, j_{m'})$ with the lexicographically sorted sequence of $(m+1-j_1, m+1-i_1), \ldots, (m+1-j_{m'}, m+1-i_{m'})$.

For a node a of a PQ-tree T, we denote the subtree induced from a and all descendants of a by $T[a]$. For a Q-node a and its label $Label(a)$, we denote the label after applying the transformation (b) to a by $Label^r(a)$. The *frontier* of a PQ-tree T is the ordering of its leaves obtained by reading them from left to right. The frontier of a node a, denoted by $F(a)$, is the frontier of $T[a]$. An ordering of the leaves of T is *consistent* with T if it is the frontier of a PQ-tree equivalent to T. For each vertex u of a graph G, let $C(u)$ be the set of maximal cliques which contain u. It is known that G is an interval graph if and only if there exists a linear ordering of all maximal cliques of G such that for each vertex u of G, the elements of $C(u)$ appear consecutively within the ordering. Lueker and Booth [6] gave a linear time algorithm, given an interval graph G, to construct a labeled PQ-tree T so that there is a bijection ψ from the set of all leaves of T to the set of all maximal cliques of G satisfying the conditions(1)–(3).

(1) Let k be the number of leaves of T. An ordering $(b_1, \ldots, b_k)$ of the leaves of T is consistent with T if and only if for any vertex $u \in V(G)$, an element of $C(u)$ appears consecutively in $(\psi(b_1), \ldots, \psi(b_k))$.
(2) For any vertex $u \in V(G)$, the *characteristic node* of u is the deepest node a in T such that $F(a)$ contains all elements of $C(u)$. For any leaf or P-node a, $Label(a) = |\{u \in V(G) \mid a \text{ is the characteristic node of } u\}|$.
(3) For any Q-node a and its children $c_1, \ldots, c_m$, $Label(a)$ contains (i, j) $(1 \leq i \leq j \leq m)$ if and only if there is a vertex $u \in V(G)$ such that $\psi^{-1}(C(u))$ is the set of all leaves in the frontiers of $c_i, \ldots, c_j$.

We denote the labeled PQ-tree obtained from an interval graph G by $T(G)$. We give an example of a linear interval graph G and its labeled PQ-tree $T(G)$ in Fig. 2. For a labeled PQ-tree T, we denote the sets of nodes and edges by $V(T)$ and $E(T)$, respectively.

Theorem 1 ([6]). *For interval graphs G_1 and G_2, $G_1 \cong G_2$ if and only if $T(G_1) \equiv T(G_2)$.*

Definition 5. Let g be an interval graph pattern and $G = (V(g) \cup H(g), E(g))$. The *PQ-tree pattern* of g is the labeled PQ-tree with variables, denoted by $T(g)$, which is obtained from $T(G)$ by, for all characteristic nodes $a \in V(T(G))$ of the variables $h \in H(g)$, decreasing the label of a by one and attaching the variable label of h to a as its variable label. We note that the characteristic node of any variable is a leaf in $T(G)$ since all variables are simplicial in g.

A PQ-tree pattern is called *linear* if all variables in it have mutually distinct variable labels. We give an example of an interval graph pattern f and its PQ-tree pattern $T(f)$ in Fig. 2. We denote the set of all labeled PQ-trees by $\mathcal{PQT}$, the set of all PQ-tree patterns by $\mathcal{PQTP}$, and the set of all linear PQ-tree patterns by $\mathcal{LPQTP}$.

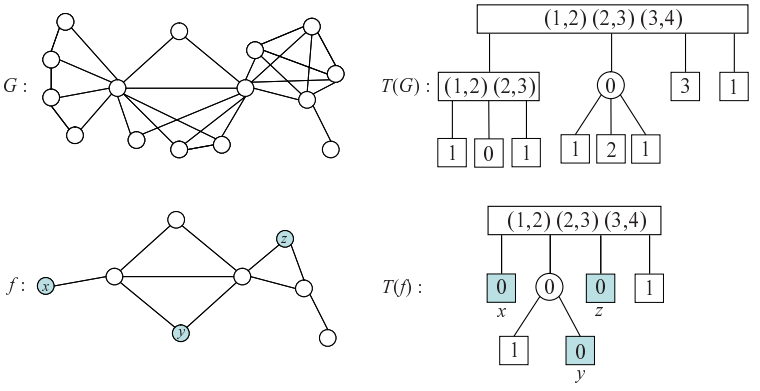


Fig. 2. An interval graph G has 8 maximal cliques (see also Fig. 1). Each maximal clique corresponds to one of leaves of $T(G)$. $T(f)$ is the PQ-tree pattern of a linear interval graph pattern f (Definition 5). In this and subsequent figures, P-nodes are drawn as circles, Q-nodes and leaves as rectangles, and variables as dot-filled rectangles.

Definition 6. Let t_1 and t_2 be PQ-tree patterns and r the root of t_2. We assume that if a PQ-tree pattern consists of a single node, the label of the node is a positive integer. Let h be a variable of t_1 whose variable label is $x \in X$. Let x/t_2 be a binding for x. A new PQ-tree pattern $t_1[x/t_2]$ is obtained from t_1 by applying x/t_2 to t_1 in the following way (see Fig. 3).

1. If r is a Q-node with k children, r is identified with h (the new node is a Q-node) and its label is the lexicographically sorted sequence of the concatenation of $Label(r)$ and $\underbrace{(1,k),\ldots,(1,k)}_{Label(h) \text{ times}}$.
2. If either r is a P-node or t_2 is a single node, and
   (a) if $Label(r) + Label(h) = 0$ and the parent of h is a P-node, the children of r are directly connected to the parent of h (h and r are removed),
   (b) otherwise, r is identified with h (the new node is a P-node) and its label is $Label(r) + Label(h)$.

A finite collection of bindings $\tau = [x_1/t_1, \ldots, x_n/t_n]$ is called a *substitution*, where x_i's are mutually distinct variable labels in X.

For example, in Fig. 4, the bindings x/t_1, y/t_2 and z/t_3 are the cases (1), (2-a) and (2-b), respectively. The PQ-tree pattern $t\tau$, called the *instance* of t by τ, is obtained by applying all the bindings x_i/t_i on t simultaneously. We have the next lemma for an interval graph pattern and a PQ-tree pattern.

Lemma 1. *Let f and g be interval graph patterns and x a variable label of a vertex in $H(f)$. Then, $T(f)[x/T(g)] \equiv T(f[x/g])$.*

For a PQ-tree T and a PQ-tree pattern t, we say that t *matches* T if there exists a substitution τ such that $T \equiv t\tau$.

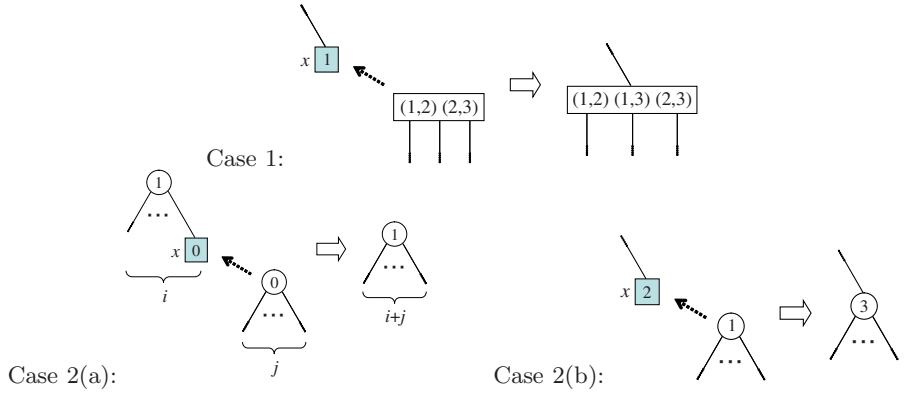


Fig. 3. Replacements of PQ-trees in Definition 6

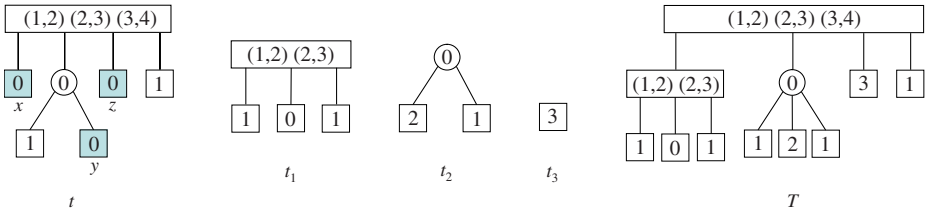


Fig. 4. t_1, t_2 and t_3 are the labeled PQ-trees of g_1, g_2 and g_3 in Fig. 1, respectively. And t is the PQ-tree patterns of f in Fig. 1. T is the instance of t by a substitution $\tau = [x/t_1, y/t_2, z/t_3]$.

3 Interval Graph Pattern Matching

Unfortunately the general interval graph pattern matching problem is hard to compute. We can give a polynomial time reduction from the CLIQUE problem to it.

Theorem 2. *The problem of deciding, given an interval graph pattern $g \in \mathcal{IGP}$ and an interval graph $G \in \mathcal{IG}$, whether or not g matches G is NP-complete.*

Here we give an outline of the proof. It is easy to see that the problem is in NP. We reduce the CLIQUE problem, which is to decide whether a given graph $G = (V, E)$ has a clique of size k, to the matching problem. We assume that $V = \{1, 2, \ldots, |V|\}$. Each vertex in V is transformed uniquely into an interval graph and its labeled PQ-trees (Fig. 5). By using these unique PQ-trees, we transform the graph G to a labeled PQ-tree T_G. The root of T_G has just $|E|$ children each of which represents an edge in G (Fig. 6). We can see that there is an interval graph G' such that T_G is the representation of G'. In a similar way,

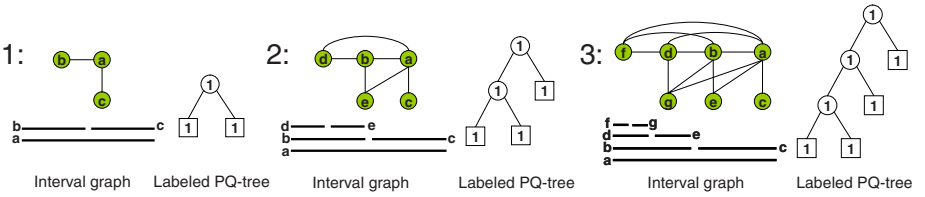


Fig. 5. The first three interval graphs and their labeled PQ-trees which represent the digits in $V = \{1, 2, \ldots, |V|\}$

we transform k-clique to a PQ-tree pattern t_k. We assume that all vertices of k-clique are attached with different variable labels in X. The root of t_k has just $|E|$ children which represent $k(k-1)/2$ edges of k-clique and $|E| - k(k-1)/2$ variables for garbage collections. It is easy to see that there is an interval graph pattern g_k such that t_k is the representation of g_k. Finally we can show that g_k matches G' if and only if G has a k-clique.

Next we give a polynomial time algorithm for deciding, given a linear interval graph pattern $g \in \mathcal{LIGP}$ and a given interval graph $G \in \mathcal{IG}$, whether or not g *matches* G.

Firstly, we transform g and G into a linear PQ-tree pattern $T(g)$ and a labeled PQ-tree $T(G)$, respectively. These transformations need $O(n+m)$ and $O(N+M)$ times, respectively, where $n = |V(g)| + |H(g)|$, $m = |E(g)|$, $N = |V(G)|$, and $M = |E(G)|$ [6].

From Theorem 1 and Lemma 1, there is a substitution θ such that $G \cong g\theta$ if and only if there is a substitution τ such that $T(G) \cong T(g)\tau$. Secondly, we decide whether or not there is a substitution τ such that $T(G) \cong T(g)\tau$. Below, we briefly denote $T(G)$ by T and $T(g)$ by t. We call a set of nodes of a given labeled PQ-tree T a *candidate set*. We assign a candidate set to each node of a given linear PQ-tree pattern t. We denote the candidate set of a node a in $V(t)$ by $NS(a)$. For a node b in $V(T)$, $b \in NS(a)$ if and only if $t[a]$ matches $T[b]$, where $t[a]$ is the PQ-tree pattern induced from a and all the nodes and variables which are descendants of a. The following algorithm computes $NS(a)$ for each $a \in V(t)$ by using $NS(a_1), \ldots, NS(a_\ell)$ where $a_1, \ldots, a_\ell$ are all children of a. The algorithm terminates when a candidate set is assigned to the root of t. We denote the root of t by r_t. We can show that $NS(r_t)$ contains the root of T if and only if t matches T.

Obviously, for any substitution τ and a node $a \in V(t)$, the depth of a in $t\tau$ is equal to that of a of t. Thus, if b is in $NS(a)$ for any node $b \in V(T)$, the depth of b is equal to that of a. We show how to assign a candidate set $NS(a)$ to a node a in $V(t)$. The assignment method depends on the type of a node a.

Leaf: For a leaf a, $NS(a) := \{b \in V(T) \mid b$ is a leaf of T and $Label(b) = Label(a)\}$.

Variable: Let us suppose that a labeled PQ-tree T is substituted for a variable a. If T is a single node, its label must be at least 1. And if the root of T

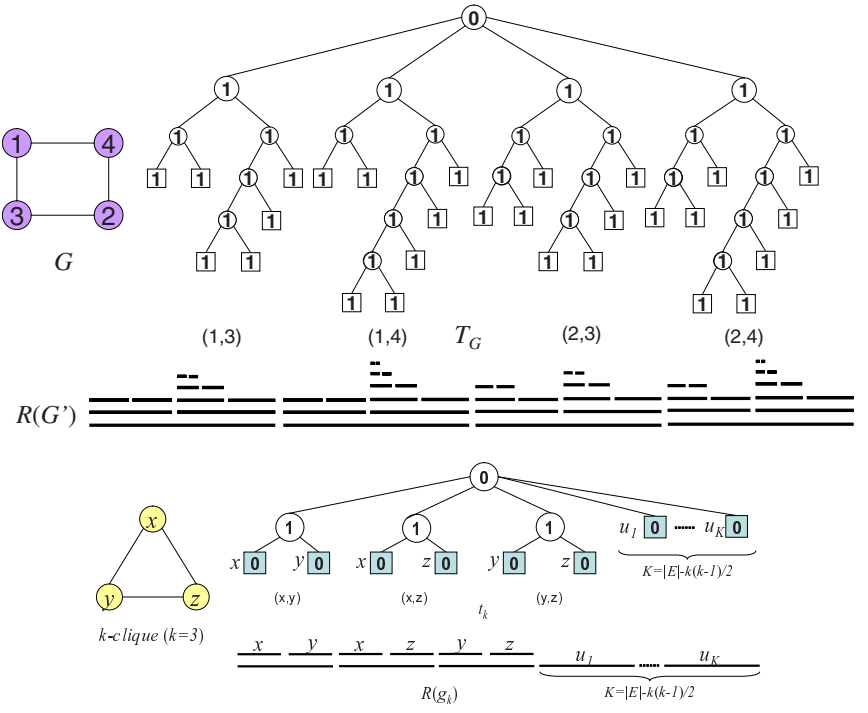


Fig. 6. A graph $G = (V, E)$ and a positive integer $k = 3$ are supposed to be instances of CLIQUE. Both G and 3-clique are transformed into an interval graph G' and an interval graph pattern g_3, respectively. In this figure, we only draw the labeled PQ-tree T_G and the PQ-tree pattern t_3 instead of G and g_3.

is a P-node, the label of the root is at least 0. Therefore $NS(a)$ is the set consisting of the following nodes:

(1) all leaves b such that $Label(b) > Label(a)$,
(2) all P-nodes b such that $Label(b) \geq Label(a)$, and
(3) all Q-nodes b such that $Label(b)$ contains at least $Label(a)$ pairs of the form $(1, m)$ where m is the number of children of b.

P-node: Let us suppose that a P-node a has ℓ children $a_1, \ldots, a_\ell$. If a has a variable with label 0 as its child, any node $b \in NS(a)$ must be a P-node which has at least ℓ children and $Label(b) = Label(a)$. Otherwise, any node $b \in NS(a)$ must be a P-node which has just ℓ children and $Label(b) = Label(a)$. The nodes in $NS(a)$ are picked up in all nodes $b \in V(T)$ satisfying the above condition by using the following procedure. Let $b_1, \ldots, b_m$ $(m \geq \ell)$ be the children of a P-node b. We note that $t[a]$ matches $T[b]$ if and only if there is an index subsequence $m_1, \ldots, m_\ell$ $(1 \leq m_1 < \cdots < m_\ell \leq m)$ such that $b_{m_k} \in NS(a_k)$ $(1 \leq k \leq \ell)$. Therefore we compute the maximum matching problem for a bipartite graph $G = (U, V, E)$ where $U = \{NS(a_1), \ldots, NS(a_\ell)\}$,

$V = \{b_1, \ldots, b_m\}$, $E = \{(NS(a_i), b_j) \mid 1 \leq i \leq \ell, 1 \leq j \leq m, b_j \in NS(a_i)\}$). A node b is in $NS(a)$, i.e., $t[a]$ matches $T[b]$, if and only if the bipartite graph $G = (U, V, E)$ has a matching of size ℓ.

Q-node: It is easy to see that any node $b \in NS(a)$ must be a Q-node which has just ℓ children and satisfies either $Label(b) = Label(a)$ or $Label(b) = Label^r(a)$. Let $b_1, \ldots, b_\ell$ be the children of a Q-node b. If $Label(b) = Label(a)$, a child b_i must be in $NS(a_i)$ for all i ($1 \leq i \leq \ell$), and if $Label(b) = Label^r(a)$, a child $b_{\ell-i+1}$ must be in $NS(a_i)$ for all i ($1 \leq i \leq \ell$). The candidate set $NS(a)$ is the set of all Q-nodes b which satisfies the above conditions.

We can show that the above PQ-tree pattern matching algorithm works in $O(nN^{1.5})$ time. Then the total complexity for linear interval graph pattern matching is $O(nN^{1.5} + m + M)$ time.

Theorem 3. *The problem of deciding, given a linear interval graph pattern $g \in \mathcal{LIGP}$ and an interval graph $G \in \mathcal{IG}$, whether or not g matches G is solvable in polynomial time.*

4 Minimally Generalized Linear Interval Graph Patterns

For an interval graph pattern g, let $L(g) = \{G \in \mathcal{IG} \mid g \text{ matches } G\}$. For a finite set of interval graphs $S \subset \mathcal{IG}$, a *minimally generalized linear interval graph pattern* explaining S is defined as a linear interval graph pattern $g \in \mathcal{LIGP}$ such that $S \subseteq L(g)$ and there exists no linear interval graph pattern $g' \in \mathcal{LIGP}$ satisfying $S \subseteq L(g') \subsetneq L(g)$.

For a PQ-tree pattern t, the PQ-tree pattern language $L_T(t)$ of t is defined as $\{T \in \mathcal{PQT} \mid t \text{ matches } T\}$. For a finite set of PQ-trees $S_T \subset \mathcal{PQT}$, a *minimally generalized linear PQ-tree pattern* explaining S_T is defined as a linear PQ-tree pattern $t \in \mathcal{LPQTP}$ such that $S_T \subseteq L_T(t)$ and there exists no linear PQ-tree pattern $t' \in \mathcal{LPQTP}$ satisfying $S_T \subseteq L_T(t') \subsetneq L_T(t)$.

Let S be a finite set of interval graphs and $S_T = \{T(g) \mid g \in S\}$. For a linear interval graph pattern g, $G \in L(g)$ if and only if $T(G) \in L_T(T(g))$. Then, $S \subseteq L(g)$ if and only if $S_T \subseteq L_T(T(g))$. Moreover, for two linear interval graph patterns g and g', $L(g') \subseteq L(g)$ if and only if $L_T(T(g')) \subseteq L_T(T(g))$. Therefore we have the following lemma.

Lemma 2. *For $S \subset \mathcal{IG}$ and $g \in \mathcal{LIGP}$, g is a minimally generalized linear interval graph pattern explaining S if and only if $T(g)$ is a minimally generalized linear PQ-tree pattern explaining S_T.*

The *size* of a PQ-tree pattern t is defined as $|V(t)| + |H(t)|$. Let S_T be a finite set of PQ-trees. Obviously there exists a minimally generalized linear PQ-tree pattern explaining S_T which has the maximum size in $C = \{t \in \mathcal{PQTP} \mid S_T \subseteq L_T(t)\}$. We define the following 4 classes of linear PQ-tree patterns s, p, $q_{(w,Z)}$, and r (Fig. 7).

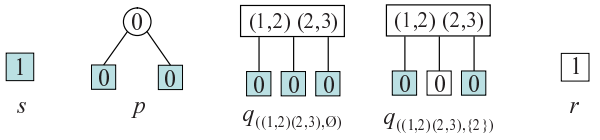


Fig. 7. A basic set of linear PQ-tree patterns. The 3rd and 4th linear PQ-tree patterns are two examples of the form $q(w, Z)$ which are obtained from the PQ-tree of minimum size in S_T.

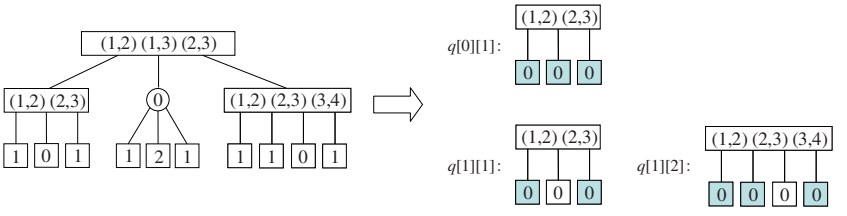


Fig. 8. A linear PQ-tree pattern $q[i][j]$ is obtained from the j-th Q-node of depth i of the left PQ-tree. Then we have $q[0][1] = q((1,2)(2,3), \emptyset)$, $q[1][1] = q((1,2)(2,3), \{2\})$ and $q[1][2] = q((1,2)(2,3)(3,4), \{3\})$.

- The linear PQ-tree pattern s consists of only one variable of label 1.
- The linear PQ-tree pattern p consists of one P-node with label 0 and two variables with label 0. Both variables are the children of the P-node.
- The linear PQ-tree patterns $q_{(w,Z)}$ consist of one Q-node with label w and a series of variables or leaves with label 0. All the variables and leaves are the children of the Q-node. The index (w, Z) satisfies the following conditions: w is the label of the Q-node which does not contain a pair $(1, m)$, where m be the number of children of the Q-node, and Z is the set $\{i \in \{1, \ldots, m\} \mid w$ contains two pairs (j, i) and (i, k), and the i-th child of the Q-node is a leaf$\}$.

  From the PQ-tree of minimum size in S_T, we generate all linear PQ-tree patterns of the form $q_{(w,Z)}$. For example, in Fig. 8, we obtain three linear PQ-tree patterns $q((1,2)(2,3), \emptyset)$, $q((1,2)(2,3), \{2\})$ and $q((1,2)(2,3)(3,4), \{3\})$ from the left PQ-tree.
- The linear PQ-tree pattern r consists of only one leaf node of label 1.

The PQ-tree pattern consisting of only one variable with label 0 is only a linear PQ-tree pattern whose language properly contains $L_T(s)$, $L_T(p)$, $L_T(q_{(w,Z)})$, and $L_T(r)$. Let t be a linear PQ-tree pattern and h a variable in $H(t)$ with variable label x. It is easy to see that $L_T(t)$ properly contains $L_T(t[x/s])$, $L_T(t[x/p])$, $L_T(t[x/q_{(w,Z)}])$, and $L_T(t[x/r])$. Moreover, if $T \in L_T(t)$, we have either $T \in L_T(t[x/s])$, $T \in L_T(t[x/p])$, $T \in L_T(t[x/r])$, or there are w and Z such that $T \in L_T(t[x/q_{(w,Z)}])$.

We show an overview of our algorithm for finding a minimally generalized PQ-tree pattern explaining S_T. We start with the linear PQ-tree pattern t consisting

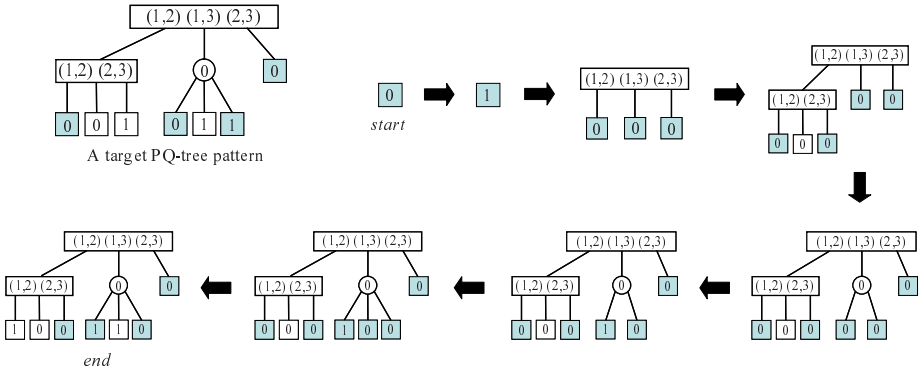


Fig. 9. A refinement process of our algorithm for finding a minimally generalized PQ-tree pattern explaining S_T

of only one variable with label 0. This pattern t generates all labeled PQ-trees. Then we repeatedly apply a combination of the following refinements to all variables of a temporary linear PQ-tree pattern t while $S_T \subset L_T(t)$. Let h be a variable of t whose variable label is x.

(1) **if** $S_T \subseteq L_T(t[x/s])$ **then** $t := t[x/s]$;
(2) **if** $S_T \subseteq L_T(t[x/p])$ **then** $t := t[x/p]$;
(3) **if** there are w and Z such that $S \subseteq L_T(t[x/q_{(w,Z)}])$ **then** $t := t[x/q_{(w,Z)}]$;

If none of the above refinements can be applied to the current linear PQ-tree pattern t, we repeatedly apply the next refinement while $S_T \subseteq L_T(t)$.

(4) **if** $S_T \subseteq L_T(t[x/r])$ **then** $t := t[x/r]$;

An example of a process of refinements is shown in Fig. 9. We can show that for a given finite set of labeled PQ-trees $S_T \subset \mathcal{PQT}$, the above algorithm correctly outputs a minimally generalized linear PQ-tree patterns explaining S_T.

Since the PQ-tree pattern matching algorithm in Section 4 works in $O(nN^{1.5})$ time, we need $O(|S_T|N_{\min}N_{\max}^{1.5})$ time to decide whether or not $S_T \subseteq L_T(t)$, where $N_{\min}$ and $N_{\max}$ are the minimum and maximum sizes of PQ-trees in S_T. Since the refinement operations (1)–(4) are applied at most $O(N_{\min}^2)$ times, the total time complexity of this algorithm is $O(|S_T|N_{\min}^3 N_{\max}^{1.5})$ time.

For a given set $S \subset \mathcal{IG}$ of interval graphs, we find a minimally generalized linear interval graph pattern explaining S in the following way. First we transform the set S into a set of labeled PQ-trees $S_T \subset \mathcal{PQT}$. The transformation needs $O(|S|(n_{\max} + m_{\max}))$ time, where $n_{\max}$ (resp. $m_{\max}$) is the maximum number of vertices (resp. edges) of interval graphs in S. Then we find a minimally generalized linear PQ-tree pattern explaining S_T. It needs $O(|S|n_{\min}^3 n_{\max}^{1.5})$ time, where $n_{\min}$ is the minimum number of vertices of interval graphs in S. Finally we transform the obtained minimally generalized linear PQ-tree pattern explaining S_T into the minimally generalized linear interval graph pattern explaining S. The time complexity of the algorithm is $O(|S|(n_{\min}^3 n_{\max}^{1.5} + m_{\max}))$.

Theorem 4. *For a given finite set of interval graphs $S \subset \mathcal{IG}$, a minimally generalized linear interval graph pattern explaining S can be computed in polynomial time.*

5 Conclusions and Future Work

From Theorems 3 and 4, we can conclude that the class of linear interval graph pattern languages is polynomial time inductively inferable from positive data by using the theorems in [1,8]. We are now considering the polynomial time learnability of graph languages on the classes of graph structured patterns expressing probe interval graphs, chordal graphs, and outerplanar graphs.

References

1. D. Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21, pp. 46–62, 1980.
2. K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13, pp. 335–379, 1976.
3. F. Gardi. The mutual exclusion scheduling problem for proper interval graphs. LIF Research Report 02-2002, Laboratoire d'Informatique Fondamentale de Marseille, April, 2002.
4. J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.*, 2(3), pp. 225–231, 1973.
5. T. Horváth, J. Ramon, and S. Wrobel. Frequent Subgraph Mining in Outerplanar Graphs. *Proc. KDD 2006*, pp. 197–206, 2006.
6. G.S. Lueker and K.S. Booth. A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *J. ACM*, 26(2), pp. 183–195, 1979.
7. F.R. McMorris, C. Wang, and P. Zhang. On probe interval graphs. *Disc. Appl. Math.*, 88, pp. 315–324, 1998.
8. T. Shinohara. Polynomial time inference of extended regular pattern languages. *Proc. RIMS Symp. Software Science and Engineering, Springer-Verlag, LNCS 147*, pp. 115–127, 1982.
9. Y. Suzuki, T. Shoudai, T. Miyahara, and S. Matsumoto. Polynomial Time Inductive Inference of Ordered Tree Languages with Height-Constrained Variables from Positive Data. *Proc. PRICAI 2004, Springer-Verlag, LNAI 3157*, pp. 211–220, 2004.
10. Y. Suzuki, T. Shoudai, T. Miyahara, and T. Uchida. Ordered Term Tree Languages Which Are Polynomial Time Inductively Inferable from Positive Data. *Theor. Comput. Sci.*, 350, pp. 63–90, 2006.
11. R. Takami, Y. Suzuki, T. Uchida, T. Shoudai, and Y. Nakamura. Polynomial Time Inductive Inference of TTSP Graph Languages from Positive Data. *Proc. ILP 2005, Springer-Verlag, LNAI 3625*, pp. 366-383, 2005.
12. P. Zhang, E.A. Schon, S.G. Fisher, E. Cayanis, J. Weiss, S. Kistler, and P.E. Bourne. An algorithm based on graph theory for the assembly of contings in physical mapping of DNA. *CABIOS*, 10, pp. 309–317, 1994.
13. P. Zhang. Probe Interval Graph and Its Applications to Physical Mapping of DNA. *Int. Conf. Computational Molecular Biology* RECOMB 2000 (Poster Session), 2000.

Elementary Differences Among Jump Hierarchies

Angsheng Li*

State Key Laboratory of Computer Science, Institute of Software
Chinese Academy of Sciences, P.O. Box 8718, Beijing 100080, P.R. China
angsheng@ios.ac.cn

Abstract. It is shown that $\mathrm{Th}(\mathbf{H}_1) \neq \mathrm{Th}(\mathbf{H}_n)$ holds for every $n > 1$, where $\mathbf{H}_m$ is the upper semi-lattice of all high$_m$ computably enumerable (c.e.) degrees for $m > 0$, giving a first elementary difference among the highness hierarchies of the c.e. degrees.

1 Introduction

Let $n \geq 0$. We say that a *computably enumerable* (c.e.) degree $\mathbf{a}$ is high$_n$ (or low$_n$), if $\mathbf{a}^{(n)} = \mathbf{0}^{(n+1)}$ (or $\mathbf{a}^{(n)} = \mathbf{0}^{(n)}$), where $\mathbf{x}^{(n+1)} = (\mathbf{x}^{(n)})'$, $\mathbf{x}^{(0)} = \mathbf{x}$, $\mathbf{y}'$ is the *Turing jump* of $\mathbf{y}$. Let $\mathbf{H}_n$ ($\mathbf{L}_n$) be the set of all high$_n$ (low$_n$) c.e. degrees. For $n = 1$, we also call an element of $\mathbf{H}_1$ (or $\mathbf{L}_1$) high (or low).

Sacks [1963] showed a (Sacks) *Jump Theorem* that for any degrees $\mathbf{s}$ and $\mathbf{c}$, if $\mathbf{s}$ is c.e.a in $\mathbf{0}'$ and $\mathbf{0} < \mathbf{c} \leq \mathbf{0}'$, then there exists a c.e. degree $\mathbf{a}$ such that $\mathbf{a}' = \mathbf{s}$ and $\mathbf{c} \not\leq \mathbf{a}$, and that there exists a non-trivial high c.e. degree. Note that an easy priority injury argument gives a nonzero low c.e. degree. By relativising the construction of high and low c.e. degrees to $\mathbf{0}^{(n)}$ and using the Sacks Jump Theorem, it follows that for all n, $\mathbf{H}_n \subset \mathbf{H}_{n+1}$ and $\mathbf{L}_n \subset \mathbf{L}_{n+1}$. And Martin [1966a], Lachlan [1965] and Sacks [1967] each proved that the union of the high/low hierarchies does not exhaust the set $\mathcal{E}$ of the c.e. degrees. And Sacks [1964] proved the (Sacks) *Density Theorem* of the c.e. degrees. While early researches were aiming at characterisations of the high/low hierarchy. The first result on this aspect is the *Martin* [1966b] *Characterisation of High Degrees*: A set A satisfies $\emptyset'' \leq_{\mathrm{T}} A'$ iff there is a function $f \leq_{\mathrm{T}} A$ such that f dominates all computable functions. And Robinson [1971a] proved a *Low Splitting Theorem* that if $\mathbf{c} < \mathbf{b}$ are c.e. degrees and $\mathbf{c}$ is low, then there are c.e. degrees $\mathbf{x}, \mathbf{y}$ such that $\mathbf{c} < \mathbf{x}, \mathbf{y} < \mathbf{b}$ and $\mathbf{x} \vee \mathbf{y} = \mathbf{b}$. In the proof of this theorem, a characterisation of low c.e. degrees was given. The lowness is necessary, because Lachlan [1975] proved a *Nonsplitting Theorem* that for some c.e. degrees $\mathbf{c} < \mathbf{b}$, $\mathbf{b}$ is

* The author was partially supported by an EPSRC Research Grant, "Turing Definability", No. GR/M 91419 (UK) and by NSF Grant No. 69973048 and NSF Major Grant No. 19931020 (P.R. CHINA), by NSFC grants No. 60310213, and No. 60325206.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 79–88, 2007.
© Springer-Verlag Berlin Heidelberg 2007

not splittable over $\mathbf{c}$. The strongest nonsplitting along this line has been given by Cooper and Li [2002]: there exists a low_2 c.e. degree above which $\mathbf{0}'$ is not splittable.

Extending both the Sacks Jump Theorem and the Sacks Density Theorem, Robinson [1971b] proved an *Interpolation Theorem*: given c.e. degrees $\mathbf{d} < \mathbf{c}$ and a degree $\mathbf{s}$ c.e. in $\mathbf{c}$ with $\mathbf{d}' \leq \mathbf{s}$, there is a c.e. degree $\mathbf{a}$ such that $\mathbf{d} < \mathbf{a} < \mathbf{c}$ and $\mathbf{a}' = \mathbf{s}$. Using this theorem, we can transfer some results from lower levels to higher levels of the high/low hierarchy. For instance, every high c.e. degree bounds a properly high_n, and a properly low_n c.e. degree for each $n > 0$, so any ideal I of $\mathcal{E}$ contains an element of $\mathbf{H}_1$ will contain elements of $\mathbf{H}_{n+1} - \mathbf{H}_n$, $\mathbf{L}_{n+1} - \mathbf{L}_n$ for all $n > 0$. However the transfer procedure is constrained by the non-uniformity of the Robinson Interpolation Theorem.

Based on Martin's Characterisation of High Degrees, Cooper [1974a] proved that every high_1 c.e. degree bounds a minimal pair. And Lachlan [1979] showed that there exists a nonzero c.e. degree which bounds no minimal pair. And Cooper [1974b] and Yates proved a *Noncupping Theorem*: there exists nonzero c.e. degree $\mathbf{a}$ such that for any c.e. degree $\mathbf{x}$, $\mathbf{a} \vee \mathbf{x} = \mathbf{0}'$ iff $\mathbf{x} = \mathbf{0}'$. This result was further extended by *Harrington* [1976] *Noncupping Theorem*: for any high_1 c.e. degree $\mathbf{h}$, there exists a high_1 c.e. degree $\mathbf{a} \leq \mathbf{h}$ such that for any c.e. degree $\mathbf{x}$, if $\mathbf{h} \leq \mathbf{a} \vee \mathbf{x}$, then $\mathbf{h} \leq \mathbf{x}$. In contrast Harrington [1978] also proved a *Plus Cupping Theorem* that there exists c.e. degree $\mathbf{a} \neq \mathbf{0}$ such that for any c.e. degrees $\mathbf{x}, \mathbf{y}$, if $\mathbf{0} < \mathbf{x} \leq \mathbf{a} \leq \mathbf{y}$, then there is a c.e. degree $\mathbf{z} < \mathbf{y}$ such that $\mathbf{x} \vee \mathbf{z} = \mathbf{y}$. And remarkably, Nies, Shore and Slaman [1998] have shown that $\mathbf{H}_n$, $\mathbf{L}_{n+1}$ are definable in $\mathcal{E}$ for each $n > 0$.

A basic question about the high/low hierarchies is the following:

Question 1.1. (i) Are there any $m \neq n$ such that $\text{Th}(\mathbf{L}_m) = \text{Th}(\mathbf{L}_n)$?
(ii) Are there any $m \neq n$ such that $\text{Th}(\mathbf{H}_m) = \text{Th}(\mathbf{H}_n)$?

Since this paper was written in 2001, part (i) has been answered negatively. Jockusch, Li, and Yang [2004] proved a nice join theorem for the c.e. degrees: For any c.e. degree $\mathbf{x} \neq \mathbf{0}$, there is a c.e. degree $\mathbf{a}$ such that $(\mathbf{x} \vee \mathbf{a})' = \mathbf{0}'' = \mathbf{a}''$. Cholak, Groszek and Slaman [2001] showed that there is a nonzero c.e. degree $\mathbf{a}$ which joins to a low degree with any low c.e. degree. By combining the two theorems above, we have that for any $n > 1$, $\text{Th}(\mathbf{L}_1) \neq \text{Th}(\mathbf{L}_n)$. While the remaining case was resolved by Shore [2004] by using coding of arithmetics.

For part (ii) of question 1.1, we know nothing, although Cooper proved that every high c.e. degree bounds a minimal pair, and Downey, Lempp and Shore [1993] (and independently both Lerman and Kučera) could construct a high_2 c.e. degree which bounds no minimal pair.

In this paper, we show that

Theorem 1.2. There exists a high_2 c.e. degree $\mathbf{a}$ such that for any c.e. degrees $\mathbf{x}, \mathbf{y}$, if $\mathbf{0} < \mathbf{x} \leq \mathbf{a} \leq \mathbf{y}$, then there is a c.e. degree $\mathbf{z}$ such that $\mathbf{z} < \mathbf{y}$ and $\mathbf{x} \vee \mathbf{z} = \mathbf{y}$.

Then we have:

Theorem 1.3. For each $n > 1$, $\mathrm{Th}(\mathbf{H}_1) \neq \mathrm{Th}(\mathbf{H}_n)$.

Proof. Let P be the following

$$\forall \mathbf{x} \exists \mathbf{a} \leq \mathbf{x} \forall \mathbf{y} (\mathbf{a} \vee \mathbf{y} = \mathbf{0}' \leftrightarrow \mathbf{y} = \mathbf{0}').$$

By Harrington's Noncupping Theorem, P holds for $\mathbf{H}_1$. Note that for any incomplete c.e. degree $\mathbf{a}$, there is an incomplete high c.e. degree $\mathbf{h} \geq \mathbf{a}$. So by theorem 1.2, for each $n > 1$, P fails to hold for $\mathbf{H}_n$.

Theorem 1.3 follows. □

This gives a partial solution to question 1.1 (ii), while the remaining case of it is still an intriguing open question.

We now outline the proof of theorem 1.2. In section 2, we formulate the conditions of the theorem by requirements, and describe the strategies to satisfy the requirements; in section 3, we arrange all strategies on nodes of a tree, the *priority tree* T and analyse the consistency of the strategies.

Our notation and terminology are standard and generally follow Soare [1987].

2 Requirements and Strategies

In this section, we formulate the conditions of theorem 1.2 by requirements, and describe the strategies to satisfy the requirements.

The Requirements. To prove theorem 1.2, we construct a c.e. set A, a Turing functional Γ to satisfy the following properties and requirements,

(1) For any x, y, z, $\Gamma(A; x, y, z)$ is defined.
(2) For any x, y, $\lim_z \Gamma(A; x, y, z)$ exists.
(3) For any x, $\lim_y \lim_z \Gamma(A; x, y, z)$ exists.
$\mathcal{P}_x$: $\emptyset'''(x) = \lim_y \lim_z \Gamma(A; x, y, z)$
$\mathcal{R}_e$: $W_e = \Phi_e(A) \rightarrow (\exists X_e, \Omega_e)[X_e \leq_{\mathrm{T}} V_e \oplus A \,\&\, V_e \oplus A = \Omega_e(W_e, X_e) \,\&\, (\text{a.e.}i)\mathcal{S}_{e,i}]$
$\mathcal{S}_{e,i}$: $[W_e = \Phi_e(A) \;\&\; A = \Psi_i(X_e)] \rightarrow W_e \leq_{\mathrm{T}} \emptyset$

where $x, y, z, e, i \in \omega$, $\{(W_e, \Phi_e, V_e) \mid e \in \omega\}$ is an effective enumeration of all triples (W, Φ, V) of c.e. sets W, V and of Turing functionals Φ, $\{\Psi_i \mid i \in \omega\}$ is an effective enumeration of all Turing functionals Ψ, X_e is a c.e. set built by us, Ω_e is a Turing functional built by us for each $e \in \omega$.

Clearly meeting the requirements is sufficient to prove the theorem. We assume that the use function ϕ of a given Turing functional Φ is increasing in arguments, nondecreasing in stages. We now look at the strategies to satisfy the requirements.

A $\mathcal{P}$-Strategy. Since $\emptyset''' \in \Sigma_3$, we can choose a c.e. set J such that for all x, both (i) and (ii) below hold,

(i) $x \in \emptyset'''$ iff $(\text{a.e.}y)[J^{[\langle y,x\rangle]} = \omega^{[\langle y,x\rangle]}]$.
(ii) $x \notin \emptyset'''$ iff $(\forall y)[J^{[\langle y,x\rangle]} =^* \emptyset]$.

To satisfy $\mathcal{P}_x$, we introduce infinitely many subrequirements $\mathcal{Q}_{x,y}$ for all $y \in \omega$. $\mathcal{Q}$-strategies will define and rectify the Turing functional Γ. Before describing the $\mathcal{Q}$-strategies, we look at some properties of Γ.

Γ-Rules. We ensure that the Turing functional Γ will satisfy the following properties, which are called *Γ-rules*.

(i) Whenever we define $\Gamma(A; x, y, z)$, we locate it at a node ξ say.
Let $\Gamma(A; x, y, z)[s]$ be located at ξ.
(ii) $\gamma(x, y, z)[s] \downarrow \neq \gamma(x, y, z)[s+1]$ iff $\gamma(x, y, z)[s]$ is enumerated into $A_{s+1} - A_s$ iff there is a strategy $\xi' <_{\mathrm{L}} \xi$ which is visited at stage $s + 1$.

Therefore for all x, y, z, the permanent computation $\Gamma(A; x, y, z)$ is the computation which is located at a node, ξ say, at a stage, s say, such that there is no $\alpha <_{\mathrm{L}} \xi$ which can be visited at any stage $v > s$.

A $\mathcal{Q}$-Strategy. Given a $\mathcal{Q}_{x,y}$-strategy σ, we use J^σ to denote the set $J^{[\langle y,x\rangle]}$ which is measured by σ. We say that s is *σ-expansionary*, if $J^\sigma[v] \subset J^\sigma[s]$ for all $v < s$ at which some $\alpha \supseteq \sigma$ is visited. Then σ will proceed as follows.

1. If s is σ-expansionary, then
   – let $\langle y', z'\rangle$ be the least pair $\langle m, n\rangle$ such that $m \geq y$ and $\Gamma(A; x, m, n)$ is not defined,
   – define $\Gamma(A; x, y', z') \downarrow = 1$ with $\gamma(x, y', z')$ fresh in the sense that it is the least natural number greater than any number mentioned so far, and
   – locate $\Gamma(A; x, y', z')$ at $\sigma^\frown\langle 0\rangle$.
2. Otherwise, then
   – let z' be the least n such that $\Gamma(A; x, y, n) \uparrow$,
   – define $\Gamma(A; x, y, z') \downarrow = 0$ with $\gamma(x, y, z')$ fresh, and locate it at $\sigma^\frown\langle 1\rangle$.

So the *possible outcomes of* σ are $0 <_{\mathrm{L}} 1$ to denote infinite and finite actions respectively. By the strategy, if there are infinitely many σ-expansionary stages, then for almost every pair $\langle y', z'\rangle$ with $y' \geq y$, $\Gamma(A; x, y', z') \downarrow = 1$ is defined and located at $\sigma^\frown\langle 0\rangle$. In this case, $\lim_y \lim_z \Gamma(A; x, y, z) \downarrow = 1$, and by the choice of J, $x \in \emptyset'''$. $\mathcal{P}_x$ is satisfied. Otherwise, then by the $\mathcal{Q}_{x,y}$-strategy σ, we have that for almost every z, $\Gamma(A; x, y, z) \downarrow = 0$ is defined and located at $\sigma^\frown\langle 1\rangle$, so that $\lim_z \Gamma(A; x, y, z) \downarrow = 0$, giving $\lim_y \lim_z \Gamma(A; x, y, z) \downarrow = 0$. Therefore in any case, $\mathcal{P}_x$ is satisfied.

An $\mathcal{R}$-Strategy. First we define the notion of *α-believable computation.* Given a node α, we say that $\Phi(A; w) \downarrow = v$ is *α-believable*, if for any x, y, z, if $\Gamma(A; x, y, z)$ is defined and located at some node ξ with $\alpha <_{\mathrm{L}} \xi$, then $\phi(w) < \gamma(x, y, z)$.

An $\mathcal{R}$-strategy, α say, will satisfy an $\mathcal{R}$-requirement, $\mathcal{R}$ say (we drop the index), we define the *length function of agreement* $l(\alpha) = (W, \Phi(A))$ as usual, of course α uses only α-believable computations. We say that s is *α-expansionary*, if $l(\alpha)[s] > l(\alpha)[v]$ for all $v < s$ at which α is visited.

If there are only finitely many α-expansionary stages, then either $l(W, \Phi(A))[s]$ is bounded over the construction, or there is a fixed w say such that there are infinitely many stages at which α is visited and at which $\Phi(A; w) \downarrow$ is not α-believable, and by the Γ-rules, at which some elements $\leq \phi(w)$ are enumerated into A. In this case, $\phi(w)[s]$ will be unbounded over the construction. Therefore in either case, $W \neq \Phi(A)$, $\mathcal{R}$ is satisfied.

Suppose that there are infinitely many α-expansionary stages. Then we will build a c.e. set X, two Turing functionals Ξ and Ω such that both (a) and (b) below hold.

(a) $X = \Xi(V, A)$,
(b) $V \oplus A = \Omega(W, X)$.

For Ξ, whenever we define $\Xi(V, A; x)$, we define $\Xi(V, A; x) \downarrow = X(x)$ with $\xi(x) = x$. And once $V \upharpoonright (x+1)$ or $A \upharpoonright (x+1)$ changes, we set $\Xi(V, A; x)$ to be undefined. We ensure that an element x is enumerated into X, only if $\Xi(V, A; x)$ is currently undefined. So if $\Xi(V, A)$ is total, then $\Xi(V, A) = X$.

For Ω, whenever we define $\Omega(W, X; x)$, we define $\Omega(W, X; x) \downarrow = (V \oplus A)(x)$ with $\omega(x)$ fresh. And if $\Omega(W, X; x) \downarrow \neq (V \oplus A)(x)$, we enumerate $\omega(x)$ into X. This ensures that if $\Omega(W, X)$ is total, then $\Omega(W, X) = V \oplus A$.

Of course we have to ensure that W-change will never make $\Omega(W, X)$ partial, in fact, we ensure that Ω and Ξ will have the following properties,

(i) if $\Omega(W, X)$ is total, then $\Xi(V, A)$ is total, and
(ii) if $\Omega(W, X)$ is partial, then either $\Phi(A)$ is partial or $W \leq_{\mathrm{T}} \emptyset$.

Finally we define the *possible outcomes of an $\mathcal{R}$-strategy* to be $0 <_{\mathrm{L}} 1$ to denote infinite and finite actions respectively.

An $\mathcal{S}$-Module. An $\mathcal{S}_{e,i}$-module assumes that an $\mathcal{R}_e$-strategy α, say, is building a Turing functional Ω. It will try to satisfy its $\mathcal{S}$-requirement, $\mathcal{S}_{e,i}$. For simplicity, we drop the indices e, i in the following discussion.

Suppose that β is an $\mathcal{S}$-module. Let $\alpha^\frown\langle 0\rangle \subseteq \beta$. Then β will have to deal with the injury from the building of $\Omega(W, X)$. It will work with a fixed *threshold* k say. Whenever we define the threshold, we define it as fresh. If $(V \oplus A) \upharpoonright k$ changes, then any previous action of β is cancelled but keep the threshold k unchanged, in which case, we say that β is *reset*. Clearly β is reset only finitely many times. Then the $\mathcal{S}$-module β will build a Turing function f and will proceed as follows.

1. Define an *agitator* a to be fresh.
   [Note that if both a and $\omega(k)$ are defined, then $a < \omega(k)$, where k is the threshold of β.]
2. (Create a Link (α, β)) Wait for a stage, v say, at which
   (2a) $\Psi(X; a) \downarrow = 0 = A(a)$,
   (2b) $W \upharpoonright (\omega(k)+1) = \Phi(A) \upharpoonright (\omega(k)+1)$ via β-believable computations.
   Then:
   – define $r = -1$ to be the A-restraint of β,
   – enumerate a into A, and
   – create a *link* (α, β).

3. (Travel the Link (α, β)) Wait for the next α-expansionary stage at which $W \upharpoonright (\omega(k)+1) = \Phi(A) \upharpoonright (\omega(k)+1)$ via α-believable computations. Then travel the link (α, β) through one of the following cases.
Case 3a. $W_v \upharpoonright (\omega(k)+1) \neq W \upharpoonright (\omega(k)+1)$. Then
– set $\omega(k)$ to be undefined,
– remove the link (α, β) and stop.
[Now we have created and preserved an inequality $\Psi(X; a) \downarrow = 0 \neq 1 = A(a)$. $\mathcal{S}$ is satisfied.]
Case 3b. Otherwise, and $\Phi(A) \upharpoonright (\omega(k)+1)$ are β-believable. Then:
– remove the link (α, β),
– for each $x \leq \omega(k)$, if $f(x) \uparrow$, then define $f(x) = W(x)$,
– enumerate $\omega(k)$ into X,
– define an agitator a as fresh, and
– define $r = \phi(\omega(k))$ to be the A-restraint of β.
[The enumeration of a into A at stage v created a $(V \oplus A) \upharpoonright \omega(k)$-permission via Ω, which has been kept by the link (α, β). So we can enumerate $\omega(k)$ into X at this stage.]
Case 3c. Otherwise, then do nothing.

The Possible Outcomes
The *possible outcomes of the $\mathcal{S}$-module* are as follows.

g: Case 3b occurs infinitely many times.

In this case, $\omega(k)[s]$ will be unbounded, so that f is defined to be a computable function. We prove that for every x, if $f(x) \downarrow = y$, then $W(x) = y$. Given x, let s_1 be the stage at which $f(x)$ is defined for the first time, then $f(x) = W_{s_1}(x)$. Let v_1 be minimal greater than s_1 at which step 2 of the module occurs. By the A-restraint $r[s] = r[s_1]$ for all $s \in [s_1, v_1)$, $f(x) = W_{v_1}(x)$. Let s_2 be the least stage greater than v_1 at which case 3b of β occurs. By the choice of s_2, $W_{s_2}(x) = f(x)$. Suppose by induction that $s_n \geq s_2$, that case 3b of β occurs at stage s_n, and that $W_{s_n}(x) = f(x)$. Let v_n be the least stage $> s_n$ at which step 2 of β occurs. Then for each $s \in [s_n, v_n)$, $r[s] = r[s_n]$, which ensures that $W_{v_n}(x) = f(x)$. Let s_{n+1} be the least stage greater than v_n at which case 3b of β occurs. By the choice of s_{n+1}, we have that $W_{s_{n+1}}(x) = f(x)$. It follows that there are infinitely many stages at which $W(x) = f(x)$, giving $W(x) = f(x)$. Since x is arbitarily given we have that $f = W$. $\mathcal{R}$ is satisfied.

u: Otherwise, and case 3c occurs infinitely many times.

In this case, there is a link (α, β) which was created and which will neither be cancelled nor be removed, and which is called *a permanent link*. We note that $\lim_s \omega(k)[s] \downarrow = v < \omega$ for some v, and that there are infinitely many stages at which $\Phi(A; v)$ is not β-believable, and at which some elements $\gamma(x, y, z) \leq \phi(v)$ are enumerated into A, by the Γ-rules. Therefore $\Phi(A)$ is partial. Both $\mathcal{R}$ and $\mathcal{S}$ are satisfied.

However every ξ strictly between α and β is *covered* by β in the sense that ξ is visited only finitely many times. The solution is the following observation:

(1) If ξ is either an $\mathcal{R}$- or a $\mathcal{P}$-strategy, then ξ's requirement has lower priority than that of α, we can introduce a *backup strategy* below $\beta^\frown\langle u\rangle$ for the requirement of ξ. Therefore the injury of ξ from β is harmless.

(2) If ξ is a $\mathcal{Q}$- or an $\mathcal{S}$-strategy which works on a subrequirement whose global requirement has lower priority than that of α, then we can neglect this ξ, because, for a $\mathcal{P}$-, or an $\mathcal{R}$-requirement, we are allowed to give up finitely many subrequirements $\mathcal{Q}$ or $\mathcal{S}$.

(3) Otherwise and $\xi = \sigma$ is a $\mathcal{Q}$-strategy. Then we have that $\sigma^\frown\langle 1\rangle \subseteq \beta$ holds. Now in case 3c of β, we may allow σ to act if the current stage is σ-expansionary.

(4) Otherwise and $\xi = \beta'$ is an $\mathcal{S}$-strategy. Then $\beta'^\frown\langle w\rangle \subseteq \beta$ holds. In this case, whenever case 3c of β occurs, we may allow β' to act, if β' is ready to create a link (or to open an A-gap), in the sense that step 2 of strategy β' appears.

w: Otherwise. Now it is easy to see that one of the following cases occurs.

Case 1. Case 3a of β occurs. Then $\Psi(X;a)\downarrow = 0 \neq 1 = A(a)$ is created and preserved for some fixed a.

Case 2. Otherwise, and (2a) in step 2 fails to hold infinitely often. This means that $\Psi(X;a) \neq 0 = A(a)$.

Case 3. Otherwise, then there are infinitely many stages at which if $W \upharpoonright (\omega(k)+1) = \Phi(A) \upharpoonright (\omega(k)+1)$, then $\Phi(A;\omega(k))$ is not β-believable, in which case, by the Γ-rules, some elements $\gamma(x,y,z) \leq \phi(\omega(k))$ are enumerated into A infinitely many times. We have that $W \neq \Phi(A)$.

So in any case, we have that either $\Psi(X) \neq A$ or $W \neq \Phi(A)$, $\mathcal{S}$ is satisfied.

We define the priority ordering of the possible outcomes of β by $g <_{\mathrm{L}} u <_{\mathrm{L}} w$.

And a general $\mathcal{S}$-strategy is just an modification of the $\mathcal{S}$-module according to the observations in (1)–(4) above.

3 The Priority Tree T

In this section, we build the priority tree T and analyse some basic properties about the priority tree. First we define *the priority ranking of the requirements*.

Definition 3.1. Given a sequence $\mathcal{L} = (X_0, X_1, \cdots, X_n)$ of requirements, let m be the greatest $j \leq n$ such that X_j is a $\mathcal{P}$- or an $\mathcal{R}$-requirement. Then:

(i) We say that $\mathcal{P}_x$ *is complete in* $\mathcal{L}$ if there is a k such that $m < k \leq n$ and $X_k = \mathcal{Q}_{x,y}$ for some $y \in \omega$.

(ii) We say that $\mathcal{R}_e$ *is complete in* $\mathcal{L}$, if there is a k such that $m < k \leq n$ and $X_k = \mathcal{S}_{e,i}$ for some $i \in \omega$.

(iii) We say that $\mathcal{L} = (X_0, X_1, \cdots, X_n)$ *is complete*, if for every j, if X_j is a $\mathcal{P}$- or an $\mathcal{R}$-requirement, then X_j is complete in $\mathcal{L}$.

We now define the priority ranking $\mathcal{L}$ of the requirements inductively.

Definition 3.2. (i) Define the priority ranking of the $\mathcal{P}$- and $\mathcal{R}$-requirements such that $\mathcal{P}_e < \mathcal{R}_e < \mathcal{P}_{e+1} < \mathcal{R}_{e+1}$ holds for each $e \in \omega$.

(ii) Define $\mathcal{L} = \emptyset$.

Suppose by induction that $\mathcal{L} = (X_0, X_1, \cdots, X_n)$ has been defined.

(iii) If $\mathcal{L}$ is not complete, then let j be the least k such that X_k is a $\mathcal{P}$- or an $\mathcal{R}$-requirement which is not complete in $\mathcal{L}$. If $X_j = \mathcal{P}_x$ for some x, then let y be minimal such that $\mathcal{Q}_{x,y}$ is not in $\mathcal{L}$, and set $X_{n+1} = \mathcal{Q}_{x,y}$. If $X_j = \mathcal{R}_e$ for some e, then let i be the least i' such that $\mathcal{S}_{e,i'}$ is not in $\mathcal{L}$ and set $X_{n+1} = \mathcal{S}_{e,i}$.

Set $\mathcal{L} = (X_0, X_1, \cdots, X_n, X_{n+1})$ and go back to (iii).

(iv) Otherwise, then let X_{n+1} be the least $\mathcal{P}$- or $\mathcal{R}$-requirement as defined in (i) which is not in $\mathcal{L}$, set $\mathcal{L} = (X_0, X_1, \cdots, X_n, X_{n+1})$ and go back to (iii).

(v) Suppose that $\mathcal{L} = (X_0, X_1, \cdots)$. Then we define $X_i < X_j$ iff $i < j$, giving the *priority ranking of the requirements*.

Proposition 3.3. Suppose that $\mathcal{L}$ is the priority ranking of the requirements defined in definition 3.2. Then for all $e \in \omega$, we have:

(i) $\mathcal{P}_e < \mathcal{R}_e < \mathcal{P}_{e+1} < \mathcal{R}_{e+1}$,
(ii) $\mathcal{P}_e < \mathcal{Q}_{e,i} < \mathcal{Q}_{e,i+1}$ for all $i \in \omega$, and
(iii) $\mathcal{R}_e < \mathcal{S}_{e,i} < \mathcal{S}_{e,i+1}$ for all $i \in \omega$.

Proof. This is immediate from definitions 3.1 and 3.2. □

Definition 3.4. We define *the possible outcomes of a strategy* as the same as that in section 2.

Definition 3.5. Given a node ξ:

(i) $\mathcal{P}_x$ *is satisfied at* ξ, if there are $\mathcal{P}_x$-strategy τ and $\mathcal{Q}_{x,y}$-strategy σ for some y such that
(a) $\tau \subset \tau^\frown\langle 0\rangle \subseteq \sigma \subset \sigma^\frown\langle 0\rangle \subseteq \xi$,
(b) there is no $\mathcal{S}_{e,i}$-strategy β such that $\sigma^\frown\langle 0\rangle \subseteq \beta \subset \beta^\frown\langle u\rangle \subseteq \xi$ for any $e < x$.

(ii) $\mathcal{P}_x$ *is active at* ξ, if $\mathcal{P}_x$ is not satisfied at ξ and there is a $\mathcal{P}_x$-strategy τ such that $\tau \subset \xi$ and there is no $\mathcal{S}_{e,i}$-strategy β such that $\tau \subset \tau^\frown\langle 0\rangle \subseteq \beta \subset \beta^\frown\langle u\rangle \subseteq \xi$ for any $e < x$.

(iii) $\mathcal{R}_e$ *is satisfied at* ξ, if either (a) or (b) below holds,
(a) there is an $\mathcal{R}_e$-strategy α such that $\alpha^\frown\langle 1\rangle \subseteq \xi$ and there is no $\mathcal{S}_{e',i'}$-strategy β such that $\alpha^\frown\langle 1\rangle \subseteq \beta \subset \beta^\frown\langle u\rangle \subseteq \xi$ for any $e' < e$.
(b) there is an $\mathcal{S}_{e,i}$-strategy β such that $\beta^\frown\langle a\rangle \subseteq \xi$ for some $a \in \{g, u\}$ and such that there is no $\mathcal{S}_{e'i'}$-strategy β' with $\beta^\frown\langle a\rangle \subseteq \beta' \subset \beta'^\frown\langle u\rangle \subseteq \xi$ for any $e' < e$.

(iv) We say that $\mathcal{R}_e$ *is active at* ξ, if $\mathcal{R}_e$ is not satisfied at ξ, and there is an $\mathcal{R}_e$-strategy α such that
(a) $\alpha^\frown\langle 0\rangle \subseteq \xi$,
(b) there is no $\mathcal{Q}_{x,y}$-strategy σ such that $\alpha^\frown\langle 0\rangle \subseteq \sigma \subset \sigma^\frown\langle 0\rangle \subseteq \xi$ for any $x \leq e$, and
(c) there is no $\mathcal{S}_{e',i'}$-strategy β such that $\alpha^\frown\langle 0\rangle \subseteq \beta \subset \beta^\frown\langle b\rangle \subseteq \xi$ for any $b \in \{g, u\}$ and any $e' < e$.

(v) We say that $\mathcal{Q}_{x,y}$ *is satisfied at* ξ if there is a $\mathcal{Q}_{x,y}$-strategy $\sigma \subset \xi$.

(vi) We say that $\mathcal{S}_{e,i}$ *is satisfied at* ξ if there is an $\mathcal{S}_{e,i}$-strategy $\beta \subset \xi$.

We now define the priority tree T.

Definition 3.6. Let $\mathcal{L}$ be the priority ranking of the requirements defined in definition 3.2. Then:

(i) Define the root node $\emptyset$ to be the strategy for the first requirement in $\mathcal{L}$, which is actually $\mathcal{P}_0$.

(ii) The immediate successors of a node are the possible outcomes of the corresponding strategy.

(iii) A node ξ will work on the least element in $\mathcal{L}$ which is not satisfied, and not active at ξ.

As usual, we have the following:

Proposition 3.7. (Finite Injury Along Any Path Proposition) Let f be an infinite path through T. Then for every $\mathcal{P}$- or $\mathcal{R}$-requirement X, there is a fixed n_0 such that either X is satisfied at $f \upharpoonright n$ for all $n \geq n_0$, or X is active at $f \upharpoonright n$ for all $n \geq n_0$.

Proof. By induction on the priority ranking of the requirements. □

Given an $\mathcal{S}_{e,i}$-strategy, we define *the top of* β to be the longest $\mathcal{R}_e$-strategy α such that $\alpha^\frown\langle 0\rangle \subseteq \beta$, denoted by $\mathrm{top}(\beta)$.

We also need some more properties about the structure of the priority tree T.

Proposition 3.8. Let $\beta \in T$ be an $\mathcal{S}_{e,i}$-strategy, and $\alpha = \mathrm{top}(\beta)$. Then:

(i) If σ is a $\mathcal{Q}_{x,y}$-strategy and $\alpha \subset \alpha^\frown\langle 0\rangle \subseteq \sigma \subset \sigma^\frown\langle 0\rangle \subseteq \beta$, then $x > e$.

(ii) If β' is an $\mathcal{S}_{e',i'}$-strategy such that $\alpha \subset \alpha^\frown\langle 0\rangle \subseteq \beta' \subset \beta'^\frown\langle a\rangle \subseteq \beta$ for some $a \in \{g, u\}$, then for $\alpha' = \mathrm{top}(\beta')$, $\alpha \subset \alpha' \subset \beta' \subset \beta$, and $e' > e$.

(iii) If α' is an $\mathcal{R}_{e'}$-strategy such that $\alpha \subset \alpha' \subset \beta$, then $e' > e$.

(iv) If τ is a $\mathcal{P}_x$-strategy such that $\alpha \subset \tau \subset \beta$, then $x > e$.

Proof. It is straightforward from definitions 3.5 and 3.6. □

The full construction and its verification is a $\mathbf{0}'''$-priority tree argument which will be given in the full version of the paper.

References

1. S. B. Cooper [1974a], On a theorem of C. E. M. Yates (handwritten notes).
2. S. B. Cooper [1974b], Minimal pairs and high recursively enumerable degrees, J. Symbolic Logic 39 (1974), 655–660.
3. S. B. Cooper and Angsheng Li, Splitting and nonsplitting, II: A low_2 c.e. degree above which $\mathbf{0}'$ is not splittable, the Journal of Symbolic Logic, Vol. 67, No. 4, Dec. 2002.
4. R. G. Downey, S. Lempp and R. A. Shore [1993], Highness and bounding minimal pairs, Math. Logic Quarterly, Vol. 39, 475–491.
5. L. Harrington [1976], On Cooper's proof of a theorem of Yates, Parts I and II (handwritten notes).
6. Carl G. Jockusch, JR., Angsheng Li, and Yue Yang, A join theorem for the computably enumerable degrees, Transactions of the American Mathematical Society, Vol. 356, No. 7, pages 2557 –2568.

7. L.Harrington [1978], Plus cupping in the recursively enumerable degrees, (handwritten notes).
8. A. H. Lachlan [1965], On a problem of G. E. Sacks, Proc. Amer. Math. Soc. 16 (1965), 972–979.
9. A. H. Lachlan [1975], A recursively enumerable degree which will not split over all lesser ones, Ann. Math. Logic 9 (1975), 307–365.
10. A. H. Lachlan [1979], Bounding minimal pairs, J. Symbolic Logic, 44 (1979), 626–642.
11. D. A. Martin [1966a], On a question of G. E. Sacks, J. Symbolic Logic 31 (1966), 66–69.
12. D. A. Martin [1966b], Classes of recursively enumerable sets and degrees of unsolvability, 2, Math. Logik Grundlag Math. 12 (1966), 295–310.
13. D. Miller [1981], High recursively enumerable degrees and the anti-cupping property, in M. Lerman, J. H. Schmerl and R. I. Soare (editors), Lecture Notes in Mathematics No. 859, Springer-Verlag, Berlin, Heidelberg, Tokyo, New York, 1981.
14. A. Nies, R. A. Shore and T. A. Slaman [1998], Interpretability and definability in the recursively enumerable degrees, Proc. London Math. Soc. (3), 77 (2): 241–291, 1998.
15. R. W. Robinson [1971a], Interpolation and embedding in the recursively enumerable degrees, Ann. of Math. (2) 93 (1971), 285–314.
16. R. W. Robinson [1971b], Jump restricted interpolation in the recursively enumerable degrees, Ann. of Math. (2) 93 (1971), 586–596.
17. G. E. Sacks [1963], Recursive enumerability and the jump operator, Tran. Amer. Math. Soc. 108 (1963), 223–239.
18. G. E. Sacks [1964], The recursively enumerable degrees are dense, Ann. of Math. (2) 80 (1964), 300–312.
19. G. E. Sacks [1967], On a theorem of Lachlan and Martin, Proc. Amer. Math. Soc. 18 (1967), 140–141.
20. R. A. Shore, The low_m and low_n r.e. degrees are not elementarily equivalent, Science in China, Series A, 2004.
21. R. A. Shore and T. A. Slaman [1990], Working below a low_2 recursively enumerable degree, Archive for Math. Logic, 29 201–211.
22. R. A. Shore and T. A. Slaman [1993], Working below a high recursively enumerable degree, J. Symbolic Logic, Vol. 58 No. 3, Sept. 1993, 824–859.
23. R. I. Soare [1987], Recursively Enumerable Sets and Degrees, Springer-Verlag Berlin, Heidelberg New York London Pairs Tokyo, 1987.

The proof of this paper was written in 2001, but has not appeared in the literature. 1991 Mathematics Subject Classification. Primary 03D25, 03D30; Secondary 03D35.

Working with the LR Degrees*

George Barmpalias[1], Andrew E.M. Lewis[2], and Mariya Soskova[1]

[1] School of Mathematics, University of Leeds, Leeds, LS2 9JT, U.K.
[2] Dipartimento di Scienze Matematiche ed Informatiche Via del Capitano 15, 53100 Siena

Abstract. We say that $A \leq_{LR} B$ if every B-random number is A-random. Intuitively this means that if oracle A can identify some patterns on some real γ, oracle B can also find patterns on γ. In other words, B is at least as good as A for this purpose. We propose a methodology for studying the LR degrees and present a number of recent results of ours, including sketches of their proofs.

1 Introduction

The present paper is partly a short version of a longer draft [1] with the full proofs of the results presented here, but it also contains additional very recent material which does not appear in [1]. One of the goals of this work is to present a uniform approach to studying the LR degrees, both globally and locally. So far the known results about this degree structure have mostly been scattered and in papers dealing with a wider range of themes in algorithmic randomness (see for example [11]). An exception is Simpson's recent paper [16] which deals with themes like almost everywhere domination which are very closely related to the LR degrees.

Also, a number of results in this area have been proved via a mix of frameworks like martingales, prefix-free complexity and Martin-Löf tests, with more than one framework sometimes appearing in the same proof (see [11,12]). In contrast, we present proofs of new and old results using only the Martin-Löf approach, i.e. Σ^0_1 classes and (in the relativised case) c.e. operators. We work in the Cantor space 2^ω with the usual topology generated by the basic open intervals $[\sigma] = \{\beta \mid \beta \in 2^\omega \ \wedge \ \sigma \subseteq \beta\}$ (where σ is a finite binary string and $\sigma \subseteq \beta$ denotes that σ is a prefix of β) and the Lebesgue measure generated by $\mu([\sigma]) = 2^{-|\sigma|}$.

We systematically confuse sets of finite strings U with the class of reals which extend some string in U. Thus

* Barmpalias was supported by EPSRC Research Grant No. EP/C001389/1 and would also like to thank Steve Simpson for a number of stimulating discussions, as well as Doug Cenzer for his hospitality during a visit to the University of Florida, September-November 2006. Lewis was supported by Marie-Curie Fellowship No. MEIF-CT-2005-023657. Soskova was supported by the Marie Curie Early Training grant MATHLOGAPS (MEST-CT-2004-504029). All authors were partially supported by the NSFC Grand International Joint Project, No. 60310213, *New Directions in the Theory and Applications of Models of Computation*.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 89–99, 2007.
© Springer-Verlag Berlin Heidelberg 2007

- we write $\mu(U)$ for the measure of the corresponding class of reals
- all subset relations $U \subset V$ where U, V are sets of strings actually refer to the corresponding classes of reals
- Boolean operations on sets of strings actually refer to the same operations on the corresponding classes of reals.

In section 2 we review the basic definition of an oracle Martin-Löf test and make the simple observation that there exists a universal test with certain nice properties that will later be useful. It is worth noting that not all universal (even unrelativised) Martin-Löf tests have the same properties and for some arguments it is convenient to assume that we hold a special test. This is not new—see for example [9] where a property of the test derived by its construction (and not the general definition) is used to show that random sets are effectively immune. In section 3 we give the definition of $\leq_{LR}$ and the induced degree structure and mention some known properties. In section 4 we show that there is a continuum of reals which are LR-reducible to the halting problem and then extend this argument to show that the same is true of any α which is not GL_2. We also give a method for diagonalization in the LR degrees.

In section 5 we study the structure of the computably enumerable LR degrees. The main goal here is to show how techniques from the theory of the c.e. Turing degrees can be transferred to the c.e. LR degrees. We deal with two fundamental techniques: Sacks coding and Sacks restraints. First we show that if A has intermediate c.e. Turing degree then the lower cone of c.e. LR degrees below it properly extends the corresponding cone of c.e. Turing degrees. The second example demonstrates the use of Sacks restraints in the LR context and is a splitting theorem for the LR degrees: every c.e. set can be split into two c.e. sets of incomparable LR degree. Also some results are given concerning further connections between the LR and the Turing degrees.

Most of the proofs are either omitted or given as sketches. The exceptions are the proofs of theorems 7 and 9 which are given in full. For the full proofs we refer the reader to the draft [1] which is available online. Theorems 9 and 14 do not appear in [1].

2 Oracle Martin-Löf Tests

An oracle Martin-Löf test (U_e) is a uniform sequence of oracle machines which output finite binary strings such that if U_e^β denotes the range of the e-th machine with oracle $\beta \in 2^\omega$ then for all $\beta \in 2^\omega$ and $e \in \mathbb{N}$ we have that $\mu(U_e^\beta) < 2^{-(e+1)}$ and $U_e^\beta \supseteq U_{e+1}^\beta$. A real α is called β-random if for every oracle Martin-Löf test (U_e) we have $\alpha \notin \cap_e U_e^\beta$. A universal oracle Martin-Löf test is an oracle Martin-Löf test (U_e) such that for every $\alpha, \beta \in 2^\omega$, α is β-random iff $\alpha \notin \cap_e U_e^\beta$. The following theorem concerns oracle-enumerations of random sets.

Theorem 1. *For every $n \geq 1$ there exist sets which are n-random and which are properly n-c.e. in $\emptyset^{(n)}$.*

Given any oracle Martin-Löf test (U_e), each U_e can be thought of as a c.e. set of axioms $\langle \tau, \sigma \rangle$. If $\beta \in 2^\omega$ then $U_e^\beta = \{\sigma \mid \exists \tau (\tau \subset \beta \ \wedge \ \langle \tau, \sigma \rangle \in U_e)\}$ and for $\rho \in 2^{<\omega}$ we define $U_e^\rho = \{\sigma \mid \exists \tau (\tau \subseteq \rho \ \wedge \ \langle \tau, \sigma \rangle \in U_e)\}$. There is an analogy between oracle Martin-Löf tests as defined above and Lachlan functionals i.e. Turing functionals viewed as c.e. sets of axioms. This analogy will be exploited in a number of constructions below, especially in the constructions of c.e. LR degrees. The following lemma is easily proved and provides a universal oracle Martin-Löf test with properties which will later be useful.

Lemma 1. *There is an oracle Martin-Löf test (U_e) such that*

- *For every oracle Martin-Löf test (V_e), uniformly on its c.e. index we can compute $k \in \mathbb{N}$ such that for every real β and all e, $V_{e+k}^\beta \subseteq U_e^\beta$.*
- *If $\langle \tau_1, \sigma_1 \rangle, \langle \tau_2, \sigma_2 \rangle \in U_e$ and $\tau_1 \subseteq \tau_2$ then $\sigma_1 | \sigma_2$.*
- *If $\langle \tau, \sigma \rangle \in U_e$ then $|\tau| = |\sigma|$ and $\langle \tau, \sigma \rangle \in U_e[|\tau|] - U_e[|\tau| - 1]$.*

From the properties of (U_e) as described in lemma 1 we get the following.

Corollary 1. *Let (U_e) be the universal oracle Martin-Löf test of lemma 1 and let U be any member of it. There is a computable function which, given any input $\langle \tau, \tau' \rangle$ such that $\tau \subseteq \tau'$, outputs the finite (clopen) set $U^{\tau'} - U^\tau$.*

Different Martin-Löf tests may have different properties and some are more useful than others. If we only require the first clause of theorem 1 we can achieve the stronger condition

> For every oracle Martin-Löf test (V_e), uniformly on its c.e. index we can compute $k \in \mathbb{N}$ such that $V_{e+k} \subseteq U_e$ (as sets of axioms) for all e. (1)

The following result demonstrates an application of property (1) of a universal Martin-Löf test (U_e).

Theorem 2. *If U is a member of an oracle Martin-Löf test satisfying property (1) and $T \in \Sigma_1^0$, $\mu(T) < 1$ then there are only finitely many $\beta \in 2^\omega$ such that $U^\beta \subseteq T$. Also, there are universal Martin-Löf tests which do not have this property.*

It is worth mentioning that there are tests which satisfy both the conditions of lemma 1 and the property of theorem 2. In fact, a standard construction of the oracle Martin-Löf test of theorem 2 gives a test with these properties. Note that if V is a member of an oracle Martin-Löf test and $T \in \Sigma_1^0$ then the class $\{\beta \mid V^\beta \subseteq T\}$ consists of the infinite paths through a $\mathbf{0}'$ computable tree. Since the paths through a $\mathbf{0}'$ computable tree with only finitely many infinite paths are Δ_2^0, by theorem 3 we get Nies' result [11] that all low for random sets are Δ_2^0.

3 *LR* Reducibility and Degrees

The LR reducibility was introduced in [13].

Definition 1. *[13] Let $A \leq_{LR} B$ if every B-random real is A-random. The induced degree structure is called the LR degrees.*

Intuitively this means that if oracle A can identify some patterns on some real γ, oracle B can also find patterns on γ. In other words, B is at least as good as A for this purpose. It is not hard to show (especially in view of theorem 3) that $\leq_{LR}$ is Σ^0_3 definable and this has been noticed by a number of authors. Being Σ^0_3 means that it has some things in common with $\leq_T$ (which is also Σ^0_3) and this can be seen more clearly in section 5 where techniques from the theory of c.e. Turing degrees are seen to be applicable in the c.e. LR degrees. For more examples of similar Σ^0_3 relations see [16]. We point out (after [13,16]) that a strict relativization of the notion of low for random [8] gives that A is low for random relative to B when $A \oplus B \leq_{LR} B$, which is different than $A \leq_{LR} B$. In particular, $\oplus$ does not define a least upper bound in the LR degrees and it is an open question as to whether any two degrees always have a least upper bound in this structure [13,16].

Theorem 3. *[7] For all $A, B \in 2^\omega$ the following are equivalent:*

- *$A \leq_{LR} B$*
- *For every $\Sigma^0_1(A)$ class T^A of measure < 1 there is a $\Sigma^0_1(B)$ class V^B such that $\mu(V^B) < 1$ and $T^A \subseteq V^B$.*
- *For some member U^A of a universal Martin-Löf test relative to A there is $V^B \in \Sigma^0_1(B)$ such that $\mu(V^B) < 1$ and $U^A \subseteq V^B$.*

The following result shows how two universal oracle Martin-Löf tests are related (or how 'similar' they are).

Theorem 4. *If (U_i) is an oracle Martin-Löf test, V is a member of a universal oracle Martin-Löf test and $\tau_0, \sigma_0 \in 2^{<\omega}$ such that $[\sigma_0] \not\subseteq \cap_{\gamma \supset \tau_0} V^\gamma$ then there exist $\tau, \sigma \in 2^{<\omega}$ and $m \in \mathbb{N}$ such that*

- *$\tau \supset \tau_0$ and $\sigma \supset \sigma_0$*
- *there is $\beta \supset \tau$ such that $[\sigma] \not\subseteq V^\beta$*
- *for all $\gamma \supset \tau$, $U^\gamma_m \cap [\sigma] \subseteq V^\gamma$.*

A natural question about reducibilities $\preceq$ on the reals is to determine the measure of upper and lower cones. For the Turing reducibility the lower cones are countable (hence they are null) and the non-trivial upper cones have measure 0 [14]. For $\leq_{LR}$ although lower cones are not always countable (see section 4) it is not difficult to show that they are null. Indeed, given A the A-random numbers have measure 1 and so it is enough to show that if β is A-random then $\beta \not\leq_{LR} A$. But this is obvious since β is not β-random.

Theorem 5. *For every A the set $\{\beta \mid \beta \leq_{LR} A\}$ has measure 0.*

For the upper cones it is tempting to think that a version of the majority vote technique which settled the question for $\leq_T$ (see [4] for an updated presentation of the argument) would work for $\leq_{LR}$ (especially if one thinks of randomness in terms of betting strategies). However Frank Stephan pointed out (in discussions

with the first author) that the answer is most easily given by an application of van Lambalgen's theorem (a simple theorem with many applications) which asserts that $A \oplus B$ *is random iff* A *is random and* B *is* A*-random.*

Theorem 6 (Frank Stephan). *If* A *is random then it is* B*-random for almost all* $B \in 2^\omega$*. Also, any non-trivial upper cone in the* LR *degrees has measure* 0*.*

4 Global Structure

In computability theory we are used to structures in which every degree has only countably many predecessors. Below we show that the LR degrees do not have this property[1] and that, in fact, whenever α is not GL_2 the degree of α has an uncountable number of predecessors.

Lemma 2. *Let* U *be a member of an oracle Martin-Löf test,* $n \in \mathbb{N}$ *and* $\tau_0 \in 2^{<\omega}$*. Then there exists* $\tau_1 \supset \tau_0$ *such that for all* $\tau_2 \supset \tau_1$*,* $\mu(U^{\tau_2} - U^{\tau_1}) < 2^{-n}$*.*

In [13] (see [16] for a different proof and more detailed presentation) it was shown that the LR degrees are countable equivalence classes.

Theorem 7. *In the LR degrees the degree of* $\emptyset'$ *bounds* $2^{\aleph_0}$ *degrees.*

Proof. By cardinal arithmetic it is enough to show that the set $\mathcal{B} = \{\beta \mid \beta \leq_{LR} \emptyset'\}$ has cardinality $2^{\aleph_0}$. Let U be the second member of the universal oracle Martin-Löf test of lemma 1, so that by definition $\mu(U^\beta) < 2^{-2}$ for all $\beta \in 2^\omega$. It suffices to define a $\emptyset'$-computable perfect tree T (as a downward closed set of strings) such that $\mu(A) < \frac{1}{2}$ where $A = \cup_{\tau \in T} U^\tau$. Then $|[T]| = 2^{\aleph_0}$ (where $[T]$ is the set of infinite paths through T), and for all $\beta \in [T]$, $U^\beta \subseteq A$. Since A is $\emptyset'$-c.e., we have by theorem 3 that for all $\beta \in [T]$, $\beta \leq_{LR} \emptyset'$. We ask that $\mu(A) < \frac{1}{2}$ (rather than $\mu(A) < 1$) simply in order that figures used should be in line with what appears in the proof of theorem 8 in [1]. It remains to define such a tree T and verify the construction.

First find a string τ such that for any extension τ' of τ, $\mu(U^{\tau'} - U^\tau) < 2^{-4}$ and define $T(\emptyset) = \tau$. The existence of such a string is ensured by lemma 2. Note that $\mu(U^{T(\emptyset)}) < 2^{-2}$. Now for each of the one element extensions of $T(\emptyset)$, say τ_i, $i = 0, 1$ find some extension $\tau_i' \supseteq \tau_i$ such that for any $\tau' \supset \tau_i'$ we have $\mu(U^{\tau'} - U^{\tau_i'}) < 2^{-6}$. Define $T(0) = \tau_0'$, $T(1) = \tau_1'$ and note that $\mu((U^{T(0)} \cup U^{T(1)}) - U^{T(\emptyset)}) < 2 \cdot 2^{-4} = 2^{-3}$ by the previous step. Continue in the same way so that at the n-th stage, where we define $T(\sigma)$ for all σ with $|\sigma| = n$, we choose a value τ for $T(\sigma)$ such that for all $\tau' \supset \tau$ we have $\mu(U^{\tau'} - U^\tau) < 2^{-(2n+4)}$. Let

$$C_n = \{T(\sigma) \mid \sigma \in 2^{<\omega} \ \wedge \ |\sigma| \leq n\}$$

and note that $C_n \subseteq C_{n+1}$. Also let $A_n = \cup_{\tau \in C_n} U^\tau$ and note that $A_n \subseteq A_{n+1}$ and $A = \cup_n A_n$. By induction, for all n, $\mu(A_n) < \sum_{i=0}^{n} 2^i \cdot 2^{-(2i+2)} = \frac{1}{2}$. Note

[1] Joe Miller and Yu Liang have independently announced the existence of an LR degree with uncountably many predecessors.

that the factor 2^i in the above sum comes from the number of strings of level i in T (and where we say that τ is of level i in T if $\tau = T(\sigma)$ for σ of length i). It remains to show that we can run the construction of T computably in $\emptyset'$, but this follows immediately from corollary 1.

After we proved theorem 7 and since high degrees often resemble $\mathbf{0'}$, we considered showing that every high LR degree has uncountably many predecessors. Using a combination of highness techniques from [6,10,15] we succeeded in showing that if A is *generalized superhigh* (i.e. $A' \geq_{tt} (A \oplus \emptyset')'$) then A has uncountably many $\leq_{LR}$-predecessors. The following theorem is a stronger result showing that if A is merely $\overline{GL_2}$ (i.e. generalized non-low$_2$, $A'' >_T (A \oplus \emptyset')'$) then it has the same property. For other $\overline{GL_2}$ constructions we refer the reader to [10].

Theorem 8. *If α is $\overline{GL_2}$ then in the LR degrees the degree of α bounds $2^{\aleph_0}$ degrees.*

The basic idea behind the proof remains the same as in the proof of theorem 7 but now we need to define T using only an oracle for α (rather than an oracle for $\emptyset'$) and α-approximate a perfect tree $T^* \subseteq T$ during the course of the construction. By theorem 8 and a cardinality argument we obtain the following.

Corollary 2. *There are $A <_{LR} B$ such that for every $A_0 \equiv_{LR} A$, $B_0 \equiv_{LR} B$ we have $A_0|_T B_0$. In fact for every $\overline{GL_2}$ set B there is A with the above property.*

Next, we provide a method for destroying LR reductions (a kind of diagonalization). As an illustration of this method we construct an antichain of LR degrees of cardinality $2^{\aleph_0}$.

Theorem 9. *There exists an antichain of cardinality $2^{\aleph_0}$ in the LR degrees.*

Proof. We wish to define a perfect tree T such that, for all distinct $A, B \in [T]$, $A \not\leq_{LR} B$. In order to do so, will make use of the following lemma which was originally proved by Kučera and which is frequently very useful in dealing with Π^0_1 classes of positive measure. For a very simple proof we refer the reader to [5].

Lemma 3. *[9] Given any Π^0_1 class $\mathcal{P}$ of positive measure there exists a Π^0_1 class of positive measure $\mathcal{K}(\mathcal{P}) \subseteq \mathcal{P}$ such that the intersection of $\mathcal{K}(\mathcal{P})$ with any Π^0_1 class is either empty or of positive measure.*

Fix a member U of a universal oracle Martin-Löf test. Assume we are given an effective listing $\{V_e\}_{e\in\omega}$ of all c.e. operators V for which there exists $q \in \mathbb{Q}$ such that for all A, $\mu(V^A) < 1-q$. We say A is LR reducible to B via V_e if $U^A \subseteq V_e^B$. Clearly $A \leq_{LR} B$ iff $A \leq_{LR} B$ via some V_e. For each e we must ensure that for all distinct $A, B \in [T]$, A is not LR reducible to B via V_e. The following lemma provides a basic diagonalization technique for the $\leq_{LR}$ reducibility.

Lemma 4. *For any e and any $\tau_0, \tau_1, \mathcal{P}_0, \mathcal{P}_1$ such that $\tau_0, \tau_1 \in 2^{<\omega}$, $\mathcal{P}_0 \subseteq [\tau_0]$, $\mathcal{P}_1 \subseteq [\tau_1]$ and $\mathcal{P}_0$, $\mathcal{P}_1$ are Π^0_1 classes of positive measure there exist $\tau'_0, \tau'_1, \mathcal{P}'_0, \mathcal{P}'_1$ such that*

- $\tau_0' \supseteq \tau_0, \tau_1' \supseteq \tau_1$,
- $\mathcal{P}_0' \subseteq \mathcal{P}_0$, $\mathcal{P}_1' \subseteq \mathcal{P}_1$ *and* $\mathcal{P}_0', \mathcal{P}_1'$ *are* Π_1^0 *classes of positive measure,*
- $\mathcal{P}_0' \subseteq [\tau_0']$, $\mathcal{P}_1' \subseteq [\tau_1']$,
- *If* $A \in \mathcal{P}_0'$ *and* $B \in \mathcal{P}_1'$ *then* A *is not* LR *reducible to* B *via* V_e.

Proof. First we define $\mathcal{Q}_0 = \mathcal{K}(\mathcal{P}_0)$, where $\mathcal{K}$ is as defined in the statement of lemma 3. Now let A be any member of $\mathcal{Q}_0$ such that $A \not\leq_{LR} \emptyset$ (hence $\{\beta \mid A \leq_{LR} \beta\}$ is null). For any $\tau \subset A$ we have that $\mathcal{Q}_0 \cap [\tau]$ is of positive measure. We define $\mathcal{Q}_1$ to be the set of all $B \in \mathcal{P}_1$ such that A is not LR reducible to B via V_e. Since upper cones in the LR degrees are of measure 0, $\mathcal{Q}_1$ is of positive measure. We define for each σ, $\mathcal{Q}_{1,\sigma} = \{B : \ B \in \mathcal{P}_1 \text{ and } [\sigma] \not\subseteq V_e^B\}$. Since a countable union of sets of measure 0 is of measure 0 and $\mathcal{Q}_1 = \bigcup_{\sigma \in U^A} \mathcal{Q}_{1,\sigma}$ there exists $\sigma \in U^A$ such that $\mathcal{Q}_{1,\sigma}$ is of positive measure. Letting σ be such, we define $\tau_0' \supset \tau_0$ to be an initial segment of A such that $\sigma \in U^{\tau_0'}$. We define $\tau_1' = \tau_1$, $\mathcal{P}_0' = \mathcal{P}_0 \cap [\tau_0']$ and $\mathcal{P}_1' = \mathcal{Q}_{1,\sigma}$.

It is now clear how to use lemma 4 in order to define T. Suppose that at stage n we have already defined $T(\sigma)$ for all σ of length $\leq n$ and that for each leaf τ of T (as presently defined) we have specified some Π_1^0 class of positive measure $\mathcal{P}_\tau$ such that all strings in T extending τ must lie in $\mathcal{P}_\tau$. For each leaf τ we first choose two incompatible extensions τ_0, τ_1 such that for each $i \leq 1$, $\mathcal{P}_\tau \cap [\tau_i]$ is of positive measure. These are potential leaves of T for the next stage. Through successive applications of lemma 4 to all pairs of potential leaves we can then define $T(\sigma)$ and $\mathcal{P}_\tau$ for all σ of length $n+1$ and each $\tau = T(\sigma)$, in such a way that if A and B extend $\tau_0 = T(\sigma), \tau_1 = T(\sigma')$ respectively for distinct strings σ and σ' of length $n+1$ and $A \in \mathcal{P}_{\tau_0}, B \in \mathcal{P}_{\tau_1}$, then A is not LR reducible to B via V_n. Since for each n there exist an infinite number of n' with $V_n = V_{n'}$, this completes the proof of the theorem.

5 Computably Enumerable LR Degrees

In this section we study the structure of the c.e. LR degrees and their relationship with the Turing reducibility. The results have been chosen so that they demonstrate how to transfer selected basic techniques from the c.e. Turing degrees (like Sacks coding and restraints) to the c.e. LR degrees. We note that the relationship between $\leq_{LR}$ and $\leq_T$ is nontrivial and goes beyond what we discuss here. For example there is a half of a minimal pair in the c.e. Turing degrees which is LR-complete [2,3]. The first author, using methods similar to those in [2], has shown that there is a noncuppable c.e. Turing degree which is LR-complete. This implies that every c.e. set which is computable by all LR-complete c.e. sets must be noncuppable. It is unknown if there are such noncomputable sets. For background in the theory of c.e. degrees we refer the reader to [17]. The following theorem demonstrates how infinitary Sacks coding can be handled in the LR degrees.

Theorem 10. *If* W *is an incomplete c.e. set, i.e.* $\emptyset' \not\leq_T W$*, then (uniformly in* W*) there is a c.e. set* B *such that* $B \leq_{LR} W$ *and* $B \not\leq_T W$.

We sketch the proof. A relativisation of the classic non-computable low for random argument of [8] (also see [4]) merely gives that for all A there exists B c.e. in A such that $B \not\leq_T A$ and $A \oplus B \leq_{LR} A$. If we assumed that W is low we could prove theorem 10 with a finitary argument similar to [8] by using a lowness technique (namely Robinson's trick). To prove the full result we need infinitary coding combined with cost efficiency considerations (see [12] for examples of cost-function arguments). We need to construct a c.e. operator V and a c.e. set B such that $U^B \subseteq V^W$ where U is a member of the universal oracle Martin-Löf test of lemma 1 (so that $\mu(U) < 2^{-1}$), $\mu(V^W) < 1$, and the following requirements are satisfied

$$P_e : \Phi_e^W = B \Rightarrow \Gamma_e^W = \emptyset'$$

where (Φ_e) is an effective enumeration of all Turing functionals and Γ_e are Turing functionals constructed by us. It is useful if we assume the *hat trick* for the functionals as well as the c.e. operators U, V (see [17] for more on this). This means, for example, that there will be infinitely many stages where (the current approximation to) U^B contains only permanent strings. The operator V can be defined ahead of the construction and it essentially enumerates into V the strings of U^B with large use. We can also make sure that V is enumerated in a prefix-free way. By such a definition of V we immediately get that $U^B \subseteq V^W$ is satisfied. So the main conflict we face is that on the one hand we want a Sacks coding for each of the P_e requirements (enumerations into $\emptyset'$ may trigger B-enumerations infinitely often) and on the other hand B-enumerations may force $\mu(V^W) = 1$ (via the the way that V is defined). The connection between B-enumerations and superfluous measure in V^W (in the sense that it does not serve $U^B \subseteq V^W$, it corresponds to intervals which are not in U^B) is roughly as in the noncomputable low for random construction of [8]: some interval σ is enumerated into U^B with use u, it enters V^W with use v and subsequently $B \upharpoonright u$ changes thus ejecting σ from U^B. Then $W \upharpoonright v$ could freeze, thus capturing a useless interval in V^W. We already have $\mu(V^W) \geq \mu(U^B)$ so we want to make sure that the measure corresponding to useless strings is bounded by 2^{-1}.

Here, however, we have an advantage over the classic argument in [8] as W may also change, thus extracting the useless string from V^W. We use this fact in order to make infinitary coding into B possible while satisfying $\mu(V^W) < 1$. The full proof can be found in [1]. This approach works even if we require $U^{B \oplus W} \subseteq V^W$ instead of $U^B \subseteq V^W$. In that case we obtain $B \oplus W \equiv_{LR} W$, $W <_T B \oplus W$ and hence the following theorem, given that there are T-incomplete sets in the complete LR degree and the known embedding results for the c.e. Turing degrees (an antichain is embeddable in every nontrivial interval).

Theorem 11. *Every c.e. LR degree contains infinitely many c.e. Turing degrees (in the form of chains and antichains) and every incomplete c.e. LR degree has no maximal c.e. Turing degree.*

As far as the global structure is concerned, we can get a similar result by relativising known constructions of low for random degrees. In particular, the relativised

noncomputable low for random construction [8] gives that for every B there is A which is B-c.e. and $A \oplus B \equiv_{LR} B$, $B <_T A \oplus B$; and a slight extension of the argument gives that every B is T-below an antichain of T degrees in the same LR degree, hence the following theorem.

Theorem 12. *Every LR degree contains infinitely many Turing degrees (in the form of chains and antichains) and no maximal Turing degree.*

Next we show a splitting theorem which also shows how Sacks restraints work in the LR degrees.

Theorem 13. *If A is c.e. and not low for random then there are c.e. B, C such that $B \cap C = \emptyset$, $B \cup C = A$, $B \not\leq_{LR} C$ and $C \not\leq_{LR} B$.*

Proof. Here is a sketch of the proof. The main idea is as in the classic Sacks splitting theorem. We just have to translate the main tools like *length of agreement* and *Sacks restraints* to the case of LR reductions. This will not be a problem as $\leq_{LR}$ is Σ^0_3. Fix a member U of a universal oracle Martin-Löf test; an LR reduction is defined via a c.e. operator V (as opposed to a Turing functional), a $q \in \mathbb{Q}$ and A is LR reducible to B via V, q if

$$\mu(V^B) < 1 - q \text{ and } U^A \subseteq V^B. \tag{2}$$

To define the length of agreement $\ell(U^A, V^B)$ of this possible reduction consider computable enumerations of U, V, A, B. Let M_s be the set of strings σ such that $\sigma \in U_s^{A_t}$ for some $t \leq s$ and let (σ_s) be a computable enumeration of $M = \cup_s M_s$. Now for all s we define $\ell(U^A, V^B)[s]$ to be the maximum n such the following hold:

- $\sigma_n[s] \downarrow$ (i.e. the nth member of M has been enumerated by stage s)
- $\forall i \leq m$ $([\sigma_i] \subseteq V_s^{B_s} \ \vee \ \sigma_i \notin U_s^{A_s})$
- $\mu(\{\sigma_i \mid i < n \ \wedge \ [\sigma_i] \subseteq V_s^{B_s}\}) < 1 - q$.

It is clear that reduction (2) is total iff $\liminf_s \ell(U^A, V^B)[s] = \infty$. Now in general, if we wish to destroy a given reduction like (2) where A is a given c.e. set of nontrivial LR degree and B is enumerated by us, its enough if we respect the following restraint at every stage s:

$$r(V, q, s) = \mu t \ [\forall i \leq \ell(U^A, V^B)[s] \ ([\sigma_i] \subseteq V_s^{B_s} \text{ with } B\text{-use} < t \ \vee \ \sigma_i \notin U_s^{A_s})].$$

Indeed, it can be shown that if $r(V, q, s)$ is respected for a cofinite set of stages then

$$\lim_s \ell(U^A, V^B)[s] < \infty. \tag{3}$$

So either there is a stage where the measure goes over the threshold $1 - q$, or there is some i such that σ_i is a permanent resident of U^A and σ_i is never covered by strings in V^B. In any case (2) is destroyed and the restraint comes to a limit. This is all we need in order to apply the classic Sacks splitting argument (see [17] for a presentation). For more details we refer to [1].

The Sacks restraints argument in theorem 13 works exactly as in the Turing degrees, only that the restraints are defined in a different way. Hence it is natural to ask whether given a noncomputable B we can run the restraints argument in the Turing degrees, constructing some A such that $B \not\leq_T A$, while we code part of B into A so that $B \leq_{LR} A$. It turns out that this is possible and the reason is that compared to the Turing restraints, the LR restraints are more demanding. For example a single B-enumeration below the length of agreement of some potential reduction $\Phi^A = B$ (accompanied by the existing restraint) suffices in order to destroy the reduction; but no single such enumeration suffices in order to destroy a potential LR reduction $U^B \subseteq V$.

Theorem 14. *Given noncomputable c.e. B there is a c.e. A such that $B \not\leq_T A$ and $B \leq_{LR} A$. Moreover A can be chosen such that $A <_T B$.*

References

1. Barmpalias, G., Lewis, A.E.M., Soskova, M.: Randomness, Lowness and Degrees, Draft submitted for publication, 19 pages, current version available at `http://www.maths.leeds.ac.uk/~georgeb/`.
2. Barmpalias G., Montalban, A.: A cappable almost everywhere dominating computably enumerable degree, Electronic Notes in Theoretical Computer Science, Volume **167** (2007).
3. Binns, S., Kjos-Hanssen, B., Miller, J.S., Solomon, R.: Lowness notions, measure and domination, in preparation.
4. Downey, R., Hirschfeldt, D.: Algorithmic Randomness and Complexity, in preparation. Current draft available at `http://www.mcs.vuw.ac.nz/~downey/`.
5. Downey, R., Miller, J.S.: A basis theorem for Π^0_1 classes of positive measure and jump inversion for random reals, Proceedings of the American Mathematical Society **134** (1), pages 283-288 (2006).
6. Jockusch, C.G.: Simple proofs of some theorems on high degrees of unsolvability, Canadian Journal of Mathematics **29** (1977)
7. Kjos-Hanssen, B.: Low for random reals and positive-measure domination, to appear in the Proceedings of the AMS.
8. Kučera, A., Terwijn, S.A.: Lowness for the class of random sets, Journal of Symbolic Logic, vol. **64** (1999) 1396–1402.
9. Kučera, A.: Measure, Π^0_1 classes and complete extensions of PA. In Recursion theory week (Oberwolfach, 1984), Volume **1141** of Lecture Notes in Math., pages 245-259. Springer, Berlin, 1985.
10. Lerman, M.: Degrees of Unsolvability: Local and Global Theory, Springer-Verlag (July 1983)
11. Nies, A.: Low for random sets: the story. Unpublished draft which is available at the authors webpage `http://www.cs.auckland.ac.nz/~nies/`.
12. Nies, A.: Computability and Randomness, monograph to appear. Current draft available at `http://www.cs.auckland.ac.nz/~nies/`.
13. Nies, A.: Lowness properties and randomness. Advances in Mathematics, **197**: 274-305 (2005).
14. Sacks, G.: Degrees of Unsolvability, Princeton University Press, 1963.

15. Shore R., Slaman, T.: Working below a high recursively enumerable degree, Journal of Symbolic Logic **58** (1993), 824–859.
16. Simpson, S.G.: Almost everywhere domination and superhighness. *Draft* available at `http://www.math.psu.edu/simpson/`.
17. Soare, R.I.: Recursively enumerable sets and degrees, Berlin London: Springer-Verlag, 1987

Computability on Subsets of Locally Compact Spaces

Yatao Xu[1] and Tanja Grubba[2]

[1] Nanjing University, China
yataoxu@gmail.com
[2] University of Hagen, Germany
Tanja.Grubba@FernUni-Hagen.de

Abstract. In this paper we investigate aspects of effectivity and computability on closed and compact subsets of locally compact spaces. We use the framework of the representation approach, TTE, where continuity and computability on finite and infinite sequences of symbols are defined canonically and transferred to abstract sets by means of notations and representations. This work is a generalization of the concepts introduced in [4] and [22] for the Euclidean case and in [3] for metric spaces. Whenever reasonable, we transfer a representation of the set of closed or compact subsets to locally compact spaces and discuss its properties and their relations to each other.

1 Introduction

Computable Analysis connects Computability/Computational Complexity with Analysis/Numerical Computation by combining concepts of approximation and of computation. During the last 70 years various mutually non-equivalent models of real number computation have been proposed ([19], Chap. 9 in [22]). Among these models the representation approach (Type-2 Theory of Effectivity, TTE) proposed by Grzegorczyk and Lacombe [7,14] seems to be particularly realistic, flexible and expressive. So far the study of computability on sets of points, sets (open, closed, compact) and continuous functions has developed mainly bottom-up, i.e., from the real numbers to Euclidean space and metric spaces [26,4,24,22,27,3,28]. But often generalizations to more general spaces are needed (locally compact Hausdorff spaces [5], non-metrizable spaces [25], second countable T_0-spaces [17,8]).

In this article we investigate computability on locally compact spaces with the following motivation:

- Computability on metric spaces has been widely and deeply studied. However, the concept of a metric space is not powerful enough to capture all the interesting phenomena in computable analysis. Many results in classical topology, that hold for more general spaces such as Hausdorff spaces or locally compact spaces, can be tied with effectivity.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 100–114, 2007.
© Springer-Verlag Berlin Heidelberg 2007

– Locally compact spaces inherit some nice properties of metric spaces. Roughly speaking, a locally compact space with a countable base is metrizable. This demonstrates that locally compact spaces are "quite close" to metric spaces.
– Furthermore general topological spaces, especially locally compact spaces, have practical applications. For example in [5] Collins used locally compact spaces to study computability of reachable sets for nonlinear dynamic and control systems.

For these reasons it is necessary, reachable and meaningful to study computability on locally compact spaces.

In [4] and [22] and in [3] several representations are introduced for subsets of the Euclidean space and of a metric space respectively. We don't consider those, which are defined by means of the metric distance function. We prove that the properties $\delta_{union} \equiv \delta_{Sierpinski} \equiv \delta_{dom} \equiv \delta^{>} \equiv \delta_{fiber}$ and $\delta^{<} \leq \delta_{range}$ for closed sets and $\delta_{min-cover} \leq \delta_{cover} \equiv \delta_{\mathcal{K}}^{>}$ for compact sets, shown in [3] for metric spaces, hold true for locally compact spaces as well. A crucial point for the fact, that these results can be transfered to locally compact spaces, are two additional axioms that are required in the metric case for some of these reductions,

1. the existence of nice closed balls,
2. the effective covering property.

Both properties hold true for computably locally compact spaces. The first follows directly from the definition of computably locally compact spaces, as the closure of each base element is compact. The second property utilizes the completeness of the cover representation: a name lists all names of all finite basic covers of a compact set (shown in Lemma 1). Furthermore we introduce a new representation κ^{net} where a compact set is denoted by a decreasing sequence of finite basic covers whose intersection equals to the compact set. We show that κ^{net} is equivalent to the cover representation.

This article is organized as follows: In Section 2, we sketch some basic notions on TTE and provide some fundamental definitions and properties of representations of points and sets in computable T_0-spaces. In Section 3, we introduce and characterize computably locally compact spaces and computably Hausdorff spaces. In Section 4 we define and compare various representations of closed subsets and compact subsets of computably locally compact, computably Hausdorff spaces. The conclusion is drawn in the last Section. Since this is an extended abstract proofs of the main theorems are given in the Appendix.

2 Preliminaries

This Section consists of two parts. In Section 2.1, we sketch the concept of TTE. In Section 2.2, we introduce computable T_0-spaces and the underlying representations of points and sets.

2.1 Type-2 Theory of Effectivity (TTE)

In this article we use the framework of TTE (Type-2 theory of effectivity) [22] to explore several aspects of computability in locally compact spaces. The Type-2 theory of effectivity defines computability on Σ^* and Σ^ω via Type-2 machines and transfers a computability concept to "abstract" sets by means of naming systems.

We assume that Σ is a fixed finite alphabet containing the symbols 0 and 1 and consider computable functions on finite and infinite sequences of symbols Σ^* and Σ^ω, respectively, which can be defined, for example, by Type-2 machines, i.e., Turing machines reading from and writing on finite or infinite tapes. A Type-2 machine may have several one-way read-only input tapes, several two-way work tapes and a unique one-way write-only output tape. It permits infinite input or output, and has a finiteness property, that is, each group of prefixes of the inputs determines a unique prefix of the output. A partial function from X to Y is denoted by $f :\subseteq X \to Y$. For $\alpha_i \in \{*, \omega\}$, a function $f :\subseteq \Sigma^{\alpha_1} \times \ldots \times \Sigma^{\alpha_k} \to \Sigma^{\alpha_0}$ is called computable if $f = f_M$ for some Type-2 machine M.

The "wrapping function" $\iota : \Sigma^* \to \Sigma^*$, $\iota(a_1 a_2 \ldots a_k) := 110 a_1 0 a_2 0 \ldots a_k 011$ encodes words such that $\iota(u)$ and $\iota(v)$ cannot overlap properly. We consider standard functions for finite or countable tupling on Σ^* and Σ^ω denoted by $\langle \cdot \rangle$ and projections of the inverse π_i for $i \in \mathbb{N}$. By "$\lhd$" we denote the subword relation. A sequence $p \in \Sigma^* \cup \Sigma^\omega$ is called a list of M if $M = \{u \mid \iota(u) \lhd p\}$.

We use the concept of multi-functions. A *multi-valued partial function*, or *multi-function* for short, from A to B is a triple $f = (A, B, \mathrm{R}_f)$ such that $\mathrm{R}_f \subseteq A \times B$ (the *graph* of f). Usually we will denote a multi-function f from A to B by $f :\subseteq A \rightrightarrows B$. For $X \subseteq A$ let $f[X] := \{b \in B \mid (\exists a \in X)(a, b) \in \mathrm{R}_f\}$ and for $a \in A$ define $f(a) := f[\{a\}]$. Notice that f is well-defined by the values $f(a) \subseteq B$ for all $a \in A$. We define $\mathrm{dom}(f) := \{a \in A \mid f(a) \neq \emptyset\}$. In the applications we have in mind, for a multi-function $f :\subseteq A \rightrightarrows B$, $f(a)$ is interpreted as the set of all results which are "acceptable" on input $a \in A$. Any concrete computation will produce on input $a \in \mathrm{dom}(f)$ some element $b \in f(a)$, but usually there is no method to select a specific one. In accordance with this interpretation the "functional" composition $g \circ f :\subseteq A \rightrightarrows D$ of $f :\subseteq A \rightrightarrows B$ and $g :\subseteq C \rightrightarrows D$ is defined by $\mathrm{dom}(g \circ f) := \{a \in A \mid a \in \mathrm{dom}(f) \text{ and } f(a) \subseteq \mathrm{dom}(g)\}$ and $g \circ f(a) := g[f(a)]$ (in contrast to "non-deterministic" or "relational" composition gf defined by $g\, f(a) := g[f(a)]$ for *all* $a \in A$).

Notations $\nu :\subseteq \Sigma^* \to M$ and representations $\delta :\subseteq \Sigma^\omega \to M$ are used for introducing relative continuity and computability on "abstract" sets M. For a representation $\delta :\subseteq \Sigma^\omega \to M$, if $\delta(p) = x$ then the point $x \in M$ can be identified by the "name" $p \in \Sigma^\omega$. For representations $\delta :\subseteq \Sigma^\omega \to M$ and $\delta' :\subseteq \Sigma^\omega \to M'$ define $[\delta, \delta'] :\subseteq \Sigma^\omega \to M \times M'$ by $[\delta, \delta']\langle p, p' \rangle := \delta(p) \times \delta(p')$ and $[\delta]^\omega :\subseteq \Sigma^\omega \to M^\omega$ by $[\delta]^\omega \langle p_0, p_1, p_2, \ldots \rangle := \delta(p_0) \times \delta(p_1) \times \delta(p_2) \times \ldots$.

For a naming systems $\gamma :\subseteq Y_i \to M_i$, $Y_i = \Sigma^*$ or Σ^ω, the set X is called γ-open (γ-clopen, γ-r.e. , γ-decidable), iff $\gamma^{-1}[X]$ is open (clopen, r.e. open, decidable) in $\mathrm{dom}(\gamma)$.

For naming systems $\gamma_i :\subseteq Y_i \to M_i$ $(i = 0, \ldots, k)$, a function $h :\subseteq Y_1 \times \ldots \times Y_k \to Y_0$ is a $(\gamma_1, \ldots, \gamma_k, \gamma_0)$-realization of $f :\subseteq M_1 \times \ldots \times M_k \rightrightarrows M_0$, if $\gamma_0 \circ h(p_1, \ldots, p_k) \in f(\gamma_1(p_1), \ldots, \gamma_k(p_k))$ whenever $f(\gamma_1(p_1), \ldots, \gamma_k(p_k))$ exists. The function h is called a strong realization of f, if $h(p_1, \ldots, p_k) = \uparrow$ for all $\langle p_1, \ldots, p_k\rangle \in \mathrm{dom}([\gamma_1, \ldots, \gamma_k])$ with $[\gamma_1, \ldots, \gamma_k](p_1, \ldots, p_k) \notin \mathrm{dom}(f)$. The multi-function f is $(\gamma_1, \ldots, \gamma_k, \gamma_0)$-continuous (-computable), if it has a continuous (computable) $(\gamma_1, \ldots, \gamma_k, \gamma_0)$-realization.

For naming systems $\gamma :\subseteq Y \to M$ and $\gamma' :\subseteq Y' \to M'$ $(Y, Y' \in \{\Sigma^*, \Sigma^\omega\})$, let $\gamma \leq_t \gamma'$ (t-reducible) and $\gamma \leq \gamma'$ (reducible), iff there is some continuous or computable function $f :\subseteq Y \to Y'$ such that $\gamma(y) = \gamma' f(y)$ for all $y \in \mathrm{dom}(\gamma)$, respectively. Define t-equivalence and equivalence as follows: $\gamma \equiv_t \gamma' \iff (\gamma \leq_t \gamma' \text{ and } \gamma' \leq_t \gamma)$ and $\gamma \equiv \gamma' \iff (\gamma \leq \gamma' \text{ and } \gamma' \leq \gamma)$, respectively.

Two representations induce the same continuity or computability, iff they are t-equivalent or equivalent, respectively. If multi-functions on represented sets have realizations, then their composition is realized by the composition of the realizations. In particular, the computable multi-functions on represented sets are closed under composition. Much more generally, the computable multi-functions on represented sets are closed under flowchart programming with indirect addressing [23]. This result allows convenient informal construction of new computable multi-functions on multi-represented sets from given ones.

Let $\nu_{\mathbb{N}} :\subseteq \Sigma^* \to \mathbb{N}$ be some standard notation of the natural numbers, ρ the standard representation of $\mathbb{R}$. A $\rho^<$-name represents a real number by lower rational bounds. $\rho^<(p) = x$, if p is a list of all rational numbers $a < x$ and η^{ab} a standard representation of F^{ab}, the partial continuous functions $f :\subseteq \sigma^a \to \sigma^b$ with open or G_δ domain, if $b = *$ or $b = \omega$ respectively, with properties $\mathrm{utm}(\eta^{ab})$ and $\mathrm{smn}(\eta^{ab})$.

2.2 Representations of Points and Sets in Computable T_0-Spaces

In this Section we introduce computable T_0-spaces together with some fundamental representations of points and sets.

A topological space $\mathbf{X} = (X, \tau)$ is a T_0-space, if for all $x, y \in X$ such that $x \neq y$, there is an open set $O \in \tau$ such that $x \in O$ iff $y \notin O$. In a T_0-space, every point can be identified by the set of its neighborhoods $O \in \tau$. $\mathbf{X}$ is called second-countable, if it has a countable base [6].

In the following we consider only second countable T_0-spaces. For introducing concepts of effectivity we assume that some notation ν of a base β with recursive domain is given.

Definition 1 (computable T_0-space)
A computable T_0-space *is a tuple* $\mathbf{X} = (X, \tau, \beta, \nu)$ *such that* (X, τ) *is a second countable* T_0*-space and* $\nu :\subseteq \Sigma^* \to \beta$ *is a notation of a base* β *of* τ *with recursive domain,* $U \neq \emptyset$ *for* $U \in \beta$ *and* $\mathbf{X}$ *has* computable intersection*: there is a computable function* $h :\subseteq \Sigma^* \times \Sigma^* \to \Sigma^\omega$ *such that for all* $u, v \in \mathrm{dom}(\nu)$*,*

$$\nu(u) \cap \nu(v) = \bigcup \{\nu(w) \mid w \in \mathrm{dom}(\nu) \quad \text{and} \quad \iota(w) \lhd h(u, v)\}\,. \tag{1}$$

Call two computable T_0-spaces $\mathbf{X}_1 = (X, \tau, \beta_1, \nu_1)$ and $\mathbf{X}_2 = (X, \tau, \beta_2, \nu_2)$ recursively related, if and only if there are computable functions $g, g' : \subseteq \Sigma^* \to \Sigma^\omega$ such that

$$\nu_1(u) = \bigcup_{\iota(w) \lhd g(u)} \nu_2(w) \quad \text{and} \quad \nu_2(v) = \bigcup_{\iota(w) \lhd g'(v)} \nu_1(w)\,. \tag{2}$$

We are interested in computability concepts which are "robust", that is, which do not change if a space is replaced by a recursively related one.

In the following let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable T_0-space. Now we introduce the standard representation of X.

Definition 2 (standard representation δ of X). *Define the standard representation $\delta : \subseteq \Sigma^\omega \to X$ as follows: $\delta(p) = x$ iff*

- $u \in \mathrm{dom}(\nu)$ *if* $\iota(u) \lhd p$
- $\{u \in \mathrm{dom}(\nu) \mid x \in \nu(u)\} = \{u \mid \iota(u) \lhd p\}$.

A δ-name p of an element $x \in X$ is a list of all words u such that $x \in \nu(u)$. The definition of δ corresponds to the definition of $\delta'_{\mathbf{S}}$ in Lemma 3.2.3 of [22], in particular, δ is admissible with final topology τ (Sec. 3.2 in [22]).

Definition 3 (union representation of open and closed sets)

1. *Define the union representation $\theta^{un} : \subseteq \Sigma^\omega \to \tau$ by*

$$\mathrm{dom}(\theta^{un}) := \{q \in \Sigma^\omega | u \in \mathrm{dom}(\nu) \text{ if } \iota(u) \lhd q\} \quad \text{and} \quad \theta^{un}(p) := \bigcup_{\iota(u) \lhd p} \nu(u)\,.$$

2. *Define the union representation $\psi^{un} : \subseteq \Sigma^\omega \to \tau^c$ by $\psi^{un}(p) := X \setminus \theta^{un}(p)$.*

Thus, $\theta^{un}(p)$ is the union of all $\nu(u)$ such that u is listed by p. The union representation of the closed sets is defined by the union representation of their complements.

For technical reasons we define a notation $\nu^* : \subseteq \Sigma^* \to \{M \subseteq \beta \mid M \text{ is finite}\}$ of all finite sets of base elements by $\mathrm{dom}(\nu^*) := \{w \in \Sigma^* \mid u \in \mathrm{dom}(\nu) \text{ if } \iota(u) \lhd w\}$ and

$$\nu^*(w) := \{\nu(u) \mid \iota(u) \lhd w\}$$

and a notation $\theta^* : \subseteq \Sigma^* \to \tau^{fin}$ of all open sets that can be written as the union of finitely many base elements by

$$\theta^*(w) := \bigcup \nu^*(w).$$

The representations δ and θ^{un} are not only very natural, but they can be characterized up to equivalence as maximal elements among representations for which the element relation is open or r.e., respectively. Furthermore the following properties hold.

Lemma 1. *For computable T_0-spaces,*

1. *"$O \neq \emptyset$" is θ^{un}-r.e.,*
2. *countable union on τ is $([\theta^{un}]^\omega, \theta^{un})$-computable,*
3. *intersection is $(\theta^{un}, \theta^{un}, \theta^{un})$-computable,*
4. *finite intersection is (ν^*, θ^{un})-computable.*

Proof. Omitted.

Equivalently to the computability of finite intersection on the base is the existence of an r.e. set $I \subseteq \Sigma^* \times \mathrm{dom}(\nu)$, such that for all $w \in \Sigma^*$ and $u \in \mathrm{dom}(\nu)$

$$\bigcap_{\iota(v) \lhd w} \nu(v) = \bigcup_{(w,u) \in I} \nu(u). \tag{3}$$

Furthermore the set $P := \{w \in \Sigma^* \mid (\exists u \in \mathrm{dom}(\nu))\,(w,u) \in I\}$ of all finite prefixes of δ-names is r.e..

A topological space is a T_2-space (also called Hausdorff space), if for all $x, y \in X$ such that $x \neq y$, there are disjoint open sets $O, O' \in \tau$ such that $x \in O$ and $y \in O'$. A subset $K \subseteq X$ of a Hausdorff space (X, τ) is compact, if every open cover of K by elements of the base has a finite subcover. Let

$$\mathcal{K}(\mathbf{X}) := \{K \subseteq X \mid K \text{ compact}\}$$

denote the set of all compact subsets of a Hausdorff space (X, τ). We write $\mathcal{K}$ instead of $\mathcal{K}(\mathbf{X})$, if there is no need to specify the space or if it's obvious which space we refer to.

In the following we generalize the representation κ_c of the compact subsets of the Euclidean space [22] and δ_{cover} of the compact subsets of a computable metric space [3] defined by listing all finite basic subcovers from a countable base.

Definition 4 (cover representation κ^c of compact sets). *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable T_0-space and let (X, τ) be a Hausdorff space. Define a representation $\kappa^c : \subseteq \Sigma^\omega \to \mathcal{K}$ as follows: $K = \kappa^c(p)$ iff*

- *$w \in \mathrm{dom}(\theta^*)$ if $\iota(w) \lhd p$,*
- *$\{w \in \Sigma^* \mid \iota(w) \lhd p\} = \{w \in \Sigma^* \mid K \subseteq \theta^*(w)\}$.*

Roughly speaking, p is a name of K, if it is a list of all (!) names of all finite basic subcovers of K with base elements.

For technical reasons we define a representation of all finite sets of compact sets $\kappa^* : \subseteq \Sigma^\omega \to \{M \subseteq \mathcal{K} \mid M \text{ is finite }\}$ by

$$\kappa^*(p) = \{K_1, \ldots, K_k\} :\iff p = 1^k 0 \langle p_1, \ldots, p_k \rangle \text{ and } \kappa^c(p_i) = K_i \text{ for all } i \in \{1, \ldots, k\}.$$

Lemma 2. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable T_0-space and let (X, τ) be a Hausdorff space, then*

1. *"$K \subseteq O$" is (κ^c, θ^{un})-r.e.,*
2. *finite union on $\mathcal{K}$ is (κ^*, κ^c)-computable,*
3. *countable intersection on $\mathcal{K}$ is $([\kappa]^\omega, \kappa^c)$-computable.*

Proof. Omitted.

Every closed subset of a compact set is compact. The next Lemma gives an effective version.

Lemma 3. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable T_0-space and let (X, τ) be a Hausdorff space. The mapping $\mathrm{F} : \tau^c \times \mathcal{K} \to \mathcal{K}$ defined by*

$$\mathrm{F}(A, K) := A \cap K$$

is $(\psi^{un}, \kappa^c, \kappa^c)$-computable.

Proof. Omitted.

3 Effectivity in Locally Compact Hausdorff Spaces

In this Section we introduce an effective version of the Hausdorff property and an effective version of locally compactness.

Definition 5 (computably Hausdorff). *A computable T_0-space $\mathbf{X} = (X, \tau, \beta, \nu)$ is called* computably Hausdorff *if there exists an r.e. set $H \subseteq \mathrm{dom}(\nu) \times \mathrm{dom}(\nu)$ such that*

$$(\forall (u, v) \in H) \quad \nu(u) \cap \nu(v) = \emptyset, \tag{4}$$

$$(\forall x, y \in X \text{ with } x \neq y)\,(\exists (u, v) \in H)\; x \in \nu(u) \wedge y \in \nu(v). \tag{5}$$

Lemma 4. *For computable T_0-spaces,*

$$\mathbf{X} \text{ computably Hausdorff} \iff \{(x, y) \in X \times X \mid x \neq y\} \text{ is } (\delta, \delta) - \text{r.e.}.$$

Proof. Omitted.

Lemma 4 implies the robustness of the computably Hausdorff property, as δ is robust.

Lemma 5. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computably Hausdorff space.*

1. *The mapping $\mathrm{F}_1 :\subseteq X \times X \rightrightarrows \beta \times \beta$ defined by $\mathrm{dom}(F_1) = \{(x, y) \mid x \neq y\}$ and*

$$(U, V) \in \mathrm{F}_1(x, y) :\iff x \in U \text{ and } y \in V \text{ and } U \cap V = \emptyset$$

is $(\delta, \delta, \nu, \nu)$-computable.

2. *The mapping* $\mathrm{F}_2 :\subseteq X \times \mathcal{K} \rightrightarrows \beta \times \tau$ *defined by* $\mathrm{dom}(F_2) = \{(x, K) \mid x \notin K\}$ *and*

$$(U, O) \in \mathrm{F}_2(x, K) :\iff x \in U \quad \text{and} \quad K \subseteq O \quad \text{and} \quad U \cap O = \emptyset$$

is $(\delta, \kappa^c, \nu, \theta^*)$*-computable.*

3. *The mapping* $\mathrm{F}_3 :\subseteq \mathcal{K} \times \mathcal{K} \rightrightarrows \tau \times \tau$ *defined by* $\mathrm{dom}(F_3) = \{(K, K') \mid K \cap K' = \emptyset\}$ *and*

$$(O, O') \in \mathrm{F}_3(K, K') :\iff K \subseteq O \quad \text{and} \quad K' \subseteq O' \quad \text{and} \quad O \cap O' = \emptyset$$

is $(\kappa^c, \kappa^c, \theta^*, \theta^*)$*-computable.*

Proof. Omitted.

Every compact subspace of a Hausdorff space is closed. The next theorem is an effective version.

Theorem 1. *For computably Hausdorff spaces,* $\kappa^c \leq \psi^{un}$*.*

Proof. See Appendix.

A topological space (X, τ) is called locally compact, if for every point $x \in X$, there exists a neighborhood O of x such that the closure $\bar{O}$ is compact. Next we introduce an effective version of locally compactness by means of the representation κ^c of the compact subsets of a Hausdorff space.

Definition 6 (computably locally compact space). *A computable* T_0*-space* $\mathbf{X}' = (X, \tau, \beta', \nu')$ *is called a* computably locally compact space *if* (X, τ) *is a Hausdorff space and there is some computable* T_0*-space* $\mathbf{X} = (X, \tau, \beta, \nu)$ *such that* $\mathrm{CLS} : \beta \to \mathcal{K}(\mathbf{X})$ *defined by* $\mathrm{CLS}(U) := \bar{U}$ *is* (ν, κ^c)*-computable and* $\mathbf{X}'$ *and* $\mathbf{X}$ *are recursively related.*

The definition of computably locally compactness ensures its robustness. In the following if $\mathbf{X} = (X, \tau, \beta, \nu)$ is a computably locally compact space, we suppose CLS to be (ν, κ^c)-computable (without changing the base or its notation).

If $\mathbf{X}$ is a computably locally compact space, then it is locally compact since the closure of each base element is compact. Therefore $\mathbf{X}$ is Tychonoff, thus regular (and a Hausdorff space) and even metrizable since $\mathbf{X}$ is second countable ([6]).

Lemma 6. *For computably locally compact spaces,*

1. *"*$\bar{U} \subseteq O$*" is* (ν, θ^{un})*-r.e.,*
2. *"*$\bar{O} \subseteq O'$*" is* (θ^*, θ^{un})*-r.e..*

Proof. Omitted.

In [3] and [5] property 1 is defined as the "effective covering property" of a computable metric space. As this property holds true for each computably locally compact space, we do not need an additional axiom.

For every compact subspace K of a locally compact space X and every open set $V \subseteq X$ that contains K, there exists an open set $U \subseteq X$ such that $K \subseteq U \subseteq \bar{U} \subseteq V$ and $\bar{U}$ is compact. The next Lemma gives an effective version.

Lemma 7. *Let* $\mathbf{X} = (X, \tau, \beta, \nu)$ *be a computably locally compact space. The mapping* $\mathrm{F} :\subseteq \mathcal{K} \times \tau \rightrightarrows \tau$ *defined by* $\mathrm{dom}(F) = \{(K, O) \in \mathcal{K} \times \tau \mid K \subseteq O\}$ *and*

$$U \in \mathrm{F}(K, O) :\iff K \subseteq U \subseteq \bar{U} \subseteq O$$

is $(\kappa^c, \theta^{un}, \theta^*)$*-computable.*

Proof. Omitted.

Computably regular spaces have been introduced in [17] and [9]. The following theorem gives an effective version of the classical hierarchy, every locally compact space is regular and every regular space is a Hausdorff space.

Theorem 2. *1. A computable* T_0*-space is computably regular, if it is computably locally compact and computably Hausdorff.*
2. A computable T_0*-space is computably Hausdorff, if it is computably regular.*

Proof. Omitted.

4 Computability on Subsets of Computably Locally Compact, Computably Hausdorff Spaces

4.1 Computability on Closed Subsets

In this Section we study several representations of the closed subsets of a computably locally compact, computably Hausdorff space.

Definition 7 (representations of closed sets). *Let* $\mathbf{X} = (X, \tau, \beta, \nu)$ *be a computably locally compact, computably Hausdorff space. Let* $\eta_p^{\omega *}$, $\eta_p^{\omega\omega}$ *be standard representation of the set of continuous functions of* $F :\subseteq \Sigma^\omega \to \Sigma^*$ *and* $F :\subseteq \Sigma^\omega \to \Sigma^\omega$*, respectively.*

1. *Define the domain representation* $\psi^{dom} :\subseteq \Sigma^\omega \to \tau_c$ *by*

$$\psi^{dom}(p) = A :\iff \eta_p^{\omega *} \text{ is a strong } (\delta, \nu_{\mathbb{N}})\text{-realization of } f :\subseteq X \to \mathbb{N} \text{ such that } \mathrm{dom}(f) = A^c.$$

2. *Define the Sierpinski representation* $\psi^{sie} :\subseteq \Sigma^\omega \to \tau_c$ *by*

$$\psi^{sie}(p) = A :\iff \eta_p^{\omega\omega} \text{ is a } (\delta, \rho^<)\text{-realization of } \mathrm{cf}_A : X \to \mathbb{R}.$$

Here $\mathrm{cf}_A : X \to \mathbb{R}$ *means that*

$$x \mapsto \begin{cases} 0 & \text{if } x \in A \\ 1 & \text{otherwise} \end{cases}$$

3. *Define the fiber representation* $\psi^{fiber} :\subseteq \Sigma^\omega \to \tau_c$ *by*

$$\psi^{fiber}(p) = A :\iff \eta_p^{\omega\omega} \text{ is a } (\delta, \rho)\text{-realization of } f : X \to \mathbb{R} \text{ such that } f^{-1}\{0\} = A.$$

4. *Define the inner representation* $\psi^< :\subseteq \Sigma^\omega \to \tau_c$ *as follows:* $\psi^<(p) = A$ *iff*
   - $u \in \mathrm{dom}(\nu)$ *if* $\iota(u) \lhd p$,
   - $\{w \mid \iota(w) \lhd p\} = \{w \mid \nu(w) \cap A \neq \emptyset\}$.
5. *Define the enumeration representation* $\psi^{range} :\subseteq \Sigma^\omega \to \tau_c$ *by*

$$\begin{aligned} \psi^{range}(0^\omega) &:= \emptyset, \\ \psi^{range}(0^k 1 p) &:= cls \circ range([\nu_{\mathbb{N}} \to \delta]_{\mathbb{N}}(p)). \end{aligned}$$

6. *Define the outer representation* $\psi^> :\subseteq \Sigma^\omega \to \tau_c$ *as follows* $\psi^>(p) = A$ *iff*
   - $u \in \mathrm{dom}(\nu)$ *if* $\iota(u) \lhd p$,
   - $\{w \mid \iota(w) \lhd p\} = \{w \mid \overline{(\nu(w))} \cap A = \emptyset\}$.

All these representations of closed sets with the exception of $\psi^>$ are well-defined and robust even for computable T_0-spaces.

The representation $\psi^>$ is well-defined for computably locally compact spaces, as they are regular. For closed sets A and B, if

$$\{w \in \mathrm{dom}(\nu) \mid \overline{\nu(w)} \cap A = \emptyset\} = \{w \in \mathrm{dom}(\nu) \mid \overline{\nu(w)} \cap B = \emptyset\}$$

then

$$\{w \in \mathrm{dom}(\nu) \mid \overline{\nu(w)} \subseteq A^c\} = \{w \in \mathrm{dom}(\nu) \mid \overline{\nu(w)} \subseteq B^c\}.$$

Let $x \in A^c$ then there exists some $V \in \beta$ such that $x \in V \subseteq \bar{V} \subseteq A^c$ as X is regular. It follows $x \in V \subseteq B^c$, then $A^c \subseteq B^c$. By symmetry we can conclude that $A^c = B^c$, hence $A = B$.

However $\psi^>$ is not well-defined for Hausdorff spaces in general.

Example 1. Define a topological space $(\mathbb{R}, \tau)$ by $\tau := \{G \setminus E | G \in \tau_{\mathbb{R}}, E \subseteq \mathbb{Q}\}$, where $\tau_{\mathbb{R}}$ is the set of all open subsets of $\mathbb{R}$. The space $(\mathbb{R}, \tau)$ is a Hausdorff space, since for any two different $x_1, x_2 \in \mathbb{R}$, by the density of real numbers, there exists x between them, then we can find two open sets $(-\infty, x) \setminus E_1$, $(x, \infty) \setminus E_2$, which contains x_1, x_2 respectively, such that $((-\infty, x) \setminus E_1) \cap ((x, \infty) \setminus E_2) = \emptyset$.

Next we show that for any $G \in \tau_{\mathbb{R}}$, $G \setminus E_i$ have the same closure as G, $i \in \mathbb{N}$. In fact, we just need to prove $\bar{G} \subseteq \overline{G \setminus E}$, for any $E \subseteq \mathbb{Q}$. Suppose $x \in \bar{G}$ by the definition of closure, for any neighborhood $G' \setminus E'$ of x, $(G' \setminus E') \cap G \neq \emptyset$. Since G and G' are open sets of $\mathbb{R}$ with common topology, and by the density of irrational numbers we conclude that there exists some irrational number $y \in (G' \setminus E') \cap G$. As $E \subseteq \mathbb{Q}$, then $y \in (G' \setminus E') \cap (G \setminus E)$, that is, $(G' \setminus E') \cap (G \setminus E) \neq \emptyset$. Hence, $x \in \overline{G \setminus E}$, as required.

Given a closed subset $\mathbb{Q} \in \tau^c$, there is no open set $G \setminus E$ such that $G \cap \mathbb{Q} = \emptyset$, where G is the closure of $G \setminus E$.

Theorem 3. *For computably locally compact, computably Hausdorff spaces,*

$$\psi^{fiber} \equiv \psi^{dom} \equiv \psi^{sie} \equiv \psi^{un} \equiv \psi^>.$$

Proof. See Appendix.

Theorem 4. *For computably locally compact, computably Hausdorff spaces,*

$$\psi^{range} \leq \psi^{<}.$$

Proof. See Appendix.

In general $\psi^{<} \leq \psi^{range}$ does not hold. See [3] for a counterexample for computable metric spaces.

4.2 Computability on Compact Subsets

In this Section we study several representations of the set $\mathcal{K}$ of the compact subsets of a computably locally compact, computably Hausdorff space.

Definition 8 (representations of compact sets). *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computably locally compact, computably Hausdorff space.*

1. *Define the minimal cover representation $\kappa^{mc} :\subseteq \Sigma^\omega \to \mathcal{K}$ as follows: $K = \kappa^{mc}(p)$ iff*
   - $w \in \mathrm{dom}(\theta^*)$ *if* $\iota(w) \lhd p$,
   - $\{w \in \Sigma^* \mid \iota(w) \lhd p\} = \{w \in \Sigma^* \mid K \subseteq \theta^*(w) \text{ and } (\forall \iota(u) \lhd w)\, \nu(u) \cap K \neq \emptyset\}$.
2. *Define the union representation $\kappa^{un} :\subseteq \Sigma^\omega \to \mathcal{K}$ as follows: $\kappa^{un}\langle p, w\rangle = K$ iff*
   - $p \in \mathrm{dom}(\psi^{un})$ *and* $w \in \mathrm{dom}(\nu^*)$,
   - $K = \psi^{un}(p)$ *and* $K \subseteq \theta^*(w)$.
3. *Define the net representation $\kappa^{net} :\subseteq \Sigma^\omega \to \mathcal{K}$ as follows: $K = \kappa^{net}(p)$ iff*
   - $p = \langle w_1, w_2, \ldots\rangle$ *and* $w_i \in \mathrm{dom}(\nu^*)$ *for all* $i \in \mathbb{N}$,
   - $\overline{\theta^*(w_{i+1})} \subseteq \theta^*(w_i)$ *for all* $i \in \mathbb{N}$,
   - $K = \bigcap_{i=1}^{\infty} \theta^*(w_i)$.

A κ^{mc}-name requires that each listed base element has nonempty intersection with K.

For any compact subset of a second countable locally compact space, there exists a "strictly" decreasing cover sequence converging to it. Since every locally compact space is a Hausdorff space, then the limit of a cover sequence is unique. Therefore, κ^{net} is well-defined. Comparing with the cover representation, the advantage of net representation is that it does not require all finite basic covers.

Theorem 5. *For computably locally compact, computably Hausdorff spaces,*

$$\kappa^{mc} \leq \kappa^{c} \equiv \kappa^{un} \equiv \kappa^{net}.$$

Proof. See Appendix.

Note that $\kappa^c \leq \kappa^{mc}$ is not true in general [22].

5 Conclusion and Future Work

In this article, we mainly generalize the representations of subsets of metric spaces to locally compact spaces and analyze which relations among these representations still hold. For this reason we define and characterize computably locally compactness and computably Hausdorff spaces.

The next step is to include representations of sets of functions and generalize this research to computable T_0-spaces. Moreover, we will apply these representations to examine the effectivity of certain theorems in general topology.

Acknowledgement

The authors would like to thank Klaus Weihrauch for his valuable suggestions and also thank Decheng Ding for his support. This work is supported by NSFC 10420130638 and DFG CHV113/240/0-1.

References

1. Brattka, V.: Recursive quasi-metric spaces. Theoret. Comput. Sci. **305** (2003) 17–42
2. Brattka, V., Hertling, P.: Continuity and computability of relations. Informatik Berichte **164** Fern University in Hagen, Hagen (1994)
3. Brattka, V., Presser, G.: Computability on subsets of metric spaces. Theoret. Comput. Sci. **305** (2003) 43–76
4. Brattka, V., Weihrauch, K.: Computability on subsets of Euclidean space I: Closed and compact subsets. Theoret. Comput. Sci. **219** (1999) 65–93
5. Collins, P.: Continuity and computability on reachable sets. Theoret. Comput. Sci. **341** (2005) 162–195
6. Engelking, R.: General Topology. Heldermann, Berlin (1989)
7. Grzegorczyk, A.: Computable functions. Fund. Math. **42** (1955) 168–202
8. Grubba, T., Weihrauch, K.: A computable version of Dini's theorem for topological spaces. Lecture notes in computer science (2005) 927–936
9. Grubba, T., Weihrauch, K.: On computable metrization. CCA 2006, Third International Conference on Computability and Complexity in Analysis. In: Cenzer, D., Dillhage, R., Grubba, T., Weihrauch, K. (Eds.), Informatik Berichte **333**, Fern University in Hagen, Hagen (2006) 176–191
10. Hauck, J.: Berechenbare reelle funktionen. Z. math. Logik Grundlagen Math. **19** (1973) 121–140
11. Ko, K.I.: Complexity Theory of Real Functions. Birkhaeuser, Boston (1991)
12. Kreitz, C., Weihrauch, K.: A unified approach to constructive and recursive analysis. In: Richter, M., Borger, E., Oberschelp, W., Schinzel, B., Thomas, W. (Eds.), Computation and Proof Theory, Lecture Notes in Mathematics **1104** Springer, Berlin. (1984) 259–278
13. Kusner, B.A.: Lectures on Constructive Mathematical Analysis **60** AMS, Providence (1984)
14. Lacombe, D.: Extension de la notion de fonction recursive aux fonctions d'une ou plusieurs variables reelles I. Comptes Rendus Academie des Sciences Paris **240** (1955) 2478–2480

15. Mazur, S.: Computable Analysis **33** Razprawy Matematyczne, Warsaw (1963)
16. Pour-El, M.B., Richards, J.I.: Computability in Analysis and Physics. Pers. in Math. Logic. Springer, Berlin (1989)
17. Schröder, M.: Effective metrization of regular spaces. In: Ko, K.I., Nerode, A., Pour-El, M.B., Weihrauch, K., Wiedermann, J. (Eds.), Computability and Complexity in Analysis, Informatik Berichte **235**, Fern University in Hagen, Hagen (1998) 63–80
18. Scott, D.: Outline of a mathematical theory of computation. Tech. Mono. PRG-2. Oxford University, Oxford (1970)
19. Stoltenberg-Hansen, V., Tucker, J.V.: Concrete models of computation for topological algebras. Theoret. Comput. Sci. **219** (1999) 347–378
20. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. **42** (1936) 230–265
21. Weihrauch, K.: Computability on computable metric spaces. Theoret. Comput. Sci. **113** (1993) 191–210
22. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)
23. Weihrauch, K.: Multi-functions on multi-represented sets are closed under flowchart programming. CCA 2005, Second International Conference on Computability and Complexity in Analysis. In: Grubba, T., Hertling, P., Tsuiki, H., Weihrauch, K. (Eds.), Informatik Berichte **326**, Fern University in Hagen, Hagen (2005) 267–300
24. Yasugi, M., Mori, T., Tsujii, Y.: Effective properties of sets and functions in metric spaces with computability structure. Theoret. Comput. Sci. **219** (1999) 467–486
25. Zhong, N., Weihrauch, K.: Computability theory of generalized functions. J. Asso. for Comp. Mach **50(4)** (2003) 469–505
26. Zhou, Q.: Computable real-valued functions on recursive open and closed subsets of Euclidean space. Math. Logic Q. **42** (1996) 379–409
27. Ziegler, M.: Computability on regular subsets of Euclidean space. Math. Logic Q. **48(Suppl. 1)** (2002) 157–181
28. Ziegler, M.: Computable operators on regular sets. Math. Logic Q. **50(4, 5)** (2004) 392–404

6 Appendix: Proofs

Proof of Theorem 1:

Proof. Let $\kappa^c(p) = K$, I be the r.e. set for finite intersection defined in (3) and H the r.e. Hausdorff set. From p a sequence r can be computed such that $\iota(w) \lhd r$ iff

$$(\exists \iota(u) \lhd p \text{ listing } u_0 \cdots u_k)\,(\exists v \in \Sigma^* \text{ listing } v_0 \cdots v_k)$$
$$((u_0, v_0), \ldots, (u_k, v_k) \in H \text{ and } (v, w) \in I).$$

Then $\nu(w) \subseteq K^c$ if $\iota(w) \lhd r$. Next we show, that $K^c \subseteq \bigcup_{\iota(w) \lhd r} \nu(w)$ holds. Let $y \in K^c$. For all $x \in K$ there is some $(u, v) \in H$ such that $x \in \nu(u)$ and $y \in \nu(v)$. The familiy of all these $\nu(u)$ covers K. As K is compact it has a finite subcover $\{\nu(u_i) \mid j \in J\}$ covering K. Let v list $\{v_i \mid j \in J\}$. As $y \in \bigcap_{j \in J} \nu(v_j) = \bigcap_{v_j \lhd v} \nu(v_j)$ there is some $w \in \Sigma^*$ such that

$$(v, w) \in I \text{ and } y \in \nu(w)$$

Therefore r is a ψ^{un}-name of K.

Proof of Theorem 3:

Proof. $\psi^{fiber} \leq \psi^{dom}$: (As in [4]:) Let A be a closed subset of X and $\psi^{fiber}(p) = A$, that is, $\eta_p^{\omega\omega}$ is a (δ, ρ)-realization of a function $f_p : X \to \mathbb{R}$ such that $f_p^{-1}\{0\} = A$. Define $g_p : \subseteq X \to \mathbb{N}$ as follows:

$$g_p(x) = \begin{cases} 1 & \text{if } f_p(x) \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

for all $x \in X$. Then $\mathrm{dom}(g_p) = A^c$. As $\eta_p^{\omega\omega}$ is a realization of f_p, we have that, if $\rho\eta_p^{\omega\omega}(q) \neq 0$, $g_p\delta(q) = 1$; otherwise, $g_p\delta(q)$ diverges. By utm-theorem, there is a computable function $G : \subseteq \Sigma^\omega \times \Sigma^\omega \to \Sigma^*$ such that $g_p\delta(q) = \nu_{\mathbb{N}}G(p, q)$. And by smn-theorem for $\eta^{\omega *}$, there is a computable function $H : \Sigma^\omega \to \Sigma^\omega$ such that $g_p\delta(q) = \nu_{\mathbb{N}}\eta^{\omega *}_{H(p)}(q)$. Furthermore, for any $x \notin \mathrm{dom}(g_p)$, we have $f_p(x) = 0$, then $G(p, q)$ diverges. This shows that $\eta^{\omega *}_{H(p)}$ is a strong realization. Therefore, $H(p)$ is a ψ^{dom}-name of A, as required.

$\psi^{dom} \leq \psi^{sie}$: (As in [3]) Let $\psi^{dom}(p) = A$, that is, $\eta^{\omega *}$ is a strong $(\delta, \nu_{\mathbb{N}})$-realization of function $f_p : \subseteq X \to \mathbb{N}$ such that $\mathrm{dom}(f_p) = A^c$. Let M be a Type-2 machine computing the universal function of $\eta^{\omega *}$. Define

$$H(p, q) = \begin{cases} 0^\omega & \text{if } M \text{ does not halt on input } (p, q) \\ 0^k 1^\omega & \text{if } M \text{ halts on input } (p, q) \text{ after } k \text{ steps} \end{cases}$$

Then H is computable and by utm- and smn-theorem for $\eta^{\omega\omega}$ there exists a computable function F such that $\eta^{\omega\omega}_{F(p)}(q) = H(p, q)$. Now we have $\rho^< \eta^{\omega\omega}_{F(p)}(q) = \rho^< H(p, q) = \mathrm{cf}_A\delta(q)$, that is, $\eta^{\omega\omega}_{F(p)}$ is a $(\delta, \rho^<)$-realization of cf_A. Therefore, $F(p)$ is a ψ^{sie}-name of A, as required.

$\psi^{sie} \leq \psi^{un}$: Let $\psi^{sie}(p) = A$ and M be a Type-2 machine computing the universal function of $\eta^{\omega\omega}$. There is a Type-2 machine that on input p computes a sequence r such that $\iota(u) \lhd r$ iff
$(\exists w \in \mathrm{dom}(\nu^*))$ (M on input (p, q) writes some $\langle a \rangle$ such that $\nu_{\mathbb{Q}}(a) > 0$ and M has at most read the prefix w of q) and $(w, u) \in I$, where I is the r.e. set for finite intersection defined in (3).

$\psi^{un} \leq \psi^>$: Note that a ψ^{un}-name of A is just a θ^{un}-name of A^c. Since $\bar{U} \subseteq A^c$ is (ν, θ^{un})-r.e. by Lemma 6, we can now construct a Type-2 machine M, which on input a θ^{un}-name p outputs a list of u such that $\overline{\nu(u)} \subseteq A^c$, that is, $\overline{\nu(u)} \cap A = \emptyset$. Therefore, f_M translates ψ^{un} to $\psi^>$.

$\psi^> \leq \psi^{fiber}$: By Theorem 2 $\mathbf{X}$ is computably regular, if it is computably locally compact and a computably Hausdorff space. Then the multivalued Urysohn operator UR mappping every pair (A, B) of disjoint closed sets to all continuous functions $f : X \to [0; 1]$ such that $f[A] = 0$ and $f[B] = 1$ is $(\psi^>, \psi^>, [\delta \to \rho])$-computable ([9]).

Let $\psi^>(p) = A$ and let p be a list of $\{u_i \in \mathrm{dom}(\nu) \mid i \in \mathbb{N}\}$. There is a Type-2 machine M that on input p

- computes a $\psi^>$-name p_i of $\overline{\nu(u_i)}$ for all $i \in \mathbb{N}$ ($U \to \bar{U}$ is (ν, κ^c)-computable and $\kappa^c \leq \psi^>$),

– computes a $[\delta \to \rho]$-name q_i of some $f_i \in \mathrm{UR}(\psi^>(p), \psi^>(p_i))$ for all $i \in \mathbb{N}$,
– computes a $[\delta \to \rho]$-name q of $f : X \to [0;1]$ defined by

$$f(x) := \sum_{i=0}^{\infty} 2^{-i} f_i(x).$$

Then $A = \psi^{fiber}(q)$.

Proof of Theorem 4:

Proof. Let $\psi^{range}(p) = A$. If $A = \emptyset$ then $p = 0^\omega$ and furthermore $\psi^<(0^\omega) = \emptyset$. If $A \neq \emptyset$ then

$$\begin{aligned} \nu(w) \cap A \neq \emptyset &\iff \text{there exists some } x \in \mathrm{range}([\nu_{\mathbb{N}} \to \delta](p)) \text{ such that } x \in \nu(w) \\ &\iff \text{there exists some } u \in \mathrm{dom}(\nu_{\mathbb{N}}) \text{ such that } w \text{ is listed by } \eta_p(u). \end{aligned}$$

The following Typ-2 machine M realizes the reduction: On input p the machine M copies all zeros on the input tape to the output tape until it reads a symbol $a \neq 0$. Then M writes $\iota(w)$ iff

$$(\exists u \in \mathrm{dom}(\nu_{\mathbb{N}}))\, \iota(w) \lhd u_\eta(p, u).$$

Proof of Theorem 5:

Proof. $\kappa^{mc} \leq \kappa^c$: Let $\kappa^{mc}(p) = K$. As $\mathrm{dom}(\nu^*)$ is r.e., there is a Type-2 machine that on input p computes a sequence $r \in \Sigma^\omega$ such that $\iota(w) \lhd r$ iff

$$(\exists \iota(w') \lhd p)(\exists w'' \in \mathrm{dom}(\nu^*))\, \{u \mid \iota(u) \lhd w\} = \{u \mid \iota(u) \lhd w'\} \cup \{u \mid \iota(u) \lhd w''\}.$$

$\kappa^c \leq \kappa^{un}$: By theorem 1 $\kappa^c \leq \psi^{un}$ for computably Hausdorff spaces. Furthermore if p is a κ^c-name of K then $K \subseteq \theta^*(w)$ for any $\iota(w) \lhd p$.

$\kappa^{un} \leq \kappa^c$: Let $\kappa^{un}\langle p, w\rangle = K$ where w lists $\{v_1, \ldots, v_k\}$. Then $K^c = \psi^{un}(p)$ and $K \subseteq B := \bigcup\{\overline{\nu(v_i)} \mid i = 1, \ldots k\}$ As $\mathbf{X}$ is computably locally compact, there is a Type-2 machine M that on input $(v_1, \ldots, v_k)$ computes a κ^*-name $\langle q_1, \ldots, q_k\rangle \in \Sigma^\omega$ such that $\kappa^c(q_i) = \overline{\nu(v_i)}$ for all $i \in \{1, \ldots, k\}$. By Lemma 2 from $\langle q_1, \ldots, q_k\rangle$ a sequence $q \in \Sigma^\omega$ can be computed with $\kappa^c(q) = B$. By Lemma 3 the mapping $(K, B) \to K \cap B$ is $(\psi^{un}, \kappa^c, \kappa^c)$-computable. As $K = K \cap B$ we have shown $\kappa^{un} \leq \kappa^c$.

$\kappa^{net} \leq \kappa^c$: Let $\kappa^{net}(p) = K$. There is a Type-2 machine that on input p computes a sequence r such that $\iota(w) \lhd r$ iff

$$(\exists \iota(w') \lhd p)\, \overline{\theta^*(w')} \subseteq \theta^*(w).$$

$\kappa^c \leq \kappa^{net}$: Let $\kappa^c(p) = K$, where p is a list of $\{w_i \in \mathrm{dom}(\theta^*) \mid i \in \mathbb{N}\}$ and F the $(\kappa^c, \theta^{un}, \theta^*)$-computable mapping from Lemma 7 with

$$U \in \mathrm{F}(K, O) :\iff K \subseteq U \subseteq \bar{U} \subseteq O.$$

There is a Type-2 machine that on input p

– starts with output $\iota(w_0)$.
– if $\iota(w'_0)\iota(w'_1)\ldots\iota(w'_i)$ has been written on the output tape, then M writes some $\iota(w'_{i+1})$ such that

$$\theta^*(w'_{i+1}) \in \mathrm{F}(\kappa^c(p), \theta^*(w'_i) \cap \theta^*(w_{i+1})).$$

A New Approach to Graph Recognition and Applications to Distance-Hereditary Graphs*

Shin-ichi Nakano[1], Ryuhei Uehara[2], and Takeaki Uno[3]

[1] Department of Computer Science, Faculty of Engineering, Gunma University, Gunma 376-8515, Japan
nakano@cs.gunma-u.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan
uehara@jaist.ac.jp
[3] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. Distance-hereditary graphs consist of the isometric graphs, and hence contain trees and cographs. First, a canonical and compact tree representation of the class is proposed. The tree representation can be constructed in linear time using two prefix trees. Usually, the prefix trees are used to maintain a set of strings. The prefix trees in our algorithm are used to maintain the neighbors for each vertex, which is new approach comparing to the other known results based on the lexicographically bread first search. Several efficient algorithms for the distance-hereditary graphs based on the canonical tree representation are proposed; linear time algorithms for graph recognition and graph isomorphism, and efficient enumeration algorithm. An efficient coding for the tree representation is also presented, which requires $4n$ bits for a distance-hereditary graph of n vertices, and $3n$ bits for a cograph. The results improve previously known upper bounds of the number of distance-hereditary graphs and cographs.

Keywords: algorithmic graph theory, cograph, distance-hereditary graph, prefix tree, tree representation.

1 Introduction

Recently, data-driven computations are studied in, e.g., data mining and bioinformatics. We process over tons of data, find knowledge automatically, and classify them in these areas. To achieve the purpose efficiently, we sometimes assume a structure of the data, which is observed implicitly or explicitly. More precisely, we propose a possible structure for the data, enumerate them, and test if the model is feasible. The frequent pattern discovery problem in data mining is a typical example, and widely investigated (see, e.g., [16,19,29,1]). Once the feasible model is fixed, we move our attention to solve the problem over the structured data. However, those structures are relatively primitive from the graph algorithmic point of view, and there are many unsolved problems for more complex structure.

* This work was partially done while the second and third authors were visiting ETH Zürich, Switzerland.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 115–127, 2007.
© Springer-Verlag Berlin Heidelberg 2007

We have to achieve three efficiencies to deal with the complex structure efficiently; the structure has to be represented efficiently, essentially different instances have to be enumerated efficiently, and the property of the structure has to be checked efficiently. There are few results from this viewpoint except trees [22,23,25,13,20].

Recently, a variety of graph classes have been proposed and studied so far [4,14]. Since an early work by Rose, Tarjan, and Lueker [27], the lexicographic breadth first search (LexBFS) plays an important role as a basic tool to recognize several graph classes. The LexBFS gives us a simple ordering of vertices based on the neighbor-preferred manner (see [9] for further details).

In this paper, instead of LexBFS, we use a prefix tree, which represents a set of strings and supports to check whether a given string is included or not in the set. The notion is also known as trie [21]. We define a string to represent the open and closed neighbors, which will be defined later, and maintain those strings in two prefix trees. Using two prefix trees we can find a set of vertices having identical (or similar) neighbors efficiently. This is a quite different approach to the previously known algorithms based on LexBFS [11,5].

We apply the idea to distance-hereditary graphs. Distance in graphs is one of the most important topics in algorithmic graph theory, and there are many areas that have some geometric property. Distance-hereditary graphs were characterized by Howorka [17] to deal with the geometric distance property called *isometric*, which means that all induced paths between pairs of vertices have the same length. More precisely, a distance-hereditary graph is the graph that any pair of vertices u and v has the same distance on any vertex induced subgraph containing u and v. Intuitively, any longer path between them has a shortcut on any vertex induced subgraph.

Some characterizations of distance-hereditary graphs are given [2,12,15]. Especially, Bandelt & Mulder [2] showed that any distance-hereditary graph can be obtained from K_2 by a sequence of operations called "adding a pendant vertex" and "splitting a vertex." Many efficient algorithms on distance-hereditary graphs are based on the characterization [7,3,6,26,18,8]. We will show that the sequence can be found efficiently on the prefix trees that represent open and closed neighbors of each vertex.

In this paper, there are two key contributions for the class of distance-hereditary graphs. First, we propose a compact and canonical tree representation. This is a natural generalization for the tree representation of the class of cographs. Secondly, we show a linear time and space algorithm that constructs the tree representation for any distance-hereditary graph. To achieve the linear time and space, two prefix trees for open and close neighbors play important roles. The results have the following applications.

(1) The graph isomorphism problem can be solved in linear time and space. This is conjectured by Spinrad in [28, p.309], but it was not explicitly given.

(2) The recognition problem can be solved in linear time and space. Our algorithm is much simpler than the previously known recognition algorithm for the class (see [11, Chapter 4] for further details); the original Hammer & Maffray's algorithm [15] fails in some cases, and Damiand, Habib, and Paul's algorithm [11] requires to build the cotree of a cograph in linear time. The cotree of a cograph can be constructed in linear time by using a classic algorithm due to Corneil, Perl, and Stewart [10], or a recent algorithm based on multisweep LexBFS approach by Bretscher, Corneil, Habib, and Paul [5].

(3) For given n, all distance-hereditary graphs with at most n vertices can be enumerated in $O(n)$ time per graph with $O(n^2)$ space.

(4) We propose an efficient encoding of a distance-hereditary graph. Any distance-hereditary graph with n vertices can be represented in at most $4n$ bits. This encoding gives us a upper bound of the number of distance-hereditary graphs of n vertices: there are at most 2^{4n} non-isomorphic distance-hereditary graphs with n vertices. Applying the technique to cographs, each cograph of n vertices can be represented in at most $3n$ bits, and hence the number of cographs of n vertices is at most 2^{3n}. They improve the previously known upper bounds; both are $2^{O(n \log n)}$ [28, p.20,p.98].

Due to space limitation, proofs and some details of algorithms are omitted.

2 Preliminaries

The set of the *open neighbors* of a vertex v in a graph $G = (V, E)$ is the set $N(v) = \{u \in V \mid \{u, v\} \in E\}$. We denote the *closed neighbors* $N(v) \cup \{v\}$ by $N[v]$. Throughout the paper, we denote by $n := |V|$ and $m := |E|$. For a vertex subset U of V, we denote by $N(U)$ the set $\{v \in V \mid v \in N(u)$ for some $u \in U\}$, and by $N[U]$ the set $\{v \in V \mid v \in N[u]$ for some $u \in U\}$. A vertex set C is a *clique* if all pairs of vertices in C are joined by an edge in G. If a graph $G = (V, E)$ itself is a clique, it is said to be *complete*, and denoted by K_n. Given a graph $G = (V, E)$ and a subset U of V, the *induced subgraph* by U, denoted by $G[U]$, is the graph (U, E'), where $E' = \{\{u, v\} \mid u, v \in U$ and $\{u, v\} \in E\}$. For a vertex w, we sometimes denote the graph $G[V \setminus \{w\}]$ by $G - w$ for short. Two vertices u and v are said to be *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. For twins u and v, we say that u is a *strong sibling* of v if $\{u, v\} \in E$, and a *weak sibling* if $\{u, v\} \notin E$. We also say *strong (weak) twins* if they are strong (weak) siblings. If a vertex v is one of the strong or weak twins with another vertex u, we simply say that they are twins, and v has a sibling u. Since twins make the transitive relation, we say that a vertex set S with $|S| > 2$ is also strong (weak) twins if each pair in S consists of strong (weak) twins. For two vertices u and v, the *distance* of the vertices, denoted by $d(u, v)$, is the minimum length of the paths joining u and v. We here extend the neighbors recursively as follows: For a vertex v in G, we define $N_0(v) := \{v\}$ and $N_1(v) := N(v)$. For $k > 1$, $N_k(v) := N(N_{k-1}(v)) \setminus (\cup_{i=0}^{k-1} N_i(v))$. That is, $N_k(v)$ is the set of vertices of distance k from v.

The following operations for a graph play an important role; (α) pick a vertex x in G and add a new vertex x' with an edge $\{x, x'\}$, (β) pick a vertex x in G and add x' with edges $\{x, x'\}$ and $\{x', y\}$ for all $y \in N(x)$, and (γ) pick a vertex x in G and add x' with edges $\{x', y\}$ for all $y \in N(x)$. For the operation (α), we say that the new graph is obtained by attaching a *pendant vertex* x' to the *neck vertex* x. In (β) and (γ), it is easy to see that x and x' are strong and weak twins, respectively. In the cases, we say that the new graph is obtained by *splitting* the vertex x into strong and weak twins, respectively. It is well known that the class of *cographs* is characterized by the operations as follows:

Theorem 1. *A connected graph G with at least two vertices is a cograph iff G can be obtained from K_2 by a sequence of operations (β) and (γ).*

The operations are also used by Bandelt and Mulder to characterize the class of *distance-hereditary graphs* [2]:

Theorem 2. *A connected graph G with at least two vertices is distance-hereditary iff G can be obtained from K_2 by a sequence of operations* (α)*,* (β)*, and* (γ)*.*

In (α), a pendant vertex is attached, and in (β) and (γ), a vertex is split into two siblings. We also use the following generalized operations for $k \geq 1$; (α') pick a neck x in G and add k pendants to x, (β') pick a vertex x in G and split it into $k+1$ strong siblings, and (γ') pick a vertex x in G and split it into $k+1$ weak siblings.

For a vertex set $S \subseteq V$ of $G = (V, E)$ and a vertex $s \in S$, the *contraction* of S into s is obtained by: (1) for each edge $\{v, u\}$ with $v \in S \setminus \{s\}$ and $u \in V \setminus S$, add an edge $\{u, s\}$ into E. (2) Multiple edges are replaced by a single edge. (3) remove all vertices in $S \setminus \{s\}$ from V and their associated edges from E.

Let $v_1, v_2, \ldots, v_n$ be an ordering of a vertex set V of a connected distance-hereditary graph $G = (V, E)$. Let G_i denote the graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ for each i. Then the ordering is *pruning sequence* if G_{n-1} is K_2 and G_{i+1} is obtained from G_i by either pruning a pendant vertex v_i or contracting some twins v_i and $v \in G_{i+1}$ into v for each $i < n-1$. For a connected cograph G, the pruning sequence of G is defined similarly with only contractions of twins.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* iff there is a one-to-one mapping $\phi : V \to V'$ such that $\{u, v\} \in E$ iff $\{\phi(u), \phi(v)\} \in E'$ for every pair of vertices $u, v \in V$. We denote by $G \sim G'$ if they are isomorphic.

2.1 Open and Closed Prefix Trees and Basic Operation

We here introduce the prefix trees, which are also called tries, of open and closed neighbors. The details can be found in standard textbooks; see, e.g., [21]. A *prefix tree* is a rooted tree T; hereafter, the vertices of prefix trees are called *nodes* to distinguish from the vertices in G. Except the root, each node in T is labeled by a vertex in G, and some nodes are linked to vertices in G. We note that two or more nodes in T can have the same label (the name of a vertex of G), but each vertex in G has exactly one pointer to a node in T. The labels of a prefix tree satisfy; (1) if a node with label v is the parent of a node of label v', it holds $v' > v$, and (2) no two children of a node have the same label. We will maintain $N(v)$ and $N[v] = N(v) \cup \{v\}$ for all vertices in G by two prefix trees as follows. The path of a prefix tree from a node x to the root gives a set of labels (or vertex set in G), denoted by $set(x)$. If the node x is pointed by a vertex v, then we consider that $set(x)$ is associated with v. In this manner, two families of open/closed neighbors are represented by two prefix trees, by considering the neighbor set as the associated set. We call them *the open/closed prefix trees*, and denoted by $T(G)$ and $T[G]$, respectively. The prefix trees $T(G)$ and $T[G]$ for the graph G in Fig. 1 are depicted in Fig. 2. Except the root, each square is the node labeled by a vertex in G. Each circle indicates a vertex v in G, and the thick arrows are pointers to the corresponding node. Since we can make that every leaf of the prefix tree is pointed by at least one vertex, the size of prefix tree is $O(n+m)$. With suitable data structure, we can get the parent of a node in constant time, but to find a specified child (by a label) takes linear time in the number of children.

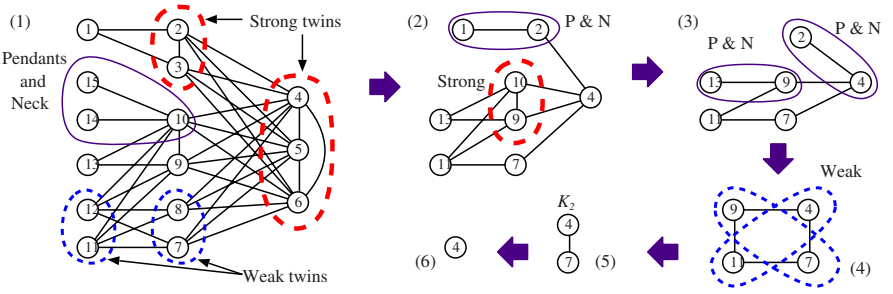


Fig. 1. A distance-hereditary graph and its contracting/pruning process

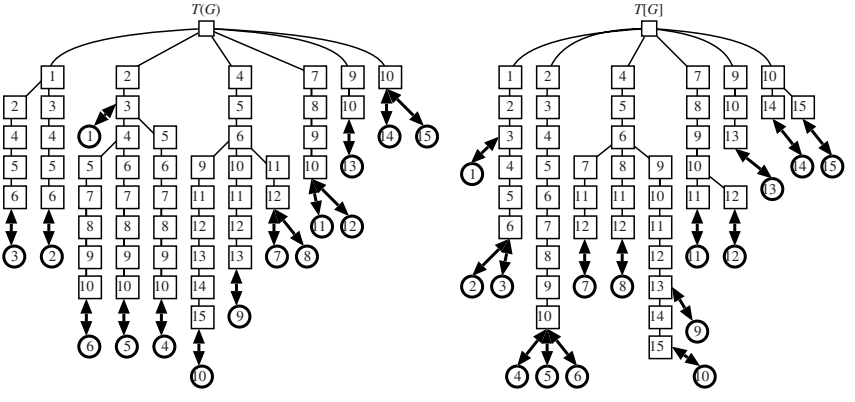


Fig. 2. Two prefix trees for G in Fig. 1(1)

Lemma 1. *For any given graph $G = (V, E)$, $T(G)$ and $T[G]$ are constructed in $O(n+m)$ time and space.*

Observation 3. *Let u and v be any vertices in G. Then (1) u is pendant of the neck v iff a child of the root of $T(G)$ has label v and pointed by u, (2) u and v are strong twins iff u and v point to the same node in $T[G]$, and (3) u and v are weak twins iff u and v point to the same node in $T(G)$.*

During the algorithm, we have to update prefix trees efficiently. Especially, removing a node from a prefix tree is a key procedure, which is described in Algorithm 1.

Lemma 2. *For a graph $G = (V, E)$ and a vertex w, $T(G-w)$ and $T[G-w]$ are obtained from $T(G)$ and $T[G]$, respectively, by Algorithm 1.*

A node x pointed by the vertex w has an ancestor with label v iff $w \in N(v)$ or $w \in N[v]$. Thus the number of ancestors of the node pointed by w is at most $|N[w]|$, and hence steps 1 and 2 can be done in $O(|N[w]|)$ time. Moreover, the number of nodes with label w is bounded by $|N[w]|$, and the sum of the number of their children is also bounded by $|N[w]|$ since a node of label w appears only on the path from the root to the node

Algorithm 1. Delete a vertex from a prefix tree

Input : An (open or closed) prefix tree T, vertex w to be deleted;
Output: Prefix tree T' which does not contain w;

```
1  let x be the node linked to the vertex w; // Remove N[w] in steps 1, 2.
2  while x is a leaf pointed by no vertices do delete x and set x to be its parent;
3  foreach node x with label w do
       // Remove all ws from N(v) or N[v]
4      attach the list of pointers from vertices to x to the list of the parent y of x;
5      connect each child of x to y as a child of y;
6      while y has two children z1, z2 with same label do
7          unify z1 and z2 to a node z, and replace z1 and z2 by z;
8          update y by z;
9      delete x from T;
10 return T.
```

pointed by $v \in N[w]$. Hence the loop from steps 3 to 9 will be repeated at most $|N[w]|$ times. Steps 4 and 9 can be done in $O(1)$ time. Therefore, our main task is to perform step 5 and the while loop from step 6 efficiently. We call the step 7 *unification* of z_1 and z_2. The following lemma is a general property of a prefix tree.

Lemma 3. *Suppose that Algorithm 1 deletes all vertices from $G = (V, E)$. Then the total number of unifications is $O(n + m)$.* □

For each y, the *unification cost* of y is time complexity for finding the pair z_1 and z_2. We will show that the unification cost of each y can be $O(1)$.

3 Canonical Trees

We introduce the notion of the DH-tree, which is a canonical tree representation of a distance-
hereditary graph. First, we define the DH-tree derived from a distance-hereditary graph G. The derivation is constructive, and the resultant tree is uniquely determined up to isomorphism. But it can be redundant. Hence we next introduce the normalized DH-tree obtained by reducing the redundancy which is also uniquely determined up to isomorphism. Hence it will be used as the canonical and compact representation of a distance-hereditary graph.

We will deal with K_1, K_2 as special distance-hereditary graphs. Hereafter, we assume that $G = (V, E)$ is a connected distance-hereditary graph with at least three vertices. For given G, we define three families of vertex sets as follows;

$$\mathcal{S} := \{S \mid x, y \in S \text{ iff } N[x] = N[y] \text{ and } |S| \geq 2\},$$

$$\mathcal{W} := \{W \mid x, y \in W \text{ iff } N(x) = N(y), |W| \geq 2, \{x, y\} \notin E, \text{ and } |N(x)| = |N(y)| > 1\}, \text{ and}$$

$$\mathcal{P} := \{P \mid x, y \in P \text{ iff } x \text{ is a pendant vertex and } y \text{ is its neck}\}.$$

Lemma 4. *(1) Each set $P \in \mathcal{P}$ contains exactly one neck with associated pendants. (2) Each $v \in V$ belongs to either (a) exactly one set in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, or (b) no set in the families. (3) For any distance-hereditary graph G, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$.*

We first introduce the notion of the DH-*tree derived from a distance-hereditary graph* G, which is a rooted ordered tree, and each inner node[1] has a label. The label of an inner node is one of $\{\mathtt{s}, \mathtt{w}, \mathtt{p}\}$, which indicate strong/weak twin, and pendant with neck, respectively. The DH-tree derived from G is defined recursively from leaves to the root. We have three basic cases:

(1) The DH-tree derived from K_1 is defined by a single root with no label.

(2) When $G \sim K_n$ with $n \geq 2$, the DH-tree of G is defined by a single root with label $\mathtt{s}$ and n leaves with no labels. The tree represents that K_n can be obtained from a single vertex K_1 by splitting it into n strong siblings.

(3) The graph G is a *star* with $n > 2$ vertices which has a center vertex with $n - 1$ pendant vertices. In the case, the DH-tree derived from G is defined by a single root with label $\mathtt{p}$, and n leaves with no labels. We remind that the tree is ordered. In the tree, the leftmost child of a node with label $\mathtt{p}$ indicates the neck. That is, the leftmost leaf corresponds to the center of the star, and $n - 1$ leaves correspond to the $n - 1$ pendants.

We note that K_2 is a clique, and not a star. Hence the root with label $\mathtt{p}$ of a DH-tree has at least three children. We also note that the number of leaves of the tree is the number of vertices in G. This is an invariant for the DH-tree.

We now define the DH-tree $\mathcal{T}$ derived from $G = (V, E)$ with $|V| = n > 2$ in general case. We assume that G is neither K_n nor a star. We start with independent n leaves of $\mathcal{T}$. Each leaf in $\mathcal{T}$ corresponds to a vertex in G, and we identify them hereafter. Then, by Lemma 4(2), we can group the leaves by three kinds of families $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$. Then, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$ by Lemma 4(3). Let S be any set in $\mathcal{S}$. Then we make a common parent with label $\mathtt{s}$ of the leaves. We repeat this process for each S in $\mathcal{S}$. For each set W in $\mathcal{W}$, we similarly make a common parent with label $\mathtt{w}$ of them. Let P be any set in $\mathcal{P}$. Then, P contains exactly one neck v and some pendants u. We make a common parent with label $\mathtt{p}$ of them, and make the neck v the leftmost child of the parent. The pendants are placed on the right of the neck in arbitrary ordering.

After the process, we contract each vertex set in $\mathcal{S} \cup \mathcal{W}$ into a new vertex on G. Each new vertex corresponds to a parent in the resultant $\mathcal{T}$ and we identify them. For each $P \in \mathcal{P}$, we also prune all pendant vertices except the neck in P. The neck corresponds to the parent of the nodes in P. We repeat the process until the resultant graph becomes one of the basic cases, and we obtain the DH-tree $\mathcal{T}$ derived from $G = (V, E)$.

An example of the DH-tree derived from G in Fig. 1 is depicted in Fig. 3. The contraction of twins is described by removing all siblings except the smallest one. Each node in the DH-tree corresponds to a vertex in the graph in Fig. 3.

The DH-tree derived from a distance-hereditary graph can be redundant: As a rule (β) can be replaced by (β'), if a node with label "$\mathtt{w}$" is the parent of the other nodes with label "$\mathtt{w}$," they can be weak siblings (in Fig. 4, the case (1) can be replaced by (2)). The same reduction can be applied to the nodes with label "$\mathtt{s}$".

Hence we can introduce the notion of the *normalized* DH-tree of a distance-hereditary graph G, which is obtained from the DH-tree derived from G by applying the reduction as possible as we can. The reduction can be done by the standard depth

[1] We will use the notation "node" for a DH-tree to distinguish from a "vertex" in G.

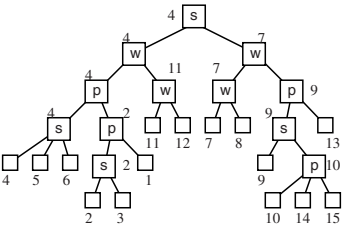


Fig. 3. DH-tree $\mathcal{T}$ derived from the graph in Fig. 1(1)

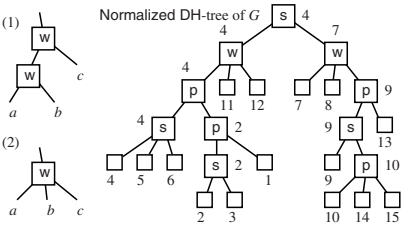


Fig. 4. Reduction rule, and the compact DH-tree

first search. Hence, the normalized DH-tree $\mathcal{T}$ of G can be obtained from the DH-tree $\mathcal{T}'$ derived from G in $O(|\mathcal{T}|) = O(|\mathcal{T}'|) = O(n)$ time and space.

Theorem 4. *The normalized* DH*-tree of a connected distance-hereditary graph is canonical. That is, the normalized* DH*-tree T for a connected distance-hereditary graph G is uniquely determined, and the original distance-hereditary graph G is uniquely constructed from the normalized* DH*-tree T.*

Corollary 1. *The normalized* DH*-tree $\mathcal{T}$ for a distance-hereditary graph $G = (V, E)$ requires $O(|V|)$ space.* □

Note: By Theorems 1 and 2, the normalized DH-tree for a cograph only consists of the nodes of labels s and w. The same notion for a cograph appears in many literature as the *cotree*, e.g., [5,10,11]. Since only the nodes of label p require the ordering of children, cotree of a cograph is the rooted non-ordered tree.

4 Linear Time Construction of Canonical Trees

In this section, we give a linear time algorithm for constructing the DH-tree of a distance-hereditary graph. From Lemma 4(2) and Observation 3, a set $W \in \mathcal{W}$ (resp., $S \in \mathcal{S}$) corresponds to a set of vertices pointing to the same node in $T(G)$ (resp., $T[G]$). From Lemma 4(1), a set $P \in \mathcal{P}$ contains one neck v. Then, by Lemma 4(1), (2), and Observation 3, the set P corresponds to a node x of depth 1 in $T(G)$ such that (1) x has label v, (2) x is pointed by at least one vertex in G, and (3) all vertex in $P \setminus \{v\}$ points to x. The outline of the construction of the canonical tree for a distance-hereditary graph is; (1) construct the open and closed prefix trees, (2) if $G \sim K_n$ or G is a star, complete $\mathcal{T}$ and halt, (3) produce nodes of $\mathcal{T}$ for vertices in $\mathcal{P} \cup \mathcal{S} \cup \mathcal{W}$, (4) contract twins and prune pendants with updates of prefix trees $T(G)$ and $T[G]$, and go to (2). Now we fix the families $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{W}$. Let w be a vertex which will be removed from G since it is one of twins or pendant. Hereafter, we assume that w is the largest vertex among them. We have two cases.

Twin: We first assume that w is the largest twin that has a sibling w' with $w' < w$.

Lemma 5. *Let x be any node of prefix tree with label w, and y the parent of x. Then x is the largest node among the children of y.*

By Lemma 5, no unification occurs in the while loop in Algorithm 1, since all children of x are larger than any other child of y. Thus we have the following:

Theorem 5. *The prefix trees $T(G - w)$ and $T[G - w]$ can be obtained from $T(G)$ and $T[G]$ in $O(|N(w)|)$ time, respectively.*

Pendant: Next we assume that w is a pendant. For the maintenance of a pendant vertex in a distance-hereditary graph, we introduce a technical vertex ordering, called *levelwise laminar ordering* (LL-ordering, for short). We here introduce notations $N_v^+(u)$, $N_v^-(u)$, and $N_v^0(u)$ as follows. Let v be a vertex in $G = (V, E)$. Then $N_v^+(u) := N_{d(u,v)+1}(v) \cap N(v)$, $N_v^-(u) := N_{d(u,v)-1}(v) \cap N(v)$, and $N_v^0(u) := N_{d(u,v)}(v) \cap N(v)$. We define $N_u^-(u) := \emptyset$. Due to space limitation, we only state here the properties of the LL-ordering below:

(L1) For any $v_i \in N_k(r)$ and $v_j \in N_{k'}(r)$, $i < j$ holds if $0 \leq k < k'$.
(L2) For any $v_i, v_j \in N_k(r), k \geq 1$, $i < j$ holds if $N_r^-(v_j) \subset N_r^-(v_i)$.
(L3) Let v_i, v_j be any vertices in $N_k(r), k \geq 1, i < j$ such that $N_r^-(v_i) = N_r^-(v_j)$.
Then we have $N_r^-(v_i) = N_r^-(v_j) = N_r^-(v)$ for all vertices $v_i < v < v_j$.

The LL-ordering is a partial order weaker than the order by LexBFS. Thus our algorithm runs under weaker assumption than the previous results in [2,11]. If the graph G is a distance-hereditary graph, G has the LL-ordering, and it can be computed in linear time. We also can remove a pendant or contract twins without violating the LL-ordering.

We are ready to remove the largest pendant w. Let w' be the neck of w. Vertices are ordered in the LL-ordering $(r = v_1, v_2, \ldots, v_n)$ from the root r in V. Then we have two subcases. First case is the special case that $w = r$; in the case, w is the only pendant, since there are no other pendant larger than w. Due to lack of space, we omit the case, and we now assume that we aim to remove the pendant w that is not the root.

Lemma 6. *Let w be the largest pendant in $N_i(r)$ with $i \geq 1$ (for fixed $\mathcal{P}$). We delete w from $T(G)$ and $T[G]$ by Algorithm 1. Then the unification cost is always $O(1)$.*

Theorem 6. *If G is a distance-hereditary graph, the* DH*-tree $\mathcal{T}$ derived from G can be constructed in $O(|V| + |E|)$ time and space.*

5 Applications

Hereafter, we assume that given graph $G = (V, E)$ is a distance-hereditary graph.

Theorem 7. *(1) The recognition problem for distance-hereditary graphs can be solved in $O(n+m)$ time and space. (2) The graph isomorphism problem for distance-hereditary graphs can be solved in $O(n + m)$ time and space.*

By the characterizations in [2], we have the following:

Corollary 2. *For the following graph classes, we have the same results in Theorem 7: cographs, bipartite distance-hereditary graphs.*

It is worth to remarking that our algorithm modified for a cotree is quite simple, since we have no pendant vertices.

Enumeration: For given n, we efficiently enumerate each distance-hereditary graph with at most n vertices exactly once as follows; (1) enumerate all unordered trees of n leaves such that each inner node has at least two children,(2) for each tree obtained in (1), enumerate all possible assignments of labels to all inner nodes, and (3) for each label assignment in (2), enumerate all possible choices of one child as a neck for each node with label p.Using the technique in [24], we have the following.

Theorem 8. *Distance-hereditary graphs with at most n vertices can be enumerated in $O(n)$ time for each, with $O(n^2)$ space.*

Compact encoding: We design an efficient encoding scheme for distance-hereditary graphs. Our scheme encodes each distance-hereditary graphs G into only (at most) $4n$ bits in $O(m + n)$ time. Also one can decode from the string to G in $O(m + n)$ time.

Given G, one can construct its normalized DH-tree $\mathcal{T}$ in $O(m + n)$ time. The number of leaves in $\mathcal{T}$ is n. Let n_i be the number of inner nodes in $\mathcal{T}$. Since each inner node has two or more children, $n_i \leq n - 1$ holds.

We first encode $\mathcal{T}$ into a string S_1 with ignoring labels, then encode the labels into a string S_2. The resulting string $S_1 + S_2$ has enough information to reconstruct $\mathcal{T}$ and so does G.

Given a normalized (ordered) DH-tree $\mathcal{T}$ we traverse $\mathcal{T}$ starting at the root with depth first manner. If we go down an edge of $\mathcal{T}$ then we code it with 0, and if we go up an edge then we code it with 1. Thus we need two bits for each edge in $\mathcal{T}$. The length of the resulting bit string is $2(n + n_i - 1) \leq 4n - 4$ bits. For instance, the bit string for the tree in Fig. 4 is

$$0000101011000101101110101100101000100101011101111$$

We can save n_i bits from the string above as follows. For each inner node v, after traversing v's first child and its descendant with depth first manner, we return back to v again then always go "down" to v's second child. Note that each inner node has two or more children. Thus we can omit this "down" to its second child for each inner node. In the following bit string those $n_i = 10$ bits are underlined.

$$0000\underline{10}1011\underline{00}0\underline{10}11\underline{10}1111\underline{0}1011\underline{00}1\underline{01}0\underline{00}0\underline{10}0\underline{10}1011\underline{10}111$$

Thus we need only $2(n + n_i - 1) - n_i \leq 3n - 3$ bits. Let S_1 be the resulting bit string.

Then we encode the labels of $\mathcal{T}$ as follows. Note that each inner node has one label among {s, w, p}, and each leaf has no label. We are going to store those labels in preorder with one bit for each label.

Let v be an inner node of $\mathcal{T}$ except for the root. Let u be the parent node of v. We show that if the label of u is given then the label of v has only two choices. By properties of the DH-tree, if the label of u is s, then the label of v is not s. Similarly, if the label of u is w, then the label of v is not w. If the label of u is p, we have two subcases. If v is the leftmost child of u, then the label of v is not p, otherwise the label of v is not w. (Note

that two or more neighbors are needed for weak twins.) Thus in any case the label of node v has only two choices.

Also the label of the root is either `s` or `p` since we assume that given graph is connected. Thus we can encode the label of each inner node with only one bit in preorder. The detail is as follows.

If the label of the root is `s` then we encode it with 0, otherwise the label is `p` and we encode it with 1. For each inner node v except for the root we have the following three cases. Let u be the parent node of v.

Case 1: The label of u is `s`. If the label of v is `w` then we encode it with 0, otherwise the label is `p` and we encode it with 1.

Case 2: The label of u is `w`. If the label of v is `p` then we encode it with 0, otherwise the label is `s` and we encode it with 1.

Case 3: The label of u is `p`. We have two subcases.

Case 3(a): v is the leftmost child of u. If the label of v is `s` then we encode it with 0, otherwise the label is `w` and we encode it with 1.

Case 3(b): v is not the leftmost child of u. If the label of v is `s` then we encode it with 0, otherwise the label is `p` and we encode it with 1.

In this way we can encode the label of each inner node with only one bit. By concatenating those bits in preorder we can encode the labels of inner nodes into a bit string of $n_i \leq n - 1$ bits. Let S_2 be the resulting string.

Thus we have encoded a distance-hereditary graph into a string $S_1 + S_2$ with $2(n + n_i - 1) \leq 4n - 4$ bits. Now we have the following Theorem and Corollary.

Theorem 9. *A distance-hereditary graph G can be represented in $4n$ bits.*

Corollary 3. *The number of distance-hereditary graphs of n vertices is at most 2^{4n}.*

Using a simpler case analysis, we also have the following corollary.

Corollary 4. *A cograph G can be represented in $3n$ bits, and the number of cographs of n vertices is at most 2^{3n}.*

References

1. T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering Frequent Substructures in Large Unordered Trees. In *Discovery Science (DS '03)*, pages 47–61. Lecture Notes in Artificial Intelligence Vol. 2843, Springer-Verlag, 2003.
2. H.-J. Bandelt and H.M. Mulder. Distance-Hereditary Graphs. *Journal of Combinatorial Theory, Series B*, 41:182–208, 1986.
3. A. Brandstädt and F.F. Dragan. A Linear-Time Algorithm for Connected r-Domination and Steiner Tree on Distance-Hereditary Graphs. *Networks*, 31:177–182, 1998.
4. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
5. A. Bretscher, D. Corneil, M. Habib, and C. Paul. A Simple Linear Time LexBFS Cograph Recognition Algorithm. In *Graph-Theoretic Concepts in Computer Science (WG 2003)*, pages 119–130. Lecture Notes in Computer Science Vol. 2880, Springer-Verlag, 2003.

6. H.J. Broersma, E. Dahlhaus, and T. Kloks. A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. *Discrete Applied Mathematics*, 99:367–400, 2000.
7. M.-S. Chang, S.-Y. Hsieh, and G.-H. Chen. Dynamic Programming on Distance-Hereditary Graphs. In *Proceedings of 8th International Symposium on Algorithms and Computation (ISAAC '97)*, pages 344–353. Lecture Notes in Computer Science Vol. 1350, Springer-Verlag, 1997.
8. M.-S. Chang, S.-C. Wu, G.J. Chang, and H.-G. Yeh. Domination in distance-hereditary graphs. *Discrete Applied Mathematics*, 116:103–113, 2002.
9. D.G. Corneil. Lexicographic Breadth First Search — A Survey. In *Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 1–19. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2004.
10. D.G. Corneil, Y. Perl, and L.K. Stewart. A Linear Recognition Algorithm for Cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.
11. G. Damiand, M. Habib, and C. Paul. A Simple Paradigm for Graph Recognition: Application to Cographs and Distance Hereditary Graphs. *Theoretical Computer Science*, 263:99–111, 2001.
12. A. D'Atri and M. Moscarini. Distance-Hereditary Graphs, Steiner Trees, and Connected Domination. *SIAM Journal on Computing*, 17(3):521–538, 1988.
13. R. Geary, N. Rahman, R. Raman, and V. Raman. A Simple Optimal Representation for Balanced Parentheses. In *Symposium on Combinatorial Pattern Matching (CPM)*, pages 159–172. Lecture Notes in Computer Science Vol. 3109, Springer-Verlag, 2004.
14. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.
15. P.L. Hammer and F. Maffray. Completely Separable Graphs. *Discrete Applied Mathematics*, 27:85–99, 1990.
16. L. B. Holder, D. J. Cook, and S. Djoko. Substructure Discovery in the SUBDUE System. In *Workshop on Knowledge Discovery in Databases*, pages 169–180. AAAI Workshop on Knowledge Discovery in Databases, AAAI, 1994.
17. E. Howorka. A Characterization of Distance-Hereditary Graphs. *Quart. J. Math. Oxford (2)*, 28:417–420, 1977.
18. S.-Y. Hsieh, C.-W. Ho, T.-S. Hsu, and M.-T. Ko. Efficient Algorithms for the Hamiltonian Problem on Distance-Hereditary Graphs. In *COCOON 2002*, pages 77–86. Lecture Notes in Computer Science Vol. 2387, Springer-Verlag, 2002.
19. A. Inokuchi, T. Washio, and H. Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 13–23. Lecture Notes in Computer Science Vol. 1910, Springer-Verlag, 2000.
20. D. E. Knuth. *Generating All Trees*, volume 4 of *The Art of Computer Programming*. Addison-Wesley, fascicle 4 edition, 2005.
21. D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, 2nd edition, 1998.
22. J. I. Munro and V. Raman. Succinct Representation of Balanced Parentheses, Static Trees and Planar graphs. In *Proc. 38th ACM Symp. on the Theory of Computing*, pages 118–126. ACM, 1997.
23. J. I. Munro and V. Raman. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM Journal on Computing*, 31:762–776, 2001.
24. S. Nakano and T. Uno. Constant Time Generation of Trees with Specified Diameter. In *Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 33–45. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2005.
25. S.-I. Nakano. Efficient Generation of Plane Trees. *Information Processing Letters*, 84(3):167–172, 2002.

26. F. Nicolai and T. Szymczak. Homogeneous Sets and Domination: A Linear Time Algorithm for Distance-Hereditary Graphs. *Networks*, 37(3):117–128, 2001.
27. D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
28. J.P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.
29. M. J. Zaki. Efficiently Mining Frequent Trees in a Forest. In *8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, ACM Press, 2002.

Finding a Duplicate and a Missing Item in a Stream

Jun Tarui

Department of Info and Comm Eng
University of Electro-Comm
Chofu, Tokyo 182-8585 Japan
tarui@ice.uec.ac.jp

Abstract. We consider the following problem in a stream model: Given a sequence $a = \langle a_1, a_2, \ldots, a_m \rangle$ wich each $a_i \in [n] = \{1, \ldots, n\}$ and $m > n$, find a *duplicate* in the sequence, i.e., find some $d = a_i = a_l$ with $i \neq l$ by using limited s bits of memory and r passes over the input sequence. In one *pass* an algorithm reads the input sequence a in the order $a_1, a_2, \ldots, a_m$. Since $m > n$, a duplicate exists by the pigeonhole principle. Muthukrishnan [Mu05a], [Mu05b] has posed the following question for the case where $m = n + 1$: For $s = O(\log n)$, is there a solution with a *constant* number of passes? We have described the problem generalizing Muthukrishnan's question by taking the sequence length m as a parameter. We give a negative answer to the original question by showing the following: Assume that $m = n + 1$. A streaming algorithm with $O(\log n)$ space requires $\Omega(\log n / \log \log n)$ passes; a k-pass streaming algorithm requires $\Omega(n^{1/(2k-1)})$ space. We also consider the following problem of *finding a missing item*: Assuming that $n < m$, find $x \in [m]$ such that $x \neq a_j$ for $1 \leq j \leq n$. The same lower bound applies for the missing-item finding problem. The proof is a simple reduction to the communication complexity of a *relation*. We also consider *one*-pass algorithms and exactly determine the minimum space required. Interesting open questions such as the following remain. For the number of passes of algorithms using $O(\log n)$ space, show an $\omega(1)$ lower bound (or an $O(1)$ upper bound) for: (1) duplicate finding for $m = 2n$, (2) missing-item finding for $m = 2n$, and (3) the case where we allow Las-Vegas type randomization for $m = n + 1$.

Keywords: Data stream algorithm, communication complexity, finding duplicate, finding missing item, pigeonhole principle, proof complexity.

1 Introduction

The duplicate finding problem as originally posed by Muthukrishnan [Mu05a], [Mu05b, Section 1.3: Pointer and Chaser] or as more generally described in the abstract is interesting by its simplicity in addition to its nature as a basic data stream problem. As we wil see, it also leads to interesting questions in communication complexity.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 128–135, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The missing-item finding problem is in a sense dual to the duplicate-finding problem. Muthukrishnan [Mu05b, Section 1.1: Finding Missing Numbers] discusses in detail an easier version where all items in a sequence are guaranteed to be distinct. Our consideration will show how the problem gets harder when this guarantee is dropped: We will see that our arguments for the duplicate finding problem also applies for the missing-item finding problem. But we note that there does *not* seem to be a general transformation of an algorithm for one problem into an algorithm for the other.

In a stream model, processing massive data in *one* pass is important for applications where data sequentially arrives one after another, e.g., packets passing through a router. Processing with a small number of *multiple* passes is important, e.g., for database applications where massive data has been stored in such a way that sequential access can be efficiently performed but random access is slower. For more information about the data stream model of computation, see a survey by Muthukrishnan [Mu05a], which includes more explanations of motivations and an extensive list of references.

Finding a duplicate and a missing item respectively corresponds to finding a *crowded pigeonhole* and a *missing pigeon* in the language of the pigeonhole principle. We have not yet really investigated a connection, if any, of our results and questions to the *proof complexity* of the pigeonhole principle. We only give some remarks in the final section. We do follow the convention of using index i for pigeons and j for pigeonholes.

2 Simple Algorithms

We explain some simple algorithms for duplicate finding; being aware of them will be useful for us.

1. Assume that $m = n+1$. With *random access* as opposed to streaming access, the following finds a duplicate with time $O(n)$ and space $O(\log n)$. Consider $a_1, \ldots, a_{n+1}$ as *pointers* to some index $i \in [n]$, and thus consider the sequence a as a linked list. Starting with a_{n+1} traverse the list with *two* pointers p_1 and p_2. In each step, the pointer p_1 advances following one pointer hop and the pointer p_2 advances following *two* pointer hops. The list must contain a loop and the two pointers collide somewhere inside the loop; whey they collide a duplicate has been found.

2. Assume that $m = n+1$. The following is a two-pass streaming algorithm that uses space $O(\sqrt{n}\log n)$. Identify $[n] \cong [\sqrt{n}] \times [\sqrt{n}]$. In the first pass, for each $l \in [\sqrt{n}]$ count the number of a_i's such that $a_i \in l \times [\sqrt{n}]$ using $\sqrt{n}$ counters; thus find $l \in [\sqrt{n}]$ such that $l \times [\sqrt{n}]$ contains a duplicate. In the second pass find a duplicate using $\sqrt{n}$-bit vectors.

By identifying instead $[n] \cong [\sqrt{n/\log n}] \times [\sqrt{n \log n}]$, one can reduce the space from $O(\sqrt{n}\log n)$ to $O(\sqrt{n \log n})$. Similarly, with $k \geq 2$ passes, one can find a duplicate with space $O(n^{1/k}\,(\log n)^{(k-1)/k})$.

3. Assume that $m = n+1$. In the first pass, count the number of items in $[n/2]$. If the number exceeds $n/2$, then there is a duplicate in $[n/2]$, and otherwise in $[n] - [n/2]$. The binary search finds a duplicate with $\log_2 n$ passes and space $O(\log n)$.

4. Assume that $m = 2n$. Randomly choose an index $i \in [m]$. In one pass, "ignore" all $a_1, \ldots, a_{i-1}$, "remember" $d = a_i$, and check if d occurs among $a_{i+1}, \ldots, a_m$; if so, declare that a duplicate d is found, or else declare a failure. The algorithm fails only when a_i is the last occurrence of $d = a_i$ in the sequence. Each $j \in [n]$ has at most one last occurrence, and hence the algorithm succeeds with probability at least $(m-n)/m = 1/2$. The algorithm uses one pass and $O(\log n)$ space. Note that it is a Las-Vegas-type randomized algorithm in the sense that it never reports a non-duplicate as a duplicate.

5. Let $m = n+1$. Assume that there is a promise that the input sequence contains precisely one duplicate, or equivalently, $\{a_1, \ldots, a_{n+1}\} = [n]$. Then one can find a duplicate in one pass and with space $O(\log n)$ by summing a_i's.

3 Limitations of Multiple-Pass Streaming Algorithms

We give a negative answer to Muthukrishnan's original question by showing the following.

Theorem 1. Assume that $m = n+1$. A streaming algorithm with $O(\log n)$ space requires $\Omega(\log n/\log\log n)$ passes. A k-pass streaming algorithm requires $\Omega(n^{1/(2k-1)})$ space.

Proof. Assume that n is odd; similar arguments apply for the case when n is even. Consider the following transformation into a communication game. We consider only a restricted set of input sequences such that the first $(n+1)/2$ items are all distinct and the last $(n+1)/2$ items are all distinct. Give Alice the set $A = \{a_1, \ldots, a_{(n+1)/2}\}$ and give Bob the set $B = \{a_{(n+1)/2+1}, \ldots, a_{n+1}\}$; the task for Alice and Bob is to agree on some $x \in (A \cap B)$. An algorithm that finds a duplicate using s bits of memory and r passes yields a $2r$-round protocol for the game with at most s communication bits per round. But the game above is the monotone Karchmer-Wigderson communication game for Majority ([KW92]; also explained in [KN97] and [BS90]). In the game Alice and Bob are respectively given a maxterm (or a 1-input x) and a minterm (or a 0-input y) with respect to Majority, and they have to find some l lying in the intersection of the maxterm and the minterm (or find some index l such that $x_l = 1$ and $y_l = 0$). An $r(n)$-round protocol with $s = O(\log n)$ communication bits per round for the game is equivalent to a monotone circuit for Majority with depth $2r(n)$ and fan-in at most $2^{s(n)} = \mathrm{poly}(n)$. Boppana [Bo86] has shown that such a circuit must have depth at least $\Omega(\log n/\log\log n)$. Håstad [Hå87] gave the same lower bound for *non*monotone circuits computing Majority. (Explanations can also explained

in, e.g., [BS90].) The space lower bound for k-pass algorithms also follows from the size lower bounds by Boppana and Håstad for depth-k circuits with unbounded fan-in AND/OR gates. □

Remark 1. For $x \in \{0,1\}^n$, let $p(x)$ denote the fraction of i's with $x_i = 1$; i.e., $p(x) = |\{i : x_i = 1\}|/n$. Consider the following computation: Given $x \in \{0,1\}^n$ with the promise that either (1) $p(x) \geq 1/2 + \epsilon(n)$ or (2) $p(x) \leq 1/2 - \epsilon(n)$, output 1 in case (1) and 0 in case (2). Ajtai and Ben-Or have shown that for $\epsilon(n) = \log^{-O(1)} n$, such computations can be done by depth-$O(1)$ polynomial-size monotone circuits ([AB84]; also explained in [BS90]). Thus if we write $m = (1 + \epsilon(n))n$, our argument above cannnot produce an $\omega(1)$ lower bound for the number of passes for the case where $\epsilon(n) \geq \log^{-O(1)}(n)$.

Applying essentially the same argument readily yields the following.

Propostion 1. The bounds in Theorem 1 apply for streaming algorithms for finding a missing item for the case where $m = n + 1$.

Proof. Give Alice and Bob respectively the complement, with respect to $[m]$, of the set of the first and the last $(m-1)/2$ items. An algorithm for finding a missing item yields a protocol solving the same task as in the proof of Theorem 1. □

4 The Minimum Space Required for One-Pass Algorithms

Now we consider the space complexity of one-pass algorithms. We measure the space complexity in a detailed way: We consider the minimum required number of *states* of algorithms.

First consider the duplicate finding problem. For the one pass case, it is meaningful to distinguish two kinds of tasks: (1) Find a duplicate d. (2) Find a duplicate d *and* some witnessing indices, i.e., some pair $i \neq l$ such that $a_i = a_l = d$. A natural model for the one pass case is an automaton with one way access to the input. But we give a lower bound in a more general model of an n-way read-once branching program [BC82][RWY02]: an internal node labeled by index i reads a_i and branches out to at most n nodes depending on the value of a_i; each sink node corresponds to an answer, i.e., a duplicate d for Task 1, and d together with $i \neq l$ satisfying $a_i = a_l = d$ for Task 2.

For any $m > n$, we have the following natural automata for the two tasks. For Task 1, remember the set of the integers a_i that have appeared so far; for Task 2, remember the set of the pairs (i, a_i) that have appeared so far. If the next input item is one that has already appeared, declare that a duplicate has been found. The number of the *internal* nodes in the corresponding branching programs are respectively 2^n and $1 + \Sigma_{k=1}^{n} n(n-1)\cdots(n-k+1)$.

Razborov, Wigderson, and Yao [RWY02] have given an asymptotically tight lower bound for Task 2: Read-once branching programs solving Task 2 must have size $2^{\Omega(n \log n)}$. Their argument is by considering a adversarial Markov chain defined in terms of the branching program that it tries to fool.

By a simple adversary argument that does *not* involve any probability distribution, we exactly determine the one-pass space complexity of Task 1. One cannot do any better than the automaton given above:

Theorem 2. For any $m > n$ (including $m = \infty$), if P is a read-once branching program solving Task 1, then the number of non-sink nodes in P is at least 2^n.

Proof. Let P be a read-once branching program solving Task 2. For each internal node u in P, define the set $K(u) \subseteq [n]$ to be the set of the items that node u has "seen for sure"; that is, define $K(u) \subseteq [n]$ to be the set of items $k \in [n]$ satisfying the following condition: for every path w in P from the source (a unique initial node) to node u there exists an edge in w that is labeled by "$a_l = k$" for some l. Note that if a program P correctly solves the problem, then whenever it declares d as a duplicate in node u, it holds that $d \in K(u)$. Also note that if $e = (u, v)$ is a directed edge labeled by $a_l = k$, i.e., a program at node u reads a_l and follows e if $a_l = k$, then $K(v) \subseteq (K(u) \cup \{k\})$.

We claim that the family $\{K(u) : u \text{ is an internal node in } P\}$ equals the family of all subsets of $[n]$. To see the claim, assume, for the sake of contradiction, that a set $S \subseteq [n]$ does *not* appear as $K(u)$ for any node u. For the source node u_0 in P, $K(u_0) = \emptyset$, and hence the set S must be nonempty. Consider inputs produced by the following adversary: Starting from the initial state, when the value of a_i is queried at node u, fix it to be any element in the nonempty set $S - K(u)$. When inputs are determined this way, a program remains in nodes u such that $K(u)$ is a proper subset of S, and never produces an answer; a contradiction. □

Similar, even simpler arguments yield the following.

Propostion 2 For $m = n+1$, if P is a read-once branching program solving the missing-item finding problem, then the number of nodes in P is at least $2^m - 1$.

5 One-Pass Las-Vegas Randomized Algorithms

Theorem 3. Assume that $m = n + 1$. Let P be a Las-Vegas-type randomized read-once oblivious branching program that finds a duplicate with probability at least half, where one fixed ordering of reading inputs applies for all the executions of the program. Then the number of nodes in P is at least $2^{n/4-o(n)}$.

Proof. The proof is by considering the distributional complexity of a deterministic algorithm and using the min-max principle for a two-person game; this method is a common one in communication complexity: The lower bound for distributional complexity is a lower bound for randomized complexity.

Assume that $m = n + 1$ is even. Let A_n be the set of sequences $a = \langle a_1, \ldots, a_{n+1} \rangle$ with $a_i \in [n]$ such that (1) $\{a_1, \ldots, a_{n+1}\} = [n]$, i.e., each $j \in [n]$ appears,

and that (2) a unique duplicate in a appears once in $\{a_1, \ldots, a_{(n+1)/2}\}$ and once in $\{a_{(n+1)/2+1}, \ldots, a_{n+1}\}$. Let α_n be the uniform distribution on A_n.

Let P be a deterministic oblivious branching program that either finds a duplicate or reports a failure. Let $\epsilon > 0$ be an arbitrary real number. We show that if the number of nodes in P is at most $2^{(1/4-2\epsilon)n}$, then the probability with respect to α_n that P fails is at least $1/2 + \epsilon$ for sufficiently large n. Note that this yields the asserion of the theorem. Assume that P has at most $2^{(1/4-2\epsilon)n}$ nodes and assume that n is large enough so that $2^{-\epsilon n} \leq \epsilon$.

Let v be a node in P such that P reaches v after reading the first half from some $a \in A_n$, i.e., afer reading some $a_1, \ldots, a_{(n+1)/2}$ such that $a_i \in [n]$ and $a_i \neq a_l$ for $1 \leq i < l \leq (n+1)/2$. Consider the set $K(v)$, where $K(v)$ is defined as in the proof of Theorem 2. It is not hard to see that the following holds.

$$\Pr_\alpha[P \text{ reaches } v] \leq 2^{-(|K(v)|-1)}.$$

Since P has at most $2^{(1/4-2\epsilon)n}$ nodes, the probability that after reading the items in the first half the program P reaches some node v such that $|K(v)| \geq (1/4-\epsilon)n + 1$ is at most

$$2^{(1/4-2\epsilon)n} \cdot 2^{-(1/4-\epsilon)n} = 2^{-\epsilon n} \leq \epsilon,$$

where the last inequality is our assumption.

Consider the case where after reading $a_1, \ldots, a_{(n+1)/2}$ the program P reaches a node v such that $|K(v)| \leq (1/4-\epsilon)n$. The program P is not allowed to make any error when it reports a duplicate. Thus a duplicate that P can report after visiting node v must be an element of $K(v)$. But with respect to distribution α_n each a_i in $\{a_1, \ldots, a_{(n+1)/2}\}$ is a unique duplicate with equal probability. Thus the probability that P fails conditioned upon the event that P reaches v is at least

$$\frac{(1/2)(n+1) - (1/4-\epsilon)n}{(1/2)(n+1)} > 1 - \frac{(1/4-\epsilon)n}{(1/2)n} = 1 - (1/2 - 2\epsilon) = 1/2 + 2\epsilon.$$

Thus the probability with respect to α_n that P fails is at least $1/2 + \epsilon$. □

Remark 2. The algorithm 5 in Section 2 finds a duplicate for *all* the sequences in A_n above, but it does produce an error when applied for arbitrary inputs.

Remark 3. The lower bound for space in Theorem 3 applies for randomized one-way communication complexity of the Karchmer-Wigderson game. This can be seen fairly directly from the proof above: For each possible message v from Alice to Bob, let $K(v)$ be the set of $x \in [n]$ such that there is some input sequnce that makes Bob output x as an answer after receiving v; then $K(v)$ has the same meaning as in the proof above.

6 Open Problems and Discussions

There are interesting open problems that remain. First note the trivial fact that the duplicate finding problem and the missing-item finding problem does not get harder as m gets bigger and as n gets smaller.

Open Problem 1. Consider a duplicate finding streaming algorithm that uses $O(\log n)$ space. Show an $\omega(1)$ or better lower bound for the number of passes for bigger m, e.g., for $m = 2n$; we do not see the way how allowing bigger m makes the problem any easier. Similarly consider and show limitations for the missing-item finding problem for smaller n, e.g., $n = m/2$.

Open Problem 2. Consider the duplicate finding problem when we allow randomization. In particular, show an $\omega(1)$ or better lower bound for the number of passes for the case where $m = n + 1$.

A naturally corresponding question in terms of communication complexity is the following: Allow randomization in the Karchmer-Wigderson protocol described above and show limitations. Following the key work of Alon, Matias, and Szegedy [AMS96] communication complexity arguments have often been used for showing limitations of streaming algorithms including ones that use randomization. These arguments usually focus on the communication complexity of computing a *function* that is related to the problem considered. What seems different for the problems described above is the fact that we are naturally led to consider the randomized communication complexity of a *relation* in the Karchmer-Wigderson framework. We also note that without the k-round restriction there is a deterministic protocol with $O(\log n)$-bit communication that solves the Karchmer-Wigderson game for Majority.

As mentioned in the introduction, a connection, if any, of this work to the proof complexity of the pigeonhole principle has not been worked out. One translation of Theorem 1 seems to be the following. Consider the following form of resolution refutation proofs of the pigeonhole principle. Let x_{ij}'s be mn binary variables representing whether pigeon i is in pigeonhole j. Split these variables in two halves between Alice and Bob. In one *round*, Alice or Bob can perform an arbitrary number of resolutions free-of-charge on those variables that she/he owns, and writes down some formulas on a blackboard for the other party to continue the next round. How much is written on the blackboard is the proof size in this model. It can not be small since any resolution refutation proof can be converted to an algorithm that, given an assignment, searchs for an unsatisfied clause, which can be set to be a crowded pigeonhole or a missing pigeon.

Acknowledgements

The author would like to thank Muthu Muthukrishnan for stimulating discussions. He would also like to thank Takeshi Tokuyama for stimulating discussions and pointing out that a small number of multiple passes are important in database applications.

References

[AB84] M. Ajtai and M. Ben-Or. A Theorem on Probabilistic Constant Depth Circuits, *Proc. of STOC84*, pp. 471–474, 1984.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments, *Proc. of STOC97*, pp. 20–29, 1996.

[Bo86] R. Boppana. Threshold Functions and Bounded Depth Monotone Circuits. *Journal of Computer and System Sciences*, vol. 32, pp. 222-229, 1986.

[BS90] R. Boppana and M. Sipser. The Complexity of Finite Functions, in: J. van Leeuwen, ed. *Handbook of Theoretical Computer Science vol. A*, MIT Press, 1990.

[BC82] A. Borodin and S. Cook. A Time-Space Trade-Off for Sorting on a General Sequential Model of Computation. *SIAM Journal on Computing*, vol. 11, pp. 287–297, 1982.

[Hå87] J. Håstad. *Computational Limits for Small-Depth Circuits*, MIT Press, 1987.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*, Cambridge University Press, 1997.

[KW92] M. Karchmer and A. Wigderson. Monotone Circuits for Connectivity Requires Super-Logarithmic Depth, *SIAM Journal on Discrete Mathematics*, vol. 5, no. 4, pp. 545–557, 1992.

[Mu05a] S. Muthukrishnan. *talk* given at the Kyoto Workshop on New Horizons in Computing, March, 2005.

[Mu05b] S. Muthukrishnan. Data Streams: Algorithms and Applications, *Foundations and Trends in Theoretical Computer Science*, volume 1, issue 2, 2005; a preliminary version available at `http://www.cs.rutgers.edu/~muthu`.

[RWY02] A. Razborov, A. Wigderson, and A. Yao. Read-Once Branching Programs, Rectangular Proofs of the Pigeon-Hole Principle and the Transversal Calculus, *Combinatorica*, vol. 22, no. 4, pp. 555–574, 2002; also in *Proc. of STOC97*, pp. 739–748, 1997.

Directed Searching Digraphs: Monotonicity and Complexity

Boting Yang[1] and Yi Cao[2]

[1] Department of Computer Science, University of Regina
boting@cs.uregina.ca
[2] Department of Computing Science, University of Alberta
cao1@cs.ualberta.ca

Abstract. In this paper, we introduce and study two new search models on digraphs: the directed searching and mixed directed searching. In these two search models, both searchers and intruders must follow the edge directions when they move along edges. We prove the monotonicity of both search models, and we show that both directed and mixed directed search problems are NP-complete.

1 Introduction

Many real-world problems can be naturally modeled by graph search problems. Examples include: capturing intruders in a building, clearing a complex system of interconnected pipes which is contaminated by some noxious gas, and killing a computer virus in a network system. The meaning of a cleared or contaminated edge varies with the problems. For example, in the problem of capturing intruders, a cleared edge means that there is no intruder hiding along this edge, while a contaminated edge means that there may be some intruders hiding along this edge.

In general, a *graph* or *digraph search problem* is to find the minimum number of searchers required to capture all the intruders hiding in a graph or digraph. In the edge search problem introduced in [9], there are three types of actions for searchers, i.e., placing, removing and sliding, and an edge is cleared only by a sliding action in a proper way. In the node search problem introduced in [7], there are only two types of actions for searchers, i.e., placing and removing, and an edge is cleared if both end vertices are occupied by searchers. Kirousis and Papadimitriou [7] showed that the node search number is equal to the pathwidth plus one. Bienstock and Seymour [4] introduced the mixed search problem that combines the edge search and node search problems. Thus, in the mixed search problem, an edge is cleared if both end vertices are occupied by searchers or cleared by a sliding action in a proper way. In these three graph search problems, intruders are invisible and they can move along a path that contains no searchers at a great speed at any time. Seymour and Thomas [13] introduced another variant of the graph search problem in which an intruder hiding in the graph is visible to searchers. They showed that the search number of this variant is equal to the treewidth plus one.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 136–147, 2007.
© Springer-Verlag Berlin Heidelberg 2007

When studying search problems from a computational complexity viewpoint, we are interested in deciding the search number of a graph. Megiddo et al. [9] showed that the edge search problem is NP-hard. This problem belongs to the NP class follows from the monotonicity result of [8] in which LaPaugh showed that recontamination of edges cannot reduce the number of searchers needed to clear a graph. Monotonicity is a very important issue in graph search problems. Bienstock and Seymour [4] proposed a method that gives a succinct proof for the monotonicity of the mixed search problem, which implies the monotonicity of the edge search problem and the node search problem. Fomin and Thilikos [5] provided a general framework that can unify monotonicity results in a unique minmax theorem.

An undirected graph is not always sufficient in representing all the information of a real-world problem. For example, directed edges are required if the graph models one-way streets in a road system. Johnson et al. [6] generalized the concepts of tree-decomposition and treewidth to digraphs and introduced a digraph search problem accordingly. Reed [11] defined another treewidth on digraphs. Safari [12] introduced a new parameter of digraphs, d-width, which is related to the directed treewidth of a digraph. Barat [2] generalized the cops-and-robber game to digraphs. He proved that an optimal monotonic search strategy for a digraph needs at most one more searcher than the search number of the digraph and he conjectured that the monotonicity is held for the cops-and-robber game on digraphs. Yang and Cao [14] introduced the strong searching model in which the intruder must follow the edge directions but searchers need not when they move along edges. Yang and Cao [15] also introduced the weak searching model in which searchers must follow the edge directions but the intruder need not when they move along edges. In [16] Yang and Cao introduced the directed vertex separation and investigated the relations between different digraph searching models, directed vertex separation, and directed pathwidth. In the digraph searching models in [14,15], there are three types of actions for searchers: placing, removing and sliding. Alspach et al. [1] proposed four digraph search models in which searchers cannot be removed from digraphs.

Throughout this paper, we use D to denote a digraph, (u, v) to denote a directed edge with tail u and head v, and $u \rightsquigarrow v$ to denote a directed path from u to v. All graphs and digraphs in this paper contain at least one edge.

A natural generalization of the edge search model is the directed search model. In the *directed search model*, both searchers and intruders must move in the edge directions. Initially, all edges of D are *contaminated*. Each intruder can move from vertex u to vertex v along a directed path $u \rightsquigarrow v$ that contains no searchers at a great speed at any time. There are three types of actions for searchers: (1) placing a searcher on a vertex, (2) removing a searcher from a vertex, and (3) sliding a searcher along an edge from its tail to its head. A *directed search strategy* is a sequence of actions such that the final action leaves all edges of D *uncontaminated* (or *cleared*). A contaminated edge (u, v) can be cleared in one of two ways by a sliding action: (1) sliding a searcher from u to v along (u, v) while at least one searcher is located on u and (2) sliding a searcher from u to v along

(u, v) while all edges with head u are already cleared. The digraph D is *cleared* if all of the edges are cleared. The minimum number of searchers needed to clear D in the directed search model is the *directed search number* of D, denoted by $\mathrm{ds}(D)$.

A generalization of the mixed search model is the mixed directed search model, which can be considered as a kind of combination of the directed search model and node search model. We will give a precise definition of this search model in the next section.

In the directed or mixed directed search model, a cleared edge will be recontaminated if there is a directed path from the head of a contaminated edge to the tail of this cleared edge such that there is no searcher stationing on any vertex of this path.

We say that a vertex in D is *occupied* at some moment if at least one searcher is located on this vertex at this moment. Any searcher that is not on D at some moment is said *free* at this moment.

In the directed or mixed directed search model, let S be a search strategy and let A_i be the set of cleared edges immediately after the ith action. S is *monotonic* if $A_i \subseteq A_{i+1}$ for each i. We say that this search model has the property of *monotonicity* (or is *monotonic*) if for any digraph D, there exists a monotonic search strategy that can clear D using k searchers, where k is the search number of D in this search model.

This paper is organized as follows. In Section 2, we prove the monotonicity of the mixed directed search model. In Section 3, we prove the monotonicity of the directed search model. In Section 4, we show the NP-completeness results for both directed and mixed directed search problems. Finally, we conclude this paper in Section 5.

2 Monotonicity of the Mixed Directed Search Model

We will show the monotonicity of the mixed directed search model in this section, which means that recontamination does not help to reduce the mixed directed search number of a digraph. We will extend the method proposed by Bienstock and Seymour [4]. We first give the definition of a critical vertex.

Definition 1. Let D be a digraph. For an edge set $X \subseteq E(D)$, a vertex in $V(D)$ is *critical* if it is the tail of an edge in X and the head of an edge in $E(D) - X$. The set of all critical vertices in $V(D)$ is denoted by $\delta(X)$.

We then define the campaign and the progressive campaign.

Definition 2. Let D be a digraph. A *campaign* in D is a sequence $(X_0, X_1, \ldots, X_n)$ of subsets of $E(D)$ such that $X_0 = \emptyset$, $X_n = E(D)$ and $|X_i - X_{i-1}| \leq 1$, for $1 \leq i \leq n$. The *width* of the campaign is defined as $\max_{0 \leq i \leq n} |\delta(X_i)|$. A campaign is *progressive* if $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_n$ and $|X_i - X_{i-1}| = 1$, for $1 \leq i \leq n$.

Similar to [2] and [4], we have the following lemma for the progressive campaign.

Lemma 1. *If there is a campaign in D of width at most k, then there is a progressive campaign in D of width at most k.*

In the remainder of this section, we will prove that the mixed directed search problem is monotonic. The *mixed directed search model* can be obtained by modifying the directed search model as follows. Recall that there are two ways to clear an edge by a sliding action in the directed search model. In the mixed directed search model, we replace the first way by the *node-search-clearing* rule: an edge can be cleared if both of its end vertices are occupied. Another modification is to disallow the recontamination caused by a sliding action, that is, the action of sliding a searcher from u to v along edge (u, v) changes the state of edge (u, v) from contaminated to clear, but it does not change the state of any other edge. Thus, in a mixed directed search strategy, each sliding along an edge must clear this edge if it is contaminated. More precisely, we define the four types of actions in the mixed directed search model as follows.

Definition 3. Let $S = (s_1, s_2, \ldots, s_n)$ be a mixed directed search strategy for a digraph D. For $1 \leq i \leq n$, let A_i be the set of cleared edges and Z_i be the set of occupied vertices immediately after action s_i such that $\delta(A_i) \subseteq Z_i$. Let $A_0 = Z_0 = \emptyset$. Each action s_i, $1 \leq i \leq n$, is one of the following four types:

(a) (*placing a searcher on v*) $Z_i = Z_{i-1} \cup \{v\}$ for some vertex $v \in V(D) - Z_{i-1}$ and $A_i = A_{i-1}$ (note that each vertex in Z_i has exactly one searcher);
(b) (*removing the searcher from v*) $Z_i = Z_{i-1} - \{v\}$ for some vertex $v \in Z_{i-1}$ and $A_i = \{e \in A_{i-1}$: if there is a directed path $u \rightsquigarrow w$ containing e and an edge $e' \in E(D) - A_{i-1}$ such that w is the head of e and u is the tail of e', then $u \rightsquigarrow w$ has an internal vertex in $Z_i\}$;
(c) (*node-search-clearing e*) $Z_i = Z_{i-1}$ and $A_i = A_{i-1} \cup \{e\}$ for some edge $e = (u, v) \in E(D)$ with both ends u and v in Z_{i-1};
(d) (*edge-search-clearing e*) $Z_i = (Z_{i-1} - \{u\}) \cup \{v\}$ and $A_i = A_{i-1} \cup \{e\}$ for some edge $e = (u, v) \in E(D)$ with $u \in Z_{i-1}$ and $v \in V(D) - Z_{i-1}$ and every (possibly 0) edge with head u belongs to A_{i-1}.

From Definition 3, we know that at most one edge can be cleared in one action and each vertex is occupied by at most one searcher at any time. Note that recontamination in the mixed directed search problem is caused only by removing actions. In (c) and (d), if $e \in A_{i-1}$, then we say this action is *superfluous*. Adding or deleting superfluous actions will not affect the number of searchers used in a search strategy, however, sometimes allowing superfluous actions may make arguments simple.

The *mixed directed search number* of a digraph D, denoted by $\mathrm{mds}(D)$, is the minimum number of searchers needed to clear D in the mixed directed search model. The following lemma reveals the relationship between the mixed directed searching of D and a campaign in D.

Lemma 2. *Let D be a digraph without any multiple edges. If $\mathrm{mds}(D) \leq k$, then there is a campaign in D of width at most $k - 1$.*

Proof. Let $S = (s_1, s_2, \ldots, s_m)$ be a mixed directed search strategy of D using at most k searchers. For $1 \leq i \leq m$, let A_i be the set of cleared edges and Z_i be the set of occupied vertices immediately after s_i, and let $A_0 = Z_0 = \emptyset$. We first normalize S such that the normalized search strategy can also clear D using at most k searchers. The normalized search strategy may contain some superfluous actions, but this will not increase the number of searchers required to clear D.

The normalization is conducted by inserting some node-search-clearing actions after each placing action and edge-search-clearing action. Specifically, for each $1 \leq i \leq m$, if $Z_i - Z_{i-1}$ is empty, i.e., s_i is a removing or node-search-clearing action, then we leave s_i unchanged; otherwise, $Z_i - Z_{i-1}$ contains a vertex v, i.e., s_i is a placing or edge-search-clearing action such that v is occupied, then let $E_v^1 = \{(u, v) \in E(D)\colon u \in Z_{i-1}\}$, $E_v^2 = \{(v, u) \in E(D)\colon u \in Z_{i-1}$ and all edges with head u except (v, u) are in $A_{i-1}\}$, and $E_v^3 = \{(v, u) \in E(D)\colon u \in Z_{i-1}$ and $(v, u) \notin E_v^2\}$, and we then insert a subsequence of node-search-clearing actions between s_i and s_{i+1}, such that each edge in E_v^1 is cleared first, then each edge in E_v^2, and finally each edge in E_v^3 (edges in the same set are cleared in an arbitrary order). After the normalization, we obtain a new sequence of actions that contains each old action and some new node-search-clearing actions. It is easy to see that this new sequence of actions, denoted by $(s'_1, s'_2, \ldots, s'_n)$, is still a mixed directed search strategy of D using at most k searchers.

For $1 \leq i \leq n$, let A'_i be the set of cleared edges and Z'_i be the set of occupied vertices immediately after s'_i, and let $A'_0 = Z'_0 = \emptyset$. Since $\delta(A'_i) \subseteq Z'_i$, $|Z'_i| \leq k$, and $|A'_i - A'_{i-1}| \leq 1$, $1 \leq i \leq n$, we know that $(A'_0, A'_1, \ldots, A'_n)$ is a campaign in D of width at most k.

We now show that the campaign $(A'_0, A'_1, \ldots, A'_n)$ can be converted into a campaign $(X_0, X_1, \ldots, X_n)$ of width at most $k-1$. For each i from 0 to n, if $|\delta(A'_i)| \leq k-1$, then let $X_i = A'_i$. If $|\delta(A'_i)| = k$, then $\delta(A'_i) = Z'_i$. Let v be the last vertex occupied by a searcher in Z'_i. Recall that just after v receives a searcher in a placing or edge-search-clearing action, the following actions clear all edges in E_v^1, E_v^2 and E_v^3 by node-search-clearing. Note that at the step when an edge $(u, v) \in E_v^1$ is cleared, v is not a critical vertex at this step. When an edge $(v, u) \in E_v^2$ is cleared, since D has no multiple edges, all edges with head u are cleared and thus u is not a critical vertex. Hence, when $|\delta(A'_i)| = k$, s'_i must be a node-search-clearing action that clears an edge in E_v^3. Therefore, each vertex in Z'_i has at least one contaminated edge with tail not in Z'_i. Let s'_j be the first removing action after s'_i. Such an action must exist; otherwise, D will not be cleared because each vertex in Z'_i has at least one contaminated edge with tail not in Z'_i. Let $R = A'_{j-1} - A'_j$ and $X_p = A'_p - R$ for $i \leq p \leq j$. Since $|A'_p - A'_{p-1}| \leq 1$, $i \leq p \leq j$, we know that $|X_p - X_{p-1}| \leq 1$ for $1 \leq p \leq j$. Suppose that s'_j removes the searcher on w. Then all edges with tail w must be contaminated immediately after s'_j, which means that A'_j contains no edges with tail w. Hence, X_p contains no edges with tail w for $i \leq p \leq j$. Thus $w \notin \delta(X_p)$ and $|\delta(X_p)| \leq k-1$ for $i \leq p \leq j$. We then consider A'_{j+1} and construct X_{j+1}. We can continue this process and finally we obtain a campaign $(X_0, X_1, \ldots, X_n)$ in D of width at most $k-1$.

Lemma 3. *For a digraph D, if $(X_0, X_1, \ldots, X_n)$ is a progressive campaign in D of width at most $k-1$, then there is a monotonic mixed directed search strategy that clears D using at most k searchers such that the edges of D are cleared in the order $e_1, e_2, \ldots, e_n$, where $e_i = X_i - X_{i-1}$, $1 \leq i \leq n$.*

Proof. We construct the monotonic mixed directed search strategy inductively. Suppose that we have cleared the edges $e_1, \ldots, e_{j-1}$, $2 \leq j \leq n$, in order, and that no other edges have been cleared yet. Let $e_j = (u, v)$ and $C_{j-1} = \{p \in V(D)$: p has no in-edge or all in-edges of p belong to $X_{j-1}\}$. Before we clear (u, v), we may remove searchers such that each vertex in $\delta(X_{j-1})$ is occupied by a searcher and all other searchers are free. If $|\{u, v\} \cup \delta(X_{j-1})| \leq k$, we may place free searchers on both ends of e_j and execute node-search-clearing. Assume that $|\{u, v\} \cup \delta(X_{j-1})| > k$. Since $|\delta(X_{j-1})| \leq k-1$, it follows that $|\delta(X_{j-1})| = k-1$ and $\{u, v\} \cap \delta(X_{j-1}) = \emptyset$. Thus, we have one free searcher. We now prove that $u \in C_{j-1}$. If $u \notin C_{j-1}$, then $u \in \delta(X_j)$ and $|\delta(X_j)| = k$, which contradicts the condition that $(X_0, X_1, \ldots, X_n)$ has width at most $k-1$. Thus, u has no contaminated in-edges and we can place the free searcher on u and then slide the searcher from u to v along (u, v) to clear e_j by edge-search-clearing.

From Lemmas 1, 2 and 3, we have the following result.

Lemma 4. *Given a digraph D that has no multiple edges, the following are equivalent:*

(i) $\mathrm{mds}(D) \leq k$;
(ii) *there is a campaign in D of width at most $k-1$;*
(iii) *there is a progressive campaign in D of width at most $k-1$; and*
(iv) *there is a monotonic mixed directed search strategy that clears D using at most k searchers.*

From Lemma 4, we can prove the monotonicity of the mixed directed search model.

Theorem 1. *Given a digraph D, if $\mathrm{mds}(D) = k$, then there is a monotonic mixed directed search strategy that clears D using k searchers.*

3 Monotonicity of the Directed Search Model

In Section 2, we have proved that the mixed directed search problem is monotonic. In this section we will prove that the monotonicity of the mixed directed search problem implies the monotonicity of the directed search problem. The following lemma describes a relationship between the directed searching and the mixed directed searching.

Lemma 5. *If D is a digraph, then $\mathrm{ds}(D) - 1 \leq \mathrm{mds}(D) \leq \mathrm{ds}(D)$.*

The two equalities in Lemma 5 can be achieved. From Lemma 5, we know that the difference between $\mathrm{ds}(D)$ and $\mathrm{mds}(D)$ is not a fixed constant. It is not easy to use this lemma to prove the monotonicity of the directed search model. However, we can transform D into another digraph D^* such that $\mathrm{ds}(D) = \mathrm{mds}(D^*)$. Then we can use this relation to prove the monotonicity of the directed search model.

Theorem 2. *For a digraph D, let D^* be a digraph obtained from D by replacing each edge $(u, v) \in E(D)$ by two directed paths (u, v', v) and (u, v'', v). For $(u, v) \in E(D)$, let $f^1_{(u,v)} = (u, v')$, $f^2_{(u,v)} = (v', v)$, $f^3_{(u,v)} = (u, v'')$ and $f^4_{(u,v)} = (v'', v)$. The following are equivalent:*

(i) $\mathrm{ds}(D) \leq k$;
(ii) $\mathrm{mds}(D^*) \leq k$;
(iii) *there is a progressive campaign $(X_0, X_1, \ldots, X_n)$ in D^* of width at most $k-1$ such that for each $(u, v) \in E(D)$, $m_1 < m_2$ and $m_3 < m_4$, where m_i, $1 \leq i \leq 4$, is the subscript of X_{m_i} that is the first set containing $f^i_{(u,v)}$;*
(iv) *there is a monotonic mixed directed search strategy that clears D^* using at most k searchers such that for each $(u, v) \in E(D)$, $f^1_{(u,v)}$ is cleared before $f^2_{(u,v)}$ and $f^3_{(u,v)}$ is cleared before $f^4_{(u,v)}$; and*
(v) *there is a monotonic directed search strategy that clears D using at most k searchers.*

Proof. (i)⇒(ii). Let $(s_1, s_2, \ldots, s_n)$ be a directed search strategy of D using at most k searchers. We will inductively construct a mixed directed search strategy $(S'_1, S'_2, \ldots, S'_n)$ of D^* using at most k searchers, where S'_i is a subsequence of actions corresponding to s_i. Since s_1 is a placing action, let S'_1 be the same placing action. Suppose that we have constructed $S'_1, S'_2, \ldots, S'_{j-1}$ such that the following two conditions are satisfied: (1) the set of occupied vertices immediately after s_{j-1} is the same as the set of occupied vertices immediately after the last action in S'_{j-1}, and (2) if $(u, v) \in E(D)$ is cleared immediately after s_{j-1}, then the corresponding four edges $f^i_{(u,v)} \in E(D^*)$, $1 \leq i \leq 4$, are also cleared immediately after the last action in S'_{j-1}.

We now construct S'_j. If s_j is a placing action that places a searcher on an unoccupied vertex, S'_j will take the same action. If s_j is a placing action that places a searcher on an occupied vertex, S'_j will be empty. If s_j is a removing action that removes the only searcher from a vertex, S'_j will take the same action. If s_j is a removing action that removes a searcher from a vertex occupied by at least two searchers, S'_j will be empty. If s_j is a sliding action that slides a searcher from vertex u to vertex v along edge (u, v) to clear the contaminated edge (u, v), we have two cases.

Case 1. All edges with head u are cleared in D immediately before s_j. By the hypothesis, the vertex $u \in V(D^*)$ is also occupied and all edges with head u in D^* are also cleared immediately after the last action in S'_{j-1}. If v is not occupied, then we can construct S'_j as follows: edge-search-clearing (u, v'), edge-search-clearing (v', v), removing the searcher from v, placing the searcher on u, edge-search-clearing (u, v'') and edge-search-clearing (v'', v). If v is occupied,

then we can construct S'_j as follows: edge-search-clearing (u, v'), node-search-clearing (v', v), removing the searcher from v', placing the searcher on u, edge-search-clearing (u, v''), node-search-clearing (v'', v), and removing the searcher from v''.

Case 2. At least one edge with head u is contaminated in D immediately before s_j. We know that there is at least one searcher on u while performing s_j, which implies that u is occupied by at least two searchers immediately before s_j. By the hypothesis, the vertex $u \in V(D^*)$ is also occupied and we have at least one free searcher immediately after the last action in S'_{j-1}. If v is not occupied, then we can construct S'_j as follows: placing a searcher on v', node-search-clearing (u, v'), edge-search-clearing (v', v), removing the searcher from v, placing the searcher on v'', node-search-clearing (u, v'') and edge-search-clearing (v'', v). If v is occupied, then we can construct S'_j as follows: placing a searcher on v', node-search-clearing (u, v'), node-search-clearing (v', v), removing the searcher from v', placing the searcher on v'', node-search-clearing (u, v''), node-search-clearing (v'', v), and removing the searcher from v''.

If s_j is a sliding action that slides a searcher from vertex u to vertex v along edge (u, v) but does not clear the contaminated edge (u, v), we know that immediately before s_j, u is occupied by only one searcher and at least one edge with head u is contaminated. By the hypothesis, the vertex $u \in V(D^*)$ is also occupied immediately after the last action in S'_{j-1}. If v is occupied, then S'_j consists of only one action: removing the searcher from u. If v is not occupied, then S'_j consists of two actions: removing the searcher from u and placing it on v.

It is easy to see that $(S'_1, S'_2, \ldots, S'_n)$ can clear D^* using at most k searchers.

(ii)⇒(iii). Since D^* has no multiple edges, by Lemma 4, there is a progressive campaign $(X_0, X_1, \ldots, X_n)$ in D^* of width at most $k-1$. We can modify this campaign to satisfy the requirement of (iii). By Definition 2, we know that m_1, m_2, m_3 and m_4 have different values. We have four cases regarding the smallest value.

Case 1. $m_1 = \min\{m_1, m_2, m_3, m_4\}$. We already have $m_1 < m_2$. If $m_3 > m_4$, then, for each $m_1+1 \le i \le m_3$, replace X_i by $X'_i = X_{i-1} \cup \{f^3_{(u,v)}\}$. Since $X'_{m_3} = X_{m_3-1} \cup \{f^3_{(u,v)}\} = X_{m_3}$ and $v'' \notin \delta(X'_i)$, $m_1 + 1 \le i \le m_3$, it is easy to see that the updated campaign is still a progressive campaign in D^* of width at most $k-1$. Let the updated campaign still be denoted by $(X_0, X_1, \ldots, X_n)$ and m_i $(1 \le i \le 4)$ is the subscript of X_{m_i} that is the first set containing $f^i_{(u,v)}$ in the updated campaign. Thus, we have $m_1 < m_2$ and $m_3 = m_1 + 1 < m_4$ in the updated campaign.

Case 2. $m_2 = \min\{m_1, m_2, m_3, m_4\}$. For each $m_2 \le j \le m_1$, replace X_j by $X'_j = X_{j-1} \cup \{f^1_{(u,v)}\}$. After this modification, the first set containing $f^1_{(u,v)}$ is X'_{m_2} and the first set containing $f^2_{(u,v)}$ is X'_{m_2+1}. Since $f^2_{(u,v)} \in X_i$ and $f^1_{(u,v)} \notin X_i$ for $m_2 \le i \le m_1 - 1$, we know that $v' \in \delta(X_i)$, $m_2 \le i \le m_1 - 1$. Thus, for $m_2 \le i \le m_1 - 1$, we have $\delta(X'_i) \subseteq (\delta(X_i) - \{v'\}) \cup \{u\}$ and $|\delta(X'_i)| \le |\delta(X_i)|$. We also know that $X'_{m_1} = X_{m_1-1} \cup \{f^1_{(u,v)}\} = X_{m_1}$. It follows that the updated campaign is still

a progressive campaign in D^* of width at most $k-1$. Let the updated campaign still be denoted by $(X_0, X_1, \ldots, X_n)$ and m_i $(1 \leq i \leq 4)$ is the subscript of X_{m_i} that is the first set containing $f^i_{(u,v)}$ in the updated campaign. Thus, we have $m_1 < m_2$. Then we can use the method described in Case 1 to achieve $m_3 < m_4$ by modifying the campaign if necessary.

Case 3. $m_3 = \min\{m_1, m_2, m_3, m_4\}$. We already have $m_3 < m_4$ and we can use the method described in Case 1 to modify the campaign such that $m_1 < m_2$ in the updated campaign.

Case 4. $m_4 = \min\{m_1, m_2, m_3, m_4\}$. We can use the method described in Case 2 to modify the campaign such that $m_3 < m_4$ and $m_1 < m_2$ in the updated campaign.

For each $(u, v) \in E(D)$, we can repeat the above procedure for the corresponding four edges $f^i_{(u,v)} \in E(D^*)$, $1 \leq i \leq 4$. Finally, we can obtain a campaign as required.

(iii)⇒(iv). Let $(X_0, X_1, \ldots, X_n)$ be the progressive campaign described in (iii). The monotonic mixed directed search strategy constructed in Lemma 3 can use at most k searchers to clear $f^1_{(u,v)}$ before $f^2_{(u,v)}$ and to clear $f^3_{(u,v)}$ before $f^4_{(u,v)}$ for each $(u, v) \in E(D)$.

(iv)⇒(v). Let $S = (s_1, s_2, \ldots, s_n)$ be a monotonic mixed directed search strategy of D^* satisfying the condition of (iv). We will construct a monotonic directed search strategy S' of D using at most k searchers. For each edge $(u, v) \in E(D)$, without loss of generality, suppose that S clears $f^1_{(u,v)}$ before $f^3_{(u,v)}$. For each i from 1 to n, consider s_i. If s_i is a placing or removing action on a vertex that is also in $V(D)$, S' will take the same action. If s_i is a placing or removing action on a vertex in $V(D^*) - V(D)$, S' will do nothing. If s_i is an edge-search-clearing action that clears edge $f^1_{(u,v)}$, then in S', we can clear $(u, v) \in E(D)$ in the same way as s_i does. If s_i is a node-search-clearing action that clears edge $f^1_{(u,v)}$ by the searcher α on u and the searcher β on v', then in S', we know that α is also on u and β is free. Thus, we can place β on u and then clear $(u, v) \in E(D)$ by sliding β from u to v along (u, v). If s_i clears edge $f^2_{(u,v)}$, $f^3_{(u,v)}$ or $f^4_{(u,v)}$, we do nothing in S'. It is easy to see that S' can clear D using at most k searchers.

(v)⇒(i). It is trivial.

4 NP-Completeness Results

Kirousis and Papadimitriou [7] proved that the node search problem is NP-complete. In this section, we will establish a relationship between the mixed directed searching and node searching. Using this relation, we can prove the mixed directed search problem is NP-complete. We can then prove the directed search problem is NP-complete from Theorem 2.

For an undirected graph G, the minimum number of searchers needed to clear G in the node search model is the *node search number* of G, denoted by $\mathrm{ns}(G)$.

Theorem 3. *Let G be an undirected graph. If $\bar{G}$ is a digraph obtained from G by replacing each edge $uv \in E(G)$ with two directed edges (u, v) and (v, u), then* $\text{mds}(\bar{G}) = \text{ns}(G)$.

Proof. In the mixed directed search model, there are four types of actions, placing, removing, node-search-clearing, and edge-search-clearing, and in the node search model, there are only two types of actions, placing and removing. Note that there is no "clearing" action in the node search model corresponding to the node-search-clearing or edge-search-clearing. A contaminated edge is cleared in the node search model if both end vertices are occupied, while a contaminated edge is cleared in the mixed directed search model only by a node-search-clearing or edge-search-clearing action.

We first show that $\text{mds}(\bar{G}) \leq \text{ns}(G)$. Let S_n be a monotonic node search strategy that clears G using k searchers. Notice that S_n is a sequence of placing and removing actions. We will construct a mixed directed search strategy S_m by inserting some node-search-clearing actions into S_n as follows. Initially, we set $S_m = S_n$. For each placing action s in S_n, let A_s be the set of cleared edges just after s and B_s be the set of cleared edges just before s. If $A_s - B_s \neq \emptyset$, then for each edge $uv \in A_s - B_s$, we insert two node-search-clearing actions into the current S_m such that they clear both (u, v) and (v, u). The order of these clearing actions is arbitrary. Finally, we have a mixed directed search strategy S_m for $\bar{G}$. It is easy to see that S_m can clear $\bar{G}$ using k searchers. Therefore, $\text{mds}(\bar{G}) \leq \text{ns}(G)$.

We now show that $\text{ns}(G) \leq \text{mds}(\bar{G})$. Let S_m be a monotonic mixed directed search strategy that clears $\bar{G}$ using k searchers. We first prove that there is no edge-search-clearing action in S_m. Suppose that s' is an edge-search-clearing action in S_m, which clears edge (u, v) by sliding from u to v. From Definition 3, all in-edges of u are cleared. Since (v, u) is cleared but (u, v) is contaminated just before s', the vertex v must contain a searcher to protect (v, u) from recontamination. From Definition 3, (u, v) must be cleared by a node-search-clearing action because both u and v are occupied just before s'. This is a contradiction. Thus, S_m consists of only three types of actions: placing, removing, and node-search-clearing. Let S_n be a sequence of actions obtained from S_m by replacing each node-search-clearing action with an empty action. We next prove that S_n is a node search strategy that clears G using k searchers.

When an edge (u, v) is cleared by a node-search-clearing action in S_m, the corresponding edge uv in G is also cleared just before the corresponding empty action in S_n because both u and v are occupied. Note that S_n may not be monotonic. For any edge $uv \in E(G)$, when both (u, v) and (v, u) are cleared just after a node-search-clearing action in S_m, uv is also cleared just before the corresponding empty action in S_n because both u and v are occupied. We now show that this uv will keep cleared to the end of the search process. Notice that only a removing action may cause recontamination in S_n. For the sake of contradiction, suppose that there is a removing action in S_n such that just after this action an edge, which is incident on the vertex from which a search is just removed, becomes recontaminated, but the corresponding two edges in

$\bar{G}$ are still cleared just after the corresponding removing action in S_m. Let r be the first such a removing action which removes a search from vertex u, and let edge $uv \in E(G)$ becomes recontaminated just after r and both (u, v) and (v, u) in $\bar{G}$ are cleared just after the corresponding removing action r' in S_m. Since uv becomes recontaminated just after r, there must exist a contaminated edge incident on u just before r. Let wu be such a contaminated edge just before r. If (w, u) is also contaminated just before r', then (u, v) becomes recontaminated just after r'. This contradicts the monotonicity of the strategy S_m. If both (w, u) and (u, w) are cleared just before r', then wu is also cleared because uv is the first edge which is recontaminated but (u, v) and (v, u) are cleared. This is a contradiction. Thus, just before r', (w, u) is cleared and (u, w) is contaminated, and w must contain a searcher to protect (w, u) from recontamination. Hence, both w and u are occupied just before r'. It follows that wu is cleared just before r. This is a contradiction. Therefore, for each edge $uv \in E(G)$, if both (u, v) and (v, u) in $\bar{G}$ are cleared, then uv is also cleared. It is easy to see that S_n can clear G using k searchers. Thus, $\mathrm{ns}(G) \leq \mathrm{mds}(\bar{G})$.

Given a digraph D, the problem of determining $\mathrm{ds}(D)$ or $\mathrm{mds}(D)$ is the optimization problem of finding the smallest k such that D can be cleared using k searchers. The corresponding decision problem for the mixed directed search problem is as follows.
Problem: Mixed Directed Searching
Instance: Digraph D, positive integer k.
Question: Can we use k searchers to clear D under the mixed directed search model?

The decision problem for the directed search problem is to determine whether we can clear D using k searchers.
Problem: Directed Searching
Instance: Digraph D, positive integer k.
Question: Can we use k searchers to clear D under the directed search model?

From Theorem 3, we have the following result.

Theorem 4. *The Mixed Directed Searching problem is NP-complete.*

Proof. We first show that the Mixed Directed Searching problem belongs to NP. Suppose we are given a digraph D and a positive integer k. From Theorem 1, a nondeterministic algorithm needs only to guess a monotonic mixed directed search strategy such that the number of actions in this strategy is $O(V(D) + E(D))$. It is easy to see that checking whether this strategy can clear D using k searchers can be accomplished in deterministic polynomial time. Thus, the Mixed Directed Searching problem is in NP. By Theorem 3, we know that the Mixed Directed Searching problem is NP-hard because the node search problem is NP-complete [7]. Therefore, the Mixed Directed Searching problem is NP-complete.

From Theorems 2 and 4, we can prove that the directed search problem is NP-hard. From Theorem 2, we can prove that the directed search problem belongs to NP. Therefore, we have the major result of this section.

Theorem 5. *The Directed Searching problem is NP-complete.*

5 Conclusion

In this paper, we investigated two new digraph searching models, directed searching and mixed directed searching, in which both searchers and intruders must move in the edge directions. Using the method proposed by Bienstock and Seymour [4], we first proved the monotonicity of the mixed directed search model. We then proved the monotonicity of the directed searching. We also give a relationship between the mixed directed searching and the node searching. From this relation and the monotonicity results, we showed that the mixed directed and directed search problems are NP-complete.

References

1. B. Alspach, D. Dyer, D. Hanson and B. Yang, Some basic results in arc searching, *Technical report CS-2006-10*, University of Regina, 2006.
2. J. Barat, Directed path-width and monotonicity in digraph searching, *Graphs and Combinatorics*, 22 (2006) 161–172.
3. D. Berwanger, A. Dawar, P. Hunter and S. Kreutzer, DAG-width and parity games, Proceedings of *STACS*, Lecture Notes in Computer Science, Vol.3884, 524–436, 2006.
4. D. Bienstock and P. Seymour, Monotonicity in graph searching, *Journal of Algorithms*, 12 (1991) 239–245.
5. F. Fomin and D. Thilikos, On the monotonicity of games generated by symmetric submodular functions, *Discrete Applied Mathematics*, 131 (2003) 323–335.
6. T. Johnson, N. Robertson, P. Seymour and R. Thomas, Directed tree-width, *Journal of Combinatorial Theory Series B*, 82 (2001) 138–154.
7. L. Kirousis and C. Papadimitriou, Searching and pebbling, *Theoret. Comput. Sci.*, 47 (1996) 205–218.
8. A. LaPaugh, Recontamination does not help to search a graph. *Journal of ACM*, 40 (1993) 224–245.
9. N. Megiddo, S. Hakimi, M. Garey, D .Johnson and C. Papadimitriou, The complexity of searching a graph, *Journal of ACM*, 35 (1998) 18–44.
10. T. Parsons, Pursuit-evasion in a graph. *Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, 426–441, 1976.
11. B. Reed, Introducing Directed Tree Width, 6th Twente Workshop on Graphs and Combinatorial Optimization (Enschede, 1999), 8 pp. (electronic), *Electron. Notes Discrete Math.*, 3, Elsevier, Amsterdam, 1999.
12. M. Safari, D-Width: A More Natural Measure for Directed Tree Width, Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 3618, pp. 745–756, 2005.
13. P. Seymour and R. Thomas, Graph searching and a min-max theorem for tree-width, *Journal of Combinatorial Theory Series B*, 58 (1993) 22–33.
14. B. Yang and Y. Cao, Monotonicity of strong searching on digraphs, submitted.
15. B. Yang and Y. Cao, On the monotonicity of weak searching on digraphs, submitted.
16. B. Yang and Y. Cao, Digraph searching, directed vertex separation and directed pathwidth, submitted.

Protecting Against Key Escrow and Key Exposure in Identity-Based Cryptosystem

Jin Wang[1], Xi Bai[2], Jia Yu[3], and Daxing Li[1]

[1] Institute of Network and Information Security, Shandong University, Jinan 250100, China
[2] College of Computer Science and Technology, Jilin University, Changchun 130012, China
[3] College of Information Engineering, Qingdao University, Qingdao 266071, China
wangjin06@mail.sdu.edu.cn, xibai@email.jlu.edu.cn

Abstract. Standard identity-based cryptosystems typically rely on the assumption that secret keys are kept perfectly secure. However, in practice, there are two threats to the key security in identity-based cryptosystems. One inherent problem is key escrow, that is, the Key Generation Center (KGC) always knows a user's secret key and the malicious KGC can impersonate the user. Meanwhile, another threat is that a user's secret key may be exposed to an adversary in an insecure device, and key exposure typically means that security is entirely lost. At present, there is no solution that can simultaneously solve both of above problems. In this paper, we first present a secure key issuing and updating model for identity-based cryptosystems. Our suggestion is an intermediate between the identity-based key insulation and distributing authorities approach, and can simultaneously solve both key escrow and key exposure problems. We formalize the definition and security notion of the corresponding encryption scheme (IBKUE) and signature scheme (IBKUS), and then propose an IBKUE scheme based on Boneh-Franklin's scheme [2] and an IBKUS scheme based on Cha-Cheon's scheme [9]. Both of the schemes are secure in the remaining time periods against an adversary who compromises the KGC and obtains a user's secret key for the time periods of its choice. All the schemes in this paper are provably secure in the random oracle model.

Keywords: Identity-based cryptography, key escrow, key exposure, bilinear pairings, key-insulated cryptosystem.

1 Introduction

1.1 Background

In traditional public key cryptosystems (PKC), the association between a user's identity and his public key is bind with a certificate issued by a certification authority (CA), anyone who wants to use a public key must first verify the corresponding certificate. In order to simplify the certificate management process,

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 148–158, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Shamir [1] introduced the concept of identity-based cryptography (IBC) in 1984. Identity-based cryptography can eliminate the need for certificates by allowing a user's public key to be derived from his identity information, such as an email address, while the corresponding private key is calculated by a trusted authority called Key Generator Center (KGC). Ever since Boneh and Franklin [2] proposed the first practical identity-based encryption scheme based on bilinear pairings in 2001, many identity-based schemes [8,9,10] have been proposed.

Standard identity-based cryptosystems typically rely on the assumption that secret keys are kept perfectly secure. However, in practice, there are two threats to the key security in identity-based cryptosystems. One inherent problem is key escrow. In such cryptosystem, the KGC is involved in issuing secret keys to users whose identity is assumed to be unique in the system. Knowing the KGC's master key should be able to impersonate a user, that is, carrying out any cryptographic operations as the user, so there is no user privacy and authenticity in the system. Another threat is that user's secret key maybe exposure to an adversary in an insecure device. In many cases, it is easier to obtain a secret key from a stolen device than to break the computational assumption on which the security of the system is based. This threat is increasing nowadays with more and more cryptographic primitives are deployed on insecure devices such as mobile devices.

1.2 Related Work

In recent years, to tackle the key escrow problem, several proposals have been made using multiple authority approach [2,5] or using some user chosen secret information [4,11]. Boneh and Franklin's scheme [2] distributed the master key of the KGC into multiple authorities, and a user's private key is computed in a threshold manner. In addition, Chen et al.'s scheme [5] generated a new private key by adding subkeys from multiple authorities. In both the above approaches the key escrow problem can be avoided, but multiple identification by each authority is quite a burden. Lee et al. proposed a new scheme [3] in which a user's private key is issued by a KGC and its privacy is protected by multiple key privacy authorities (KPA). However, in this scheme private keys of all users have to be reconstructed if even one KPA is compromised. Gentry [4] proposed a certificate-based encryption where secure key issuing was provided using some user-chosen secret information, but it became a certificate-based scheme losing the advantage of identity-based cryptography. Certificateless public key cryptography [11] successfully removed the necessary of certificate in a similar design using user-chosen secret information, and secure in the attack from malicious KGC or a malicious third party, but their scheme provides only implicit authentication of the public key.

While on the other hand, to mitigate the damage caused by user's key exposure in identity-based cryptosystem, a recently proposed method is constructing identity-based key-insulated cryptosystem [13,14,21]. Key-insulated mechanism uses a combination of key splitting and key evolution to protect against key exposure. In identity-based key-insulated model, the secret key associated with

an ID is shared between the user and a physically-secure device. To perform cryptographic operations, the user refreshes his secret key periodically by interacting with the secure device. The life time of the system is divided into distinct periods $1, ..., N$. In an identity-based (t, N)-key-insulated cryptosystem an adversary who compromises the insecure device and obtains keys for up to t time periods is unable to violate the security of the cryptosystem for any of the remaining $N - t$ periods. Note, however, all of identity-based key-insulated cryptosystems [13,14,21] still suffer from key escrow problem.

At present, no solutions exist that solves both key escrow and key exposure problems in identity-based cryptosystems; schemes that solve the key escrow problem still face the threat of key exposure on user's insecure device, and vice versa.

1.3 Our Contribution

First, we propose an identity-based key issuing and updating model that doesn't suffer from key escrow and key exposure. Our suggestion is involving an intermediate between the identity-based key insulation and distributing authorities approaches. The main idea of our paper is using multiple security mediators, which we call key privacy authorities (KPAs) to provide private key privacy service. Assume that there are n KPAs, each KPA is corresponding to some specified time periods. Our suggestion supports unbounded number of time periods, in time period i, $\mathrm{KPA}_k(k{=}i \bmod n)$ is responsible for providing key issuing for a user, and the user generates his secret key by adding partial private keys issued by KPA_k and the KGC, so the key escrow can be eliminated. Furthermore, $\mathrm{KPA}_k(k{=}1,\ldots,n)$ also provides user key updating information for time periods $T_k{=}\{i|k{=}i \bmod n\}$. Key exposure can be reduced by updating user's keys periodically with the help of the KPAs. The key pair generated in our suggestion can be used for pairing-based identity-based cryptosystems, such as encryptions, signatures, etc. Afterwards, we construct two implementation of our key issuing and updating model: an identity-based encryption scheme based on Boneh-Franklin's scheme [2] and an identity-based signature scheme based on Cha-Cheon's scheme [9], both of the schemes are secure in the random oracle model. To the author's best knowledge, our proposed schemes are the first to protect against both key escrow and key exposure problems simultaneously in identity-based cryptosystems.

2 Preliminaries

Bilinear Pairings. Let G_1 be an additive group of prime order q and G_2be a multiplicative group of the same order q. A bilinear pairing is a map $\hat{e}$: $G_1 \times G_1 \rightarrow G_2$, with the following properties:

1. *Bilniearity:* $\hat{e}(aP,bQ){=}\hat{e}(P, Q)^{ab}$, for all $P, Q \in G_1, a,b \in Z_q^*$;
2. *Non-degeneracy*: There exist $P,Q \in G_1$, such that $\hat{e}(P,Q) \neq 1$;

3. *Computability*: There is an efficient algorithms to compute $\hat{e}(P, Q)$ for all $P, Q \in G_1$.

The security of the pairing-based schemes relies the following mathematical problems.

Bilinear Diffie-Hellman Problem (BDHP). Given (P, aP, bP, cP), compute $\hat{e}(P, P)^{abc}$. a, b, c be elements of Z_q^*.

Discrete Logarithm Problem (DLP). Given P,Q, find an integer n such that P=nQ, where such n exists.

Computational Diffie-Hellman Problem (CDHP). Given (P, aP, bP), compute abP.P,Q be elements of G_1 and a,b be elements of Z_q^*.

Decisional Diffie-Hellman Problem (DDHP). Given (P,aP,bP,cP), decide whether c=ab in Z_q^*.

3 Our Key Issuing and Updating Model

In this section, we introduce a key issuing and updating model for identity-based cryptosystem that can refresh user's secret key periodically and doesn't suffer from key escrow. The main idea behind our scheme is using n KPAs to provide private key privacy services. Our suggestion supports unbounded number of time periods, each KPA$_k$(k=1,. . . ,n) is corresponding to some specified time periods T_k=$\{i|k$=$i \bmod n\}$. The KPA$_k$(k=1,. . . ,n) provides privacy service for users by authenticating them and deliver a partial private key in key issuing stage and generating key refreshing information in key updating stage for the time periods which it associated with. There are two types of attacks that are generally considered in our model:

Type 1-Key Exposure Attack. The adversary compromises a user's secret key in an insecure device for time period $i$and tries to impersonate the user in another time period.
Type 2-Malicious KGC Attack. The adversary controls the master key of the KGC and tries to impersonate a user.

3.1 Entities Involved

There are three entities involved namely *KGC*, *KPA* and user in our protocol.

- **Key Generation Center (KGC).** The KGC is the central entity meant for user registration. It checks the user identity and issues partial private keys to the registered users.
- **Key Privacy Authority (KPA).** Multiple KPAs are used to providing the key privacy service. Each KPA$_k$(k=1,. . . ,n)is corresponding to some specified time periods. In time period i, KPA$_k$(k=$i \bmod n$) is responsible for authenticating users and issuing partial private keys in parallel with the KGC to tackle key escrow. Furthermore, each user is allowed to interact with the KPA$_k$ (k=1,. . . ,n) to derive a temporary key for time period i (k=$i \bmod n$) to eliminate key exposure damage.

- **User.** A user is an entity that uses identity-based cryptosystem. In time period$i$, if a user wants to be issued a private key, he must be initially registered at the KGC and gets partial private keys from the KGC and $\mathrm{KPA}_k(k{=}i \bmod n)$ in parallel. Any user can update its secret key periodically by interacting with KPAs.

3.2 Framework and Security Model

Definition 1. *Our identity-based Key issuing and updating model (IBKIU) is 3-tuple of algorithms (Setup, Extract, Upd), that are defined as follows:*

- **Setup:** This algorithm takes security parameter k, and returns a list of system parameters and the master key for the KGC and n KPAs. We assume that parameters are publicly and authentically available.
- **Extract:** It takes as input a user's identity *ID*, index i of a time period and the master key of the KGC and the $\mathrm{KPA_k}$ ($k = i \bmod n$), it returns his/her base secret key.
- **Upd:** The user key-update algorithm. It takes as input indices i, j for time periods, the master key of KPA_k ($k = j \bmod n$) and a secret keySK^i_{ID}, it returns the secret key SK^j_{ID} for time period j.

Therefore the key pair generated in our model can be used for pairing-based identity-based cryptosystems, such as encryptions and signatures. We construct two implementation of IBKIU model: an identity-based encryption scheme (IBKIUE) and an identity-based signature scheme (IBKIUS).

Definition 2 (IBKIUE). *An IBKIUE scheme consists five polynomial-time algorithms (Setup, Extract, Upd, Encrypt and Decrypt).*

- **Setup**, **Extract**, **Upd** are defined as the same in Definition 1.
- **Encrypt:** The encryption algorithm. On input system parameters, *ID*, index i of a time period, and a message M, it returns a ciphertext C for *ID* at time period i.
- **Decrypt:** The decryption algorithm. On input a secret key SK^i_{ID} and a ciphertext (ID, i, C), it returns a message M or the special symbol $\perp$.

For security considerations, we define the following oracles:

- $\mathcal{EO}$:The Extraction Oracle, on input *ID*, time period index i, master keys of the *KGC and KPA*$_k$(where $k{=}i \bmod n$), output the corresponding secret key SK^i_{ID} by running algorithm Extract.
- $\mathcal{KEO}$:The Key Exposure Oracle, on input *ID* and i, the challenger runs algorithm Upd to get the temporary secret key SK^i_{ID} for *time period i*.
- $\mathcal{DO}$:The Decryption Oracle, on input (*ID*, i, C), run Extract (*ID*, i) to get the private key SK^i_{ID}, then run algorithm Decrypt (*ID*, i, SK^i_{ID}, C) and return the resulting plaintext.

Here, we formalize more rigorous security notions than [2] to allow adversaries who can make both types of attacks: compromising the KGC's master key and getting based keys or temporary keys for identities and time periods of their choice. We define chosen ciphertext security for our scheme as follows.

Definition 3. *We say that our IBKIUE scheme is semantically secure against an adaptively chosen ciphertext attack (IND-ID-CCA) if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger $\mathcal{C}$ in the following game:*

Setup: The challenger takes a security parameter k and runs the Setup algorithm. It gives $\mathcal{A}$ the resulting system parameters. The challenger also gives the master-key of the KGC to $\mathcal{A}$.
Phase 1: $\mathcal{A}$ queries $\mathcal{EO}(ID, i)$, $\mathcal{KEO}$ (ID, i) and $\mathcal{DO}$ (ID, i, C) in arbitrary interleave. These queries may be asked adaptively.
Challenge: Once $\mathcal{A}$ decides that Phase 1 is over it outputs the challenge identity ID^*, time period i and two equal length plaintexts: M_0 and M_1. In particular, the constraint is that (ID^*, i) did not appear in any query to $\mathcal{EO}$ and $\mathcal{KEO}$ in phase 1. The challenger now picks a random bit $b \in \{0, 1\}$ and computes $C^* = Encrypt(ID^*, i, M_b)$. If the output of the encryption is $\perp$, then $\mathcal{A}$ has immediately lost the game. Otherwise, C^* is delivered to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ issues a second sequence of requests as in Phase 1. In particular, no private key extraction or key exposure query on (ID^*, i) is allowed. Moreover, no decryption query can be made on (ID^*, i, C^*).
Guess: Finally, the adversary outputs a guess $b' \in \{0, 1\}$. The adversary wins the game if $b = b'$. We define $\mathcal{A}$'s advantage in the game to be $Adv(\mathcal{A}) = 2(Pr[b = b'] - \frac{1}{2})$.

Definition 4 (IBKIUS). *An IBKIUS scheme consists five polynomial-time algorithms (Setup, Extract, Upd, Sign and Verify).*

- **Setup**, **Extract**, **Upd** are defined as the same in Definition 1.
- **Sign:** On input ID, an index i of a time period, the secret key SK^i_{ID} and a message M, it generates a signature σ.
- **Verify:** On input system parameters, user identity ID, indexes i of a time period, a message M and a signature σ, returns 1 or 0 for σ accept or reject, respectively.
  We also define the signature oracle as follows.
- $\mathcal{SO}$: The Signing Oracle, on input a message M, ID and i, output σ as a signature.

Definition 5. *We say that our IBKIUS scheme is semantically secure against adaptively chosen message and ID attack if no polynomially bounded adversary A has a non-negligible advantage against the challenger C in the following game:*

1. $\mathcal{C}$ takes a security parameter k and runs the Setup algorithm and the resulting system parameters are given to $\mathcal{A}$. $\mathcal{C}$ also sends the master key of the KGC to $\mathcal{A}$.

2. $\mathcal{A}$ issues a number of queries on $\mathcal{EO}(ID, i)$, $\mathcal{KEO}(ID, i)$ and $\mathcal{SO}(ID, i, M)$. These queries may be asked adaptively.
3. Finally, $\mathcal{A}$ outputs (ID^*, i, M^*, σ), where ID^* is target identity chosen by $\mathcal{A}$, i is an index of a time period, M^* is a message and σ is a signature. Here (ID^*, i) is not equal to any input of $\mathcal{EO}$ and $\mathcal{KEO}$; (ID^*, i, M^*) is not equal to any input of $\mathcal{SO}$.

$\mathcal{A}$ wins the game if σ is a valid signature of M^* for identity ID^* at period i.

4 Our Proposed Encryption Scheme

This section presents our IBKIUE scheme based on Boneh-Franklin's IBE scheme [2]. Our proposed scheme solves key escrow and key exposure problems simultaneously.

4.1 The IBKIUE Scheme

Setup: Given a security parameter k, The KGC runs $\mathcal{IG}$, BDH parameter generator, to output groups G_1, G_2 of prime order q, a generator P of G_1,a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$.It chooses three hash functions $H_1 : \{0,1\}^* \rightarrow G_1$ $H_2 : \{0,1\}^* \rightarrow G_1$, $H_3 : G_2 \rightarrow \{0,1\}^n$ for some n. The message space is $M = \{0,1\}^n$. The KGC then picks a random $s_0 \in Z_q^*$ as its secret key and computes its public key $P_0 = s_0 P$. Each $\mathrm{KPA}_i (i = 1, ..., n)$ also chooses its master key s_i and publishes $P_i = s_i P$.

Extract: In time period i, a user with identity ID first computes $k = i \bmod n$, then request the KGC and the KPA_k to issue a base private key by sending ID. The KGC and the KPA_k issue a partial private key in parallel as follows.

The KGC:

1. Checks the identification of the user.
2. Computes the public key of the user as $Q_{ID} = H_1(ID)$.
3. Set the partial base key as $d_{ID}^0 = s_0 Q_{ID}$, where s_0 is the master key of the KGC.
4. Sends d_{ID}^0 to the user securely.

The KPA_k:

1. Checks the identification of the user.
2. Computes the public key of the user as $Q_{ID} = H_1(ID)$ and $T_{ID,i} = H_2(ID, i)$.
3. Issue the partial base key as $d_{ID}^k = s_k Q_{ID}$, and $t_{ID}^k = s_k T_{ID,i}$ where s_k is the master key of the KPA_k.

The user sets $s_{ID}^i = d_{ID}^0 + d_{ID}^k$, its base secret key is $SK_{ID}^i = (s_{ID}^i, t_{ID}^i)$.
Upd: Given a time indices i and an identity ID, the $\mathrm{KGC}_k (k = i \bmod n)$ works as follows:

1. Computes $Q_{ID} = H_1(ID)$, $T_{ID,i} = H_2(ID, i)$.
2. Issues a partial private key for ID as $d^k_{ID} = s_k Q_{ID}$, $t^i_{ID} = s_k T_{ID,i}$.
3. Sends d^k_{ID} and t^i_{ID} to the user.

The user computes $s^i_{ID} = d^0_{ID} + d^k_{ID}$ and sets the secret key $SK^i_{ID} = (s^i_{ID}, t^i_{ID})$ for time period i.

Encrypt: Given an index i of a time period, a message M, and an identity ID, pick a random number $r \in Z^*_q$, set the cipher text to be $C =< U = rP, V = M \oplus H_3(g^r_{ID}) >$, where $g_{ID} = (Q_{ID}, P_0 + P_k)\hat{e}(T_{ID,i}, P_k)$.

Decrypt: Given the secret key $SK^i_{ID} = (s^i_{ID}, t^i_{ID})$ and a cipher text $C =< U, V >$ for the identity ID at a time period i, computes $M = V \oplus H_3(\hat{e}(s^i_{ID} + t^i_{ID}, U))$.

4.2 Correctness

The decryption of our encryption scheme is justified by the following equations:

$$\begin{aligned} g^r_{ID} &= \hat{e}(Q_{ID}, P_0 + P_k)^r \hat{e}(T_{ID,i}, P_k)^r \\ &= \hat{e}(Q_{ID}, s_0 P + s_k P)^r \hat{e}(T_{ID,i}, s_k P)^r \\ &= \hat{e}(s_0 Q_{ID} + s_k Q_{ID}, rP)\hat{e}(s_k T_{ID,i}, rP) \\ &= \hat{e}(s^i_{ID}, rP)\hat{e}(t^i_{ID}, rP) \\ &= \hat{e}(s^i_{ID} + t^i_{ID}, U) \end{aligned}$$

4.3 Analysis and Comparison

Consider in time period i, the KGC can't learn the KPA_k's($k = i \bmod n$) private key and, likewise, the KPA_k can't know the master key of the KGC. Note that the KGC and the KPA_k are two distinct authorities and do not collude with each other. The user's private key consists of two half parts s^i_{ID} and t^i_{ID}. $s^i_{ID} = d^0_{ID} + d^k_{ID}$ is a sum of two partial keys generated by the KGC and the KPA_k. t^i_{ID} is associated with the time index generated by the KPA_k. So the KGC and the KPA_k never knows the both half of the user's private key. Further more, if the KGC collude with a KPA_i, they can only obtain the user's secret keys for the time period corresponding to KPA_i, and can not impersonate any users successfully in other time periods. On the other hand, even if all of n KPAs collude, they can not work out the KGC's master key and user's secret key for any time period.

In our scheme, exposure of a user's secret key at a given time will not enable an adversary to derive the user's key for the remaining time, it achieves the similar property as identity-based $(N-1, N)$ key-insulated cryptosystems.

On the point of efficiency, remark that, in identity-based key-insulated cryptosystem, authentication between the user and the physically-secure device is necessary (Previous work assumed that authentication is handled by an underlying protocol). Considering in our scheme, if a user refreshes his secret key for time period i, the corresponding $\mathrm{KPA}_k(k = i \bmod n)$ also has to check user's identity, which is quite a burden. However, our scheme doesn't suffer from key escrow, which is still remained in identity-based key-insulated scheme.

Our scheme also has desirable properties such as unbounded number of time periods, and random-access updates.

4.4 Security Analysis

The proof of security for our scheme makes use of a weaker notion of security known as semantic security against a chosen plaintext attack (denoted IND-ID-CPA)[2]. IND-ID-CPA is similar to IND-ID-CCA except that the adversary is more limited: it cannot issue decryption queries while attacking the challenge public key.

Theorem 1. *In the random oracle model, if an adversary $\mathcal{A}$ has an advantage ε in IND-ID-CPA attack on our proposed encryption scheme with running in a time t and making at most q_{H_1}, q_{H_2}, q_{H_3}, q_E and q_K times queries to hash functions H_1, H_2, H_3, $\mathcal{EO}$ and $\mathcal{KEO}$, respectively, then the BDHP can be solved in G_1 with an advantage*

$$\varepsilon' > \frac{1}{n} \cdot \frac{1}{q_{H_1}}(1 - \frac{q_E + q_K}{n \cdot q_{H_1}}) \cdot \frac{2}{q_{H_3}}\varepsilon$$

within a time

$$t' < t + (q_{H_1} + q_{H_2} + q_{H_3} + 3q_E + 3q_K + 3)t_m$$

where t_m denotes the running time of computing a scalar multiplication in G_1.

Proof: The whole proof is omitted here for space limitation and can be seen in the full version of our paper.

5 Our Proposed Signature Scheme

Our key issuing and updating model can also be applied to identity-based signature scheme. In this section, an IBKIUS scheme is presented based on Cha-Cheon's identity-based signature scheme [9].

5.1 The IBKIUS Scheme

Our proposed signature scheme consists of 5-tuple of poly-time algorithms (Setup, Extract, Upd, Sign, Verify):

Setup: Given a security parameter k, the KGC selects two groups G_1, G_2 of prime order q, a generator P of G_1, a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$, and three hash functions $H_1 : \{0,1\}^* \rightarrow G_1$, $H_2 : \{0,1\}^* \rightarrow G_1$, $H_3 : G_2 \rightarrow \{0,1\}^n$. The message space is $M = \{0,1\}^n$. The KGC then picks a random $s_0 \in Z_q^*$ as its secret key and computes its public key $P_0 = s_0 P$. Each KPA$_i (i = 1, ..., n)$ also chooses its master key $s_i \in Z_q^*$ and publishes $P_i = s_i P$.

Extract: As in IBKIUE scheme.

Upd: As in IBKIUE scheme.

Sign: On inputting an index i of a time period, a message M, and a secret key $SK_{ID}^i = (s_{ID}^i, t_{ID}^i)$, pick a random number $r \in Z_q^*$, compute $U_1 = rQ_{ID}$,

$U_2 = rT_{ID,i}$, $h = H_3(M, U_1 + U_2)$, $V = (r + h)(s^i_{ID} + t^i_{ID})$, and return (i, σ), where $\sigma = (U_1, U_2, V)$.

Verify: On inputting a tuple (i, σ), the public key P_0, $P_k(k = i \bmod n)$, and a message M, the verifier does the following:

1. Parses σ as $\sigma = (U_1, U_2, V)$.
2. Computes $Q_{ID} = H_1(ID), T_{ID,i} = H_2(ID, i), h = H_3(M, U_1 + U_2)$.
3. Accepts the signature if $\hat{e}(P, V) = \hat{e}(P_0 + P_k, U_1 + hQ_{ID})\hat{e}(P_k, U_2 + hT_{ID,i})$.

5.2 Security Analysis

Theorem 2. *In the random oracle model, if an adversary $\mathcal{A}$ has an advantage ε in adaptively chosen message and ID attack on our signature scheme with running in a time t and making at most $q_{H_1}, q_{H2}, q_{H_3}, q_E, q_K$ and q_s times queries to random oracles $H_1, H_2, H_3, \mathcal{EO}, \mathcal{KEO}$, and $\mathcal{SO}$ respectively, then the CDHP can be solved in G_1 with an advantage*

$$\varepsilon' > \frac{\varepsilon - (q_S(q_{H_3} + q_S) + 1)/2^k}{e(1 + q_E + q_K)}$$

within time

$$t' < t + (q_{H_1} + q_{H_2} + q_{H_3} + 3q_E + 3q_K + 6q_S + 1)t_m$$

where t_m denotes the running time of computing a scalar multiplication in G_1.

Proof: The whole proof is omitted here for space limitation and can be seen in the full version of our paper.

6 Conclusion

In this paper, we consider how to solve both key escrow and key exposure problems in identity-based cryptosystems. In our suggestion, we subdivide the key issuing authority into KGC and n KPAs. Each KPA is corresponding to some specified time periods. With this separation, we could consider eliminating key escrow of the KGC, and even if the KGC colludes with the $\text{KPA}_k(k = 1, ..., n)$, they can only obtain the user's secret keys for the time periods $T_k = \{i | k = i \bmod n\}$, and can not impersonate users in other time periods. Furthermore, $\text{KPA}_k(k = 1, ..., n)$ also provides key refreshing information for time periods T_k. Key exposure on user's insecure device can be reduced by updating keys periodically with the help of the KPAs. The proposed concrete encryption and signature scheme will be a practical solution to consider both key escrow and key exposure simultaneously in identity-based cryptosystems.

References

1. A.Shamir, *Identity-based cryptosystems and signature schemes*, CRYPTO'84, LNCS 196, Springer-Verlag, pp.47-53, 1984.
2. D.Boneh and M.Franklin, *Identity-based encryption from the Weil pairing*, CRYPTO'01, LNCS 2139, Springer-Verlag, pp.213-229, 2001.
3. B.Lee, C.Boyd, E.Dawson, K.Kim, J.Yang and S.Yoo, *Secure Key Issuing in ID-Based Cryptography*, ACM Second Australasian Information Security Workshop, New Zealand, pp.69-74, 2004.
4. C. Gentry, *Certificate-based encryption and the certificate revocation problem*. Cryptology-EUROCRYPT 2003, LNCS. 2656, Springer-Verlag, pp. 272-293,2003.
5. L.Chen, K.Harrison, N.Smart and D.Soldera, *Applications of multiple trust authorities in pairing based cryptosystems*, InfraSec 2002, LNCS 2437, Springer-Verlag, pp.260-275, 2002.
6. Y. Dodis, J. Katz, S. Xu and M. Yung. *Key-Insulated Public-Key Cryptosystems*. EuroCrypt'02, pp. 65-82, Springer-Verlag, 2002.
7. E. Fujisaki and T. Okamoto. *Secure integration of asymmetric and symmetric encryption schemes*. Crypto'99, LNCS 1666, pp. 537-554, Springer-Verlag, 1999.
8. F.Hess, *Efficient Identity Based Signature Schemes based on Pairings*, Selected Areas in Cryptography 9th Annual International Workshop, SAC2002, LNCS 2595, Springer-Verlag, pp.310-324, 2003.
9. J.C. Cha and J.H. Cheon, *An Identity-Based Signature from Gap Diffie-Hellman Groups*, Public key Cryptography-PKC 2003, LNCS 2567, Springer-Verag, pp.1830, 2003.
10. K.G.Paterson, *ID-based signatures from pairings on elliptic curves, Electronics Letters*, vol.38, no.18, pp.1025-1026, 2002.
11. S.Al-Riyami and K.G.Paterson, *Certificateless public key cryptography*, Asiacrypt 2003, LNCS 2894, Springer-Verlag, pp.452-473, 2003.
12. D.H. Yum and P.J. Lee. *Efficient Key Updating Signature Schemes Based on IBS*, Cryptography and Coding'03, LNCS 2898, pp. 167-182, Springer-Verlag, 2003.
13. Y.Zhou,Z.Cao,Z. Chai. *Identity Based Key Insulated Signature*. ISPEC'06, LNCS 3903, pp. 226-234, Springer-Verlag, 2006.
14. Y.Hanaoka, G.Hanaoka, J.Shikata, H.Imai. *Identity-Based Hierarchical Strongly Key-Insulated Encryption and Its Application*. AsiaCrypt'05, LNCS 3788, pp. 495-514, Springer-Verlag, 2005.
15. D. Pointcheval and J. Stern, *Security arguments for digital signatures and blind signatures*, J. of Cryptology, Vol. 13, pp. 361-396, 2000.
16. Ai-fen,et.al, *Separable and Anonymous Identity-Based Key Issuing without SecureChannel*, Cryptology ePrint Archive, Report2004/322, 2004.
17. Y. Dodis, J. Katz, S. Xu and M. Yung. *Strong Key-Insulated Signature Schemes*. PKC'03, LNCS 2567, pp. 130-144, Springer-Verlag, 2003.
18. A. Shamir, *How to share secret*, Comm. ACM, 22(11) pp. 612-613, 1979.
19. Y. Desmedt and Y. Frankel, *Threshold cryptosystems*, Crypto 1989, LNCS Vol. 435, pp. 307-315, 1989.
20. M. Bellare and S. Miner, *forward-secure digital signature scheme*, Crypto 1999, LNCS Vol. 1666, pp. 431-448, 1999.
21. J.Li, F.Zhang and Y.Wang, *A Strong Identity Based Key-Insulated Cryptosystem,* Advances in IFIP'2006 LNCS 4097.PP.353-362, Springer-Verlag, 2006.

Encapsulated Scalar Multiplications and Line Functions in the Computation of Tate Pairing*

Rongquan Feng** and Hongfeng Wu

LMAM, School of Math. Sciences, Peking University, Beijing 100871, P.R. China
fengrq@math.pku.edu.cn, wuhf@math.pku.edu.cn

Abstract. The efficient computation of the Tate pairing is a crucial factor to realize cryptographic applications practically. To compute the Tate pairing, two kinds of costs on the scalar multiplications and Miller's line functions of elliptic curves should be considered. In the present paper, encapsulated scalar multiplications and line functions are discussed. Some simplified formulas and improved algorithms to compute f_{3T}, f_{4T}, $f_{2T\pm P}$, f_{6T}, $f_{3T\pm P}$ and $f_{4T\pm P}$ etc., are presented from given points T and P on the elliptic curve.

Keywords: Elliptic curve, scalar multiplication, line function, Tate pairing, Miller's path.

1 Introduction

With the discovery of the identity-based encryption scheme based on the Weil pairing by Boneh and Franklin [4], cryptographic protocols based on the Weil and Tate pairings on elliptic curves have attracted much attention in recent years. Many cryptographic schemes based on the Weil or on the Tate pairing have been introduced. The readers are referred to [11] for a survey. In most of these pairing-based cryptographic schemes, the Tate pairing is the essential tool. The pairing computations are often the bottleneck to realize the cryptographic applications in practice. Therefore efficient computation of the Tate pairing is a crucial factor to realize cryptographic applications practically.

Miller [20,21] provided an algorithm to compute the Weil/Tate pairing. The Miller's algorithm itself consists of two parts: the Miller's function f_{rP} and a final exponentiation. The main part of the computation for the Tate pairing is calculating $f_{rP}(Q)$. In the computation of the Miller's function $f_{(m+n)P}(Q)$, one needs to perform a conditional scalar multiplication and compute the line functions $l_{mP,nP}(Q)$ and $l_{(m+n)P}(Q)$ from points mP and nP on the elliptic curve. The total cost of the computation of the Tate pairing is the sum of the cost of scalar multiplications and that of the computation of line functions $l_{mP,nP}(Q)$ and $l_{(m+n)P}(Q)$. So a good algorithm must think over these two kinds of costs

* Supported by the NSF of China (10571005, 60473019), by 863 Project (No. 2006AA01Z434) and by NKBRPC (2004CB318000).

** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 159–170, 2007.
© Springer-Verlag Berlin Heidelberg 2007

simultaneously. In the first, we need to find more efficient method to compute the scalar multiplication, at the same time we need to modify the algorithm so that it can be serviced to make the computation of line functions. Secondly, we need other strategies to change and simplify the formula of $f_{(m+n)P}$ so that the good algorithm of scalar multiplications can be used to the computation of $f_{(m+n)P}$. In the computation of the Tate pairing, we often need to compute the following Miller's functions f_{4T}, f_{3T}, $f_{iT\pm P}$, f_{2^kT}, f_{3^kT}, f_{6T} etc. In this paper, efficient algorithms to compute some of these Miller's functions are presented. We propose a useful fact and use it to simplify f_{m+n}. The simplified formula of f_{m+n} and new point multiplication algorithms make our algorithms more efficient.

This paper is organized as follows. Some essential concepts to discuss pairings are reviewed in Section 2. In Section 3, some efficient algorithms to compute different Miller's paths are described. An example using those results is given in Section 4. And finally conclusions are proposed in Section 5.

2 Preliminaries

In this section, the basic arithmetic of elliptic curves, the Tate pairing and Miller's algorithm are described briefly. The readers are referred to [1] and [25] for more details. Throughout this paper, the base field K is always assumed to be the finite field $\mathbb{F}_q$, where $q = p^m$ with $p > 3$, and $\overline{K}$ is the algebraic closure of K.

An elliptic curve E over K is a curve that is given by an equation of the form

$$y^2 = x^3 + ax + b, \tag{1}$$

where $a, b \in K$ and $4a^3 + 27b \neq 0$. Let $E(K)$ denote the set of points $(x, y) \in K^2$ which satisfy the equation (1), along with a "point at infinity", denoted by $\mathcal{O}$. For any positive integer k, $F = \mathbb{F}_{q^k}$ is an extension field of K. Then $E(F)$ denotes the set of $(x, y) \in F^2$ that satisfy (1), along with $\mathcal{O}$. There is an abelian group structure in $E(K)$. The addition formulas for affine coordinates are the followings. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two elements of $E(K)$ that are different from $\mathcal{O}$. Then the addition $P_3 = P_1 + P_2 = (x_3, y_3)$ is defined by $x_3 = \lambda_{P_1,P_2}^2 - x_1 - x_2$ and $y_3 = \lambda_{P_1,P_2}(x_1 - x_3) - y_1$, where $\lambda_{P_1,P_2} = (y_2 - y_1)/(x_2 - x_1)$ is the slope of the line through P_1 and P_2 for $P_1 \neq \pm P_2$ and $\lambda_{P_1,P_2} = (3x_1^2 + a)/(2y_1)$ is the slope of the tangent line at P_1 for $P_1 = P_2$.

Let ℓ be a positive integer and let $E[\ell]$ (resp. $E(\mathbb{F}_q)[\ell]$) be the set of points $P \in E(\overline{\mathbb{F}_q})$ (resp. $P \in E(\mathbb{F}_q)$) satisfying $\ell P = \mathcal{O}$. Let P be a point on $E(\mathbb{F}_q)$ of order r, the point P, or the cyclic subgroup $\langle P \rangle$, or the integer r is said to have *embedding degree* k for some positive integer k if $r \mid q^k - 1$ and $r \nmid q^s - 1$ for any $0 < s < k$. The group $E(\mathbb{F}_q)$ is (isomorphic to) a subgroup of $E(\mathbb{F}_{q^k})$. Let $P \in E(\mathbb{F}_q)$ be a point of order r such that P has embedding degree k. Then $E(\mathbb{F}_{q^k})$ contains a point Q of the same order r but linearly independent with P (see [1]).

Definition 1. *Let r be a positive integer coprime to q and let k be the embedding degree to r. The Tate pairing $\tau_r\langle\cdot,\cdot\rangle$ is a map $\tau_r\langle\cdot,\cdot\rangle : E(\mathbb{F}_q)[r]\times E(\mathbb{F}_{q^k})[r] \to \mathbb{F}^*_{q^k}$ defined by $\tau_r(P,Q) = f_{rP}(D_Q)^{(q^k-1)/r}$ for any $P \in E(\mathbb{F}_q)[r]$ and $Q \in E(\mathbb{F}_{q^k})[r]$, where f_{rP} is a rational function satisfying $(f_{rP}) = r(P) - r(\mathcal{O})$, and $D_Q \sim (Q) - (\mathcal{O})$ such that (f_P) and D_Q have disjoint supports.*

From [2], we know that $\tau_r(P,Q)$ can be reduced to $\tau_r(P,Q) = f_{rP}(Q)^{(q^k-1)/r}$ when $r \nmid (p-1)$, $Q \neq \mathcal{O}$, and $k > 1$. Noting that in most cryptographic primitives, r is set to be a prime such that $r \mid \sharp E(\mathbb{F}_q)$. Furthermore, in practice, r is at least larger than 160 bits.

In order to compute the Tate pairing $\tau_r(P,Q)$, we need to find the function f_P and then evaluate its value at Q. The algorithm, proposed by Miller [20], and then called Miller's algorithm, can be used to compute the Tate paring. Denoted by $l_{U,V}$ the equation of the line through points $U, V \in E(\mathbb{F}_q)$. Naturally, if $U = V$, then $l_{U,V}$ is the equation of the tangent line at U, and if either U or V is the point at infinity $\mathcal{O}$, then $l_{U,V}$ represents the vertical line through the other point. Furthermore, for simplicity, we write l_U instead of $l_{U,-U}$. For a point P on elliptic curve E, define a Miller's function with parameter $n \in \mathbb{N}$ to be a rational function f_{nP} (or simply f_n) on E such that $(f_n) = n(P) - (nP) - (n-1)(\mathcal{O})$.

Theorem 1 ([1] Miller's formula). *Let $P \in E(\mathbb{F}_q)$, n be an integer and let f_n be the function with divisor $(f_n) = n(P) - (nP) - (n-1)(\mathcal{O})$. Then for any $m, n \in \mathbb{Z}$,*

$$f_{m+n}(Q) = f_m(Q) \cdot f_n(Q) \cdot \frac{l_{mP,nP}(Q)}{l_{(m+n)P}(Q)}.$$

We can use the Miller's algorithm to compute f_n and then evaluate the Tate pairing. The standard double-and-add method to compute the Tate pairing is the following algorithm.

Algorithm 1. *Miller's algorithm:*

Input: $t = \log r$, $r = (r_t = 1, r_{t-1}, \ldots, r_0)_2$, P, Q.
Output: $f_{rP}(Q)^{(q^k-1)/r}$.

1: $T = P$, $f = 1$
2: *For* $i = t-1$ *downto* 0 *do*
3: $f \leftarrow f^2 \cdot l_{T,T}(Q)/l_{2T}(Q)$ *and* $T \leftarrow 2T$;
4: *if* $r_i = 1$, *then* $f \leftarrow f \cdot l_{T,P}(Q)/l_{T+P}(Q)$ *and* $T \leftarrow T + P$
5: Return $f^{(q^k-1)/r}$

Definition 2. *Let $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^k})$, a Miller's path about Q from $(n_1P, n_2P, \cdots, n_sP)$ to $(n_1+n_2+\cdots+n_s)P$ is an algorithm* **A** *which can output $(n_1+n_2+\cdots+n_s)P$ and $f_{(n_1+n_2+\cdots+n_s)}(Q)$ when input $(n_1P, n_2P, \cdots, n_sP)$ and $f_{n_1}, f_{n_2}, \cdots, f_{n_s}$. When there is no confusion, this algorithm* **A** *is said to be a Miller's path to $(n_1P, n_2P, \cdots, n_sP)$.*

The computational cost (timing) of scalar multiplications and Tate pairings on elliptic curve operations depend on the cost of the arithmetic operations that have to be performed in the underlying field. In general, among these arithmetics, a field squaring, a field multiplication and a field inversion are more expensive than other field arithmetics, such as a field addition and a field subtraction. So we only take into account the cost of inversion, multiplication, and squaring in the field $\mathbb{F}_q$, which we denote by I, M and S, respectively, while in the extension field $\mathbb{F}_{q^k}$, those costs are denoted by I_k, M_k and S_k, respectively. Generally it is assumed that $1S = 0.8M$, $1M_k = k^2M$ and $1I_k = k^2M + I$. Also the multiplication between elements in $\mathbb{F}_q^*$ and $\mathbb{F}_{q^k}^*$ costs kM.

3 Computation of Miller's Paths

In the process of computing Tate pairings, we often need to compute $(f_{mP+nT}, mP+nT)$ from (f_P, P) and (f_T, T). A Miller's path is to compute $mP+nT$ and f_{mP+nT} at the same time. Different Miller's paths have different computation costs. A main problem is to find an optimized Miller's path so that we can quicken the computation of Tate pairings.

Throughout this section, we assume that $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^k})$. Moreover, though we don't use the denominators discarded method [1] in here, our strategies can also be used in those algorithms there.

3.1 Miller's Path to 4T

In this subsection, an improved method for obtaining $(f_{4T}, 4T)$ from (f_T, T) is given. Firstly, some facts about elliptic curves will be given from which one can simplify the computation of f_{4T}.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on E, Set $P_1 + P_2 = P_3 = (x_3, y_3)$. Let $l_{P_1,P_2} : y - \lambda_{P_1,P_2}(x - x_1) - y_1 = 0$ be the equation of the line through P_1 and P_2 and $l_{P_1+P_2} : x - x_3 = 0$ be the vertical line through the point $P_1 + P_2$. If $P_1 = P_2$ then l_{P_1} is the tangent line at P_1, and if $P_1 + P_2 = \mathcal{O}$ then we take $l_{P_1+P_2} = 1$.

Lemma 1. *For $P_1, P_2 \in E(\mathbb{F}_q)$, let $P_1 + P_2 = P_3$. Then we have $l_{P_1,P_2} \cdot l_{-P_1,-P_2} = l_{P_1} \cdot l_{P_2} \cdot l_{P_3}$, i.e.,*

$$(y - \lambda_{P_1,P_2}(x - x_1) - y_1)(y + \lambda_{P_1,P_2}(x - x_1) + y_1) = (x - x_1)(x - x_2)(x - x_3).$$

Proof. The divisor of the function l_{P_1,P_2} is $(l_{P_1,P_2}) = (P_1)+(P_2)+(-P_3)-3(\mathcal{O})$. Since $-P_1 = (x_1, -y_1)$, we have $(y + \lambda_{P_1,P_2}(x - x_1) + y_1) = (-P_1) + (-P_2) + (P_3) - 3(\mathcal{O})$. Thus

$$\begin{aligned}
&\operatorname{div}((y - \lambda_{P_1,P_2}(x - x_1) - y_1)(y + \lambda_{P_1,P_2}(x - x_1) + y_1)) \\
&= (P_1) + (P_2) + (-P_3) - 3(\mathcal{O}) + (-P_1) + (-P_2) + (P_3) - 3(\mathcal{O}) \\
&= (P_1) + (-P_1) - 2(\mathcal{O}) + (P_2) + (-P_2) - 2(\mathcal{O}) + (P_3) + (-P_3) - 2(\mathcal{O}) \\
&= \operatorname{div}(x - x_1) + \operatorname{div}(x - x_2) + \operatorname{div}(x - x_3) \\
&= \operatorname{div}((x - x_1)(x - x_2)(x - x_3)).
\end{aligned}$$

From $\mathrm{div}(f) = 0$ if and only if f is a constant function, we have that

$$(y - \lambda_{P_1,P_2}(x - x_1) - y_1)(y + \lambda_{P_1,P_2}(x - x_1) + y_1) = c \cdot (x - x_1)(x - x_2)(x - x_3)$$

for some constant $c \in K$. But since the coefficient of x^3 is 1 in both sides we have $c = 1$. □

Remark 1. From the lemma 1, we know $f_{m-n}(Q) = \frac{f_m}{f_n}(Q) \cdot \frac{l_{mP}}{l_{-mP,nP}}(Q)$, Thus we have $f_{2m-1}(Q) = f_m^2(Q) \cdot \frac{l_{mP,mP}}{l_{-2mP,P}}(Q)$.

We now describe an improved method for obtaining $(f_{4T}, 4T)$ from (f_T, T). Note that for any points T and S, we have $l_{-T,-S}(Q) = l_{T,S}(-Q)$. By Lemma 1, we have

$$f_{4T} = \left(f_T^2 \cdot \frac{l_{T,T}}{l_{2T}}\right)^2 \cdot \frac{l_{2T,2T}}{l_{4T}} = f_T^4 \cdot \frac{l_{T,T}^2}{l_{2T}^2} \cdot \frac{l_{2T,2T}}{l_{4T}} = f_T^4 \cdot \frac{l_{T,T}^2}{l_{-2T,-2T}}.$$

Let $T = (x_1, y_1)$, $2T = (x_2, y_2)$ and $4T = (x_4, y_4)$. Then $x_2 = \lambda_{T,T}^2 - 2x_1$, $y_2 = \lambda_{T,T}(x_1 - x_2) - y_1$; $x_4 = \lambda_{2T,2T}^2 - 2x_2$ and $y_4 = \lambda_{2T,2T}(x_2 - x_4) - y_2$. Set $Q = (x, y)$. We have

$$f_{4T}(Q) = f_T^4 \cdot \frac{l_{T,T}^2}{l_{-2T,-2T}}(Q) = \frac{f_T^4 \cdot l_{T,T}^2(Q)}{l_{2T,2T}(-Q)} = f_T^4 \cdot \frac{[y - y_1 - \lambda_{T,T}(x - x_1)]^2}{y + y_2 + \lambda_{2T,2T}(x + x_2)}.$$

Furthermore, let λ be defined as $\lambda = 3x_1^2 + a$. Then we have

$$\frac{1}{2y_2} = \frac{(2y_1)^3}{2\lambda(3x_1 \cdot (2y_1)^2 - \lambda^2) - (2y_1)^4}.$$

Thus

$$\lambda_{2T,2T} = (3x_2^2 + a) \cdot \frac{(2y_1)^4}{2y_1[2\lambda(3x_1 \cdot (2y_1)^2 - \lambda^2) - (2y_1)^4]};$$
$$\lambda_{T,T} = (3x_1^2 + a) \cdot \frac{2\lambda(3x_1 \cdot (2y_1)^2 - \lambda^2) - (2y_1)^4}{2y_1[2\lambda(3x_1 \cdot (2y_1)^2 - \lambda^2) - (2y_1)^4]}.$$

So we have the following algorithm to compute $4T$ and f_{4T} from T and f_T.

In the above algorithm, we need $I + 7S + 9M$ to compute λ_1, λ_2 and (x_4, y_4). It is cheaper than two doubling method which cost $2I + 4S + 4M$ for in the general finite field $I/M \geq 10$. Also it is better than the algorithm in [6] where the cost is $I + 9S + 9M$. Steps 7 and 8 cost $I_k + 2S_k + 2M_k + 2kM$ when $Q = (x, y)$, $x, y \in \mathbb{F}_{q^k}$. For general algorithm, the cost is $2I_k + S_k + 2M_k + 2kM$ (see [21]). So this algorithm is better than known algorithms not only for $4T$, but also for f_{4T}.

Algorithm 2. *(Path to 4T algorithm):*

———————————————————————————-

Input: $T = (x_1, y_1), Q = (x, y),\ f_T$.
Output: $f_{4T}(Q),\ 2T,\ 4T = (x_4, y_4)$.

———————————————————————————-

1: $t_1 = 3x_1^2 + a;\ t_2 = 2y_1; t_3 = (t_2^2)^2;\ t_4 = x_1 \cdot t_3;$
2: $t_5 = 2t_1 \cdot (3t_4 - t_1^2) - t_3;$
3: $t_6 = (t_2 \cdot t_5)^{-1};$
4: $\lambda_1 = t_1 \cdot t_5 \cdot t_6;\ x_2 = \lambda_1^2 - 2x_1;\ y_2 = \lambda_1(x_1 - x_2) - y_1;$
5: $\lambda_2 = (3x_2^2 + a) \cdot t_3 \cdot t_6;\ x_4 = \lambda_2^2 - 2x_2;\ y_4 = \lambda_2(x_2 - x_4) - y_2;$
6: $f_1 = y - y_1 - \lambda_1(x - x_1);\ f_2 = y + y_2 + \lambda_2(x + x_2);$
7: $f_{4T}(Q) = (f_T(Q)^2 \cdot f_1(Q))^2 \cdot f_2(Q)^{-1};$
8: Return $f_{4T}(Q),\ 2T,\ 4T = (x_4, y_4)$.

———————————————————————————-

3.2 Miller's Path to $2T \pm P$

In this subsection, we will describe the efficient Miller's path to $(f_{2T+P}, 2T + P)$ and $(f_{2T-P}, 2T - P)$.

Miller's Path to $2T + P$. We first give an efficient Miller's path to $(f_{2T+P}, 2T + P)$ from (f_T, T) and (f_P, P) when $T \neq P$.

First noting that

$$f_{2T+P} = f_{T+P} \cdot \frac{f_T \cdot l_{T+P,T}}{l_{2T+P}} = \frac{f_T \cdot f_P \cdot l_{T,P}}{l_{T+P}} \cdot \frac{f_T \cdot l_{T+P,T}}{l_{2T+P}} = \frac{f_T^2 \cdot f_P}{l_{2T+P}} \cdot \frac{l_{T,P} \cdot l_{T+P,T}}{l_{T+P}}.$$

Set $T = (x_1, y_1)$, $P = (x_2, y_2)$, $T + P = (x_3, y_3)$ and $2T + P = (x_4, y_4)$. From [12] we can replace $(l_{T,P} \cdot l_{T+P,T})/l_{T+P}$ by the following parabolas formula:

$$\frac{l_{T,P} \cdot l_{T+P,T}}{l_{T+P}} = (x - x_1)(x + x_1 + x_3 + \lambda_{T,P}\lambda_{T,T+P}) - (\lambda_{T,P} + \lambda_{T,T+P})(y - y_1).$$

Furthermore, $\lambda_{T,T+P}$ can be expanded as follows:

$$\begin{aligned}\lambda_{T,T+P} = \frac{y_3 - y_1}{x_3 - x_1} &= \frac{(x_1 - x_3)\lambda_{T,P} - 2y_1}{\lambda_{T,P}^2 - 2x_1 - x_2} \\ &= \frac{2y_1(x_2 - x_1)^3 - (y_2 - y_1)\left[(x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2\right]}{(x_2 - x_1)\left[(x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2\right]}.\end{aligned}$$

Since

$$\lambda_{T,P} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{(y_2 - y_1)\left[(x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2\right]}{(x_2 - x_1)\left[(x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2\right]}.$$

So we need only to compute one inversion $\left\{(x_2 - x_1)\left[(x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2\right]\right\}^{-1}$ in order to compute $\lambda_{T,P}$ and $\lambda_{T,T+P}$ simultaneously. Thus we have the following formulas.

$$\begin{aligned}
\lambda_{T,P} &= \left\{(x_2-x_1)\left[(x_2-x_1)^2(2x_1+x_2)-(y_2-y_1)^2\right]\right\}^{-1} \\
&\quad \cdot (y_2-y_1)\left[(x_2-x_1)^2(2x_1+x_2)-(y_2-y_1)^2\right]; \\
\lambda_{T,T+P} &= \left\{(x_2-x_1)\left[(x_2-x_1)^2(2x_1+x_2)-(y_2-y_1)^2\right]\right\}^{-1} \cdot 2y_1(x_2-x_1)^3-\lambda_{T,P}; \\
x_4 &= (\lambda_{T,T+P}-\lambda_{T,P})(\lambda_{T,T+P}+\lambda_{T,P})+x_2; \\
y_4 &= (x_1-x_4)\lambda_{T,T+P}-y_1.
\end{aligned}$$

Since $x_3 = \lambda_{T,P}^2 - x_1 - x_2$, the new parabolas formula is

$$\frac{l_{T,P}\cdot l_{T+P,P}}{l_{T+P}} = (x-x_1)(\lambda_{T,P}(\lambda_{T,P}+\lambda_{T,T+P})-x_2+x)-(\lambda_{T,P}+\lambda_{T,T+P})(y-y_1).$$

This procedure is described by the following algorithm.

Algorithm 3. *(Path to $2T+P$ algorithm):*

Input: $T=(x_1,y_1)$, $P=(x_2,y_2)$, f_T, f_P *and* $Q=(x,y)$
Output: $f_{2T+P}(Q)$ *and* $2T+P=(x_4,y_4)$.

1: $t_1=(x_2-x_1)^2(2x_1+x_2)-(y_2-y_1)^2$, $t_2=(x_2-x_1)t_1$, $t_3=t_2^{-1}$;
2: $\lambda_1=(y_2-y_1)t_1t_3$, $\lambda_2=t_3\cdot 2y_1(x_2-x_1)^2(x_2-x_1)-\lambda_1$;
3: $x_4=(\lambda_2-\lambda_1)(\lambda_2+\lambda_1)+x_2$;
4: $y_4=(x_1-x_4)\lambda_2-y_1$;
5: $f_{2T+P}(Q)=\dfrac{f_T^2\cdot f_P}{l_{2T+P}}\cdot[(x-x_1)(\lambda_1(\lambda_1+\lambda_2)-x_2+x)-(\lambda_1+\lambda_2)(y-y_1)](Q)$
6: return $f_{2T+P}(Q)$, $2T+P$.

In Algorithm 2, we require $1I+2S+10M$ to compute $2T+P$ and $x_1+x_3+\lambda_{T,P}\lambda_{T,T+P}$, while in [12], $2I+2S+4M$ times are required to compute them. We save $1I-6M$ field computations.

Miller's Path to $2T-P$. Now we describe an efficient Miller's path to $(f_{2T-P}, 2T-P)$ from (f_T,T) and (f_P,P) when $T\neq P$. We can use $2T+(-P)$ to get f_{2T-P} by Algorithm 3, but here we describe a direct path to f_{2T-P}.

By Remark 1 in section 3.1 we know

$$f_{2T-P}(Q) = f_T^2(Q)\cdot\frac{l_{T,T}}{l_{-2T,P}}(Q).$$

Let $P=(x_P,y_P)$,$T=(x_T,y_T)$, $2T=(x_{2T},y_{2T})$ and $2T-P=(x_{2T-P},y_{2T-P})$. Then $x_{2T}=\lambda_{T,T}^2-2x_T$, $y_{2T}=\lambda_{T,T}(x_T-x_{2T})-y_T$; $x_{2T-P}=\lambda_{2T,-P}^2-x_{2T}-x_P$ and $y_{2T-P}=\lambda_{2T,-P}(x_P-x_{2T-P})+y_P$. Set $Q=(x,y)$. We have

$$f_{2T-P}(Q) = f_T^2\cdot\frac{l_{T,T}}{l_{-2T,P}}(Q) = f_T^2(Q)\cdot\frac{y-y_T-\lambda_{T,T}(x-x_T)}{y-y_P+\lambda_{2T,-P}(x-x_P)}.$$

Furthermore, let λ be defined as $\lambda = 3x_T^2 + a$. Then we have

$$\lambda_{T,T} = \frac{(3x_T^2 + a) \cdot [\lambda^2 - (2x_T + x_P)(2y_T)^2]}{2y_T[\lambda^2 - (2x_T + x_P)(2y_T)^2]};$$
$$\lambda_{2T,-P} = \frac{(2y_T)^3(y_{2T} + y_P)}{2y_T[\lambda^2 - (2x_T + x_P)(2y_T)^2]}.$$

Therefore, when $Q = (x, y)$ and $x \in \mathbb{F}_{q^k}$, $y \in \mathbb{F}_{q^k}$, to complete the Miller's path $(f_{2T-P}, 2T - P)$ from (f_T, T) and (f_P, P) we need only $1I + 5S + (2k + 9)M + I_k + S_k + 2M_k$. However, we need $1I + 2S + (k + 10)M + I_k + S_k + 4M_k$ when use the Algorithm 3 to compute $(f_{2T-P}, 2T - P)$.

3.3 Miller's Path to $3T$

In this subsection, a Miller's path to $(f_{3T}, 3T)$ from (f_T, T) is given. By the Miller's formula we have

$$f_{3T} = f_T^3 \cdot \frac{l_{T,T}}{l_{2T}} \cdot \frac{l_{T,2T}}{l_{3T}}.$$

Let $T = (x_1, y_1)$, $2T = (x_2, y_2)$, and $3T = (x_3, y_3)$. Then

$$\frac{l_{T,T} \cdot l_{T,2T}}{l_{2T}} = (x - x_1)(x + x_1 + x_2 + \lambda_{T,T}\lambda_{T,2T}) - (\lambda_{T,T} + \lambda_{T,2T})(y - y_1).$$

By $x_2 = \lambda_{T,T}^2 - 2x_1$, we have the following parabolas formula

$$\frac{l_{T,T} \cdot l_{T,2T}}{l_{2T}} = (x - x_1)(x + \lambda_{T,T}^2 - x_1 + \lambda_{T,T}\lambda_{T,2T}) - (\lambda_{T,T} + \lambda_{T,2T})(y - y_1).$$

Noting that $x_3 = (\lambda_{T,2T} - \lambda_{T,T})(\lambda_{T,2T} + \lambda_{T,T}) + x_1$, we need only $\lambda_{T,T}$, $\lambda_{T,2T}$ and $3T$ to compute $(f_{3T}, 3T)$. From

$$\lambda_{T,2T} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\lambda_{T,T}(x_1 - (\lambda_{T,T}^2 - 2x_1)) - 2y_1}{(\lambda_{T,T}^2 - 2x_1) - x_1} = \frac{2y_1}{3x_1 - \lambda_{T,T}^2} - \lambda_{T,T}$$
$$= \frac{(2y_1)^3}{(2y_1)^2(3x_1) - (3x_1^2 + a)^2} - \lambda_{T,T}$$

and

$$\lambda_{T,T} = \frac{3x_1^2 + a^2}{2y_1} = \frac{3x_1^2 + a^2}{2y_1} \cdot \frac{(2y_1)^2(3x_1) - (3x_1^2 + a)^2}{(2y_1)^2(3x_1) - (3x_1^2 + a)^2},$$

we have the follow algorithm:

In step 5 of Algorithm 4, $I_k + S_k + 4M_k + (k+1)M$ costs are needed to compute $f_{3T}(Q)$. However the general double-add algorithm needs $I_k + S_k + 7M_k + 2kM$. Our algorithm saves $3M_k + (k-1)M$ field computations.

Algorithm 4. *(Path to 3T algorithm):*

Input: $T = (x_1, y_1)$, f_T, $Q = (x, y)$.
Output: $f_{3T}(Q)$ *and* $3T = (x_3, y_3)$.
1: $t_1 = (2y_1)^2$, $t_2 = t_1^2$, $t_3 = 3x_1^2 + a$;
2: $t_4 = 3x_1 \cdot t_1 - t_3$, $t_5 = (2y_1 \cdot t_4)^{-1}$, $\lambda_1 = t_3 t_4 t_5$, $\lambda_2 = t_2 t_5 - \lambda_1$;
3: $x_3 = (\lambda_2 - \lambda_1)(\lambda_2 + \lambda_1) + x_1$;
4: $y_3 = (x_1 - x_3)\lambda_2 - y_1$;
5: $f_{3T}(Q) = \dfrac{f_T^3(Q)}{x - x_3} \cdot [(x - x_1)(\lambda_1(\lambda_1 + \lambda_2) - x_1 + x) - (\lambda_1 + \lambda_2)(y - y_1)](Q)$;
6: Return $f_{3T}(Q)$, $3T$.

3.4 Miller's Path to $6T$

In this subsection, the Miller's path to $(f_{6T}, 6T)$ from (f_T, T) is considered. In the first we see the Miller's path $[(f_T, T) \to (f_{2T}, 2T)$ and $(f_T, T) \to (f_{4T}, 4T)] \to (f_{2T+4T}, 2T + 4T)$. By Miller's formula and the results in Section 3.1, we have

$$f_{6T} = f_{4T} \cdot f_{2T} \cdot \frac{l_{2T,4T}}{l_{6T}} = f_T^6 \cdot \frac{l_{T,T}^2}{l_{-2T,-2T}} \cdot \frac{l_{T,T}}{l_{2T}} \cdot \frac{l_{2T,4T}}{l_{6T}} = f_T^6 \cdot \frac{l_{T,T}^2}{l_{-2T,-2T}} \cdot \frac{l_{T,T} \cdot l_{2T,4T} \cdot l_{4T}}{l_{2T} \cdot l_{6T} \cdot l_{4T}}.$$

So by Lemma 1,

$$f_{6T} = f_T^6 \cdot \frac{l_{T,T}^3}{l_{-2T,-2T}} \cdot \frac{l_{2T,4T} \cdot l_{4T}}{l_{2T,4T} \cdot l_{-2T,-4T}} = f_T^6 \cdot \frac{l_{T,T}^3}{l_{-2T,-2T}} \cdot \frac{l_{4T}}{l_{-2T,-4T}}.$$

Let $T = (x_1, y_1)$, $2T = (x_2, y_2)$, $4T = (x_4, y_4)$ and $Q = (x, y)$. Then

$$\begin{aligned} f_{6T}(Q) &= (f_T^2 l_{T,T})^3 \cdot \frac{l_{4T}}{l_{-2T,-2T} \cdot l_{-2T,-4T}}(Q) \\ &= \frac{(f_T^2 l_{T,T})^3 \cdot (x - x_4)}{[y + y_2 + \lambda_{2T,2T}(x + x_2)] \cdot [y + y_2 + \lambda_{2T,4T}(x + x_2)]}. \end{aligned}$$

Secondly, there is another way to $(f_{6T}, 6T)$ from (f_T, T) as $(f_T, T) \to (f_{3T}, 3T) \to (f_{3T+3T}, 6T)$. Similarly, we have

$$\begin{aligned} f_{6T}(Q) &= f_{3T}^2 \cdot \frac{l_{3T,3T}}{l_{6T}}(Q) \\ &= f_T^6 \cdot \frac{[(x - x_1)(\lambda_{T,T}(\lambda_{T,T} + \lambda_{T,2T}) - x_1 + x) - (\lambda_{T,T} + \lambda_{T,2T})(y - y_1)]^2}{l_{3T}^2} \cdot \frac{l_{3T,3T}}{l_{6T}}(Q) \\ &= f_T^6 \cdot \frac{[(x - x_2)(\lambda_{T,T}(\lambda_{T,T} + \lambda_{T,2T}) - x_2 + x) - (\lambda_{T,T} + \lambda_{T,2T})(y - y_2)]^2}{l_{-3T,-3T}(Q)}. \end{aligned}$$

By comparing with their costs, the second way $(f_T, T) \to (f_{3T}, 3T) \to (f_{3T+3T}, 6T)$ is more efficient to compute $(f_{6T}, 6T)$. From this way, an algorithm to compute f_{6T} and $6T$ can be gotten.

3.5 Miller's Path to $iT \pm P$ for $i = 3, 4, 6$.

In this subsection, we think about the Miller's path to $iT \pm P$ from (f_T, T) and (f_P, P), where $i = 3$, 4 and 6. Firstly, let us see an optimized one to $(f_{3T+P}, 3T + P)$.

There are following four ways to get the Miller's path to $3T + P$ from (f_T, T) and (f_P, P). The first is as $(f_T, T) \to (f_{3T}, 3T) \to (f_{3T+P}, 3T + P)$. The second is as $[(f_T, T) \to (f_{2T}, 2T)$ and$(f_P, P) \to (f_{P+T}, P+T)] \to (f_{2T+(T+P)}, 2T+(T+P))$. The third is as $(f_T, T) \to (f_{2T+P}, 2T+P) \to (f_{(2T+P)+T}, (2T+P)+T)$, and the fourth is as $(f_T, T) \to (f_{T+P}, T+P) \to (f_{2T+(T+P)}, 2T+(P+T)$. Comparing with the costs of these four ways we know that the second way is more efficient than the others. Therefore, we have $f_{3T+P}(Q) = f_T^3 \frac{l_{T,T} \cdot l_{T,P}}{l_{-2T,-T-P}}(Q)$. The details are omitted.

Similarly, for the Miller's paths to $4T + P$, We have the ways $(f_T, T) \to (f_{4T}, 4T) \to (f_{4T+P}, 4T + P)$ and $(f_T, T) \to (f_{2T}, 2T) \to (f_{2(2T)+P}, 2(2T) + P)$. Comparing with the costs of these 2 ways we know that the first way is more efficient than the second. Similarly, for the Miller's paths to $6T + P$, the way $(f_T, T) \to (f_{3T}, 3T) \to (f_{2(3T)+P,2(3T)+P}$) is more efficient. From these results, algorithms to compute f_{3T+P}, f_{4T+p} and f_{6T+P} can be obtained.

The Miller's path to $iT - P$ are used to the algorithms using addition-subtraction chains. Under conditions, direct compute f_{iT-P} is more efficient. For example, for the path to $3T - P$ we have $f_{3T-P}(Q) = f_T^3[\frac{l_{T,T} l_{T,2T}}{l_{2T}}] \frac{1}{l_{-3T,P}}(Q)$. For the path to $4T - P$ we have $f_{4T-P}(Q) = f_{4T}(Q) \cdot \frac{l_{4T}}{l_{-4T,P}}(Q)$.

4 Example

As an example we describe a new algorithm to compute the Tate pairing in this section.

Algorithm 5. *Signed-Radix-2 Miller's Algorithm*

Input: $r = (r_t = 1, 0^{h_{t-1}}, r_{t-1}, \cdots, 0^{h_0}, r_0)_2$, $P = (x_P, y_P)$, $Q = (x, y)$
Output: $f_{rP}(Q)^{(q^k-1)/r} \in \mathbb{F}_{q^k}^*$

1: $T = P$, $f = 1$;
2: For $i = t - 1$ *downto* 0 *do*
3: *if* h_i *is even, then use the Algorithm 2 to compute* $f_{4^{h_i/2}T}$ *and* $4^{h_i/2}T$; *set* $T = 4^{h_i/2}T$ *then to compute* $f_{T+P}(Q)$ *if* $r_{i+1} = 1$ *or* $f_{T-P}(Q) = f_T \frac{l_T}{l_{-T,P}}(Q)$ *if* $r_{i+1} = -1$;
4: *if* h_i *is odd, then use the Algorithm 2 to compute* $f_{4^{(h_i-1)/2}T}$ *and* $4^{(h_i-1)/2}T$; *set* $T = 4^{(h_i-1)/2}T$ *then to compute* $f_{2T+P}(Q)$ *if* $r_{i+1} = 1$ *or* $f_{2T-P}(Q)$ *if* $r_{i+1} = -1$;

5: EndFor
6: Return $f^{(q^k-1)/r}$.

For an integer r, consider the signed radix-2 representation of r. This representation is advantaged for the non-zero digits is one-third of the length of the representation on average. We write the radix-2 representation of r as $r = (r_t = 1, 0^{h_{t-1}}, r_{t-1}, \cdots, 0^{h_0}, r_0)_2$, where the r_i is 1 or -1. The above algorithm is a modified Miller's algorithm with the signed radix-2 representation.

5 Conclusion

In this paper, several strategies to compute the Tate pairing efficiently are given. The concept of Miller's path which let us consider the scalar multiplications and the computation of line functions in the same time is proposed. A useful fact is stated in Lemma 1 and simple formulas to compute f_{4T}, f_{2T-P} and f_{6T} etc. are presented. Similar idea as in [12] is used to compute f_{3T} to simplify the computation. These algorithms are also used to compute $f_{3T\pm P}$, $f_{4T\pm P}$ and f_{6T+P}. Furthermore, these methods can also be applied to other algorithms for the computation of Tate pairings. In practical applications, the computation of f_{2^kT} or f_{3^kT} are also needed. Certainly, iterative method can be used to reduce some cost, but it only reduce the cost of scalar multiplications. So, how to simplify the formula of Miller's functions f_{2^kT} and f_{3^kT} are still crucial open problems.

Acknowledgements

The authors would like to thank anonymous referees for their valuable comments and suggestions.

References

1. P.S.L.M. Barreto, H.Y. Kim, B. Lynn and M. Scott. Efficient algorithms for pairing-based cryptosystems, Advances in Cryptology–Crypto'2002, LNCS 2442, 354-368, Spriger-Verlag, 2002.
2. P.S.L.M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups, SAC 2003, LNCS 3006, 17-25, Spinger-Verlag, 2004.
3. I.F.Blake, V.Kumar Murty and Guangwu Xu. Refinements of Miller's algorithm for computing the Weil/Tate pairing, J.Algorithms 58(2), 134-149, 2006.
4. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing, SIAM Journal of Computing 32, 586-615, 2003.
5. D. Boneh, B. Lynn and H. Shacham. Short signatures from the Weil pairing. Advances in Cryptology–Asiacrypt 2001, LNCS 2248, 514–532, Springer-Verlag, 2001.
6. M. Ciet, M. Joye, K. Lauter, and P.L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography, Design, Codes and Cryptography 39(2), 189-206, 2006.

7. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates, Advances in Cryptology–ASIACRYPT'98, LNCS 1514, 51–65, Springer-Verlag, 1998.
8. S. Duquesne and G. Frey. Background on pairings, In Handbook of elliptic and hyperelliptic curve cryptography, Discrete Math. Appl., 115-124, Chapman & Hall/CRC, Boca Raton, FL, 2006.
9. S. Duquesne and G. Frey. Implementation of pairings, In Handbook of elliptic and hyperelliptic curve cryptography, Discrete Math. Appl., 389-404, Chapman & Hall/CRC, Boca Raton, FL, 2006.
10. S. Duquesne and T. Lange. Pairing-based cryptography, In Handbook of elliptic and hyperelliptic curve cryptography, Discrete Math. Appl., 573-590, Chapman & Hall/CRC, Boca Raton, FL, 2006.
11. R. Dutta, R. Barua and P. Sarkar. Pairing-based cryptography: a survey, Cryptology ePrint Archive, Report 2004/064, 2004.
12. K. Eisenträger, K. Lauter and P.L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation, CT-RSA 2003, LNCS 2612, 343-354, Springer-Verlag, 2003.
13. S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing, LNCS 2369, 324-337, Springer Verlag, 2002.
14. S. Galbraith. Pairings, Advances in elliptic curve cryptography, London Math. Soc. Lecture Note Ser. 317, 183–213, Cambridge Univ. Press, 2005.
15. Robert Granger, Dan Page and Nigel Smart. High security pairing-based cryptography revisited, LNCS 4076, 480–494, Springer Verlag, 2006.
16. D. Hankerson, A.J. Menezesand S. Vanstone, Guide to Elliptic Curve Cryptography, Springer Verlag, 2004.
17. T. Izu and T. Takagi. Efficient computations of the Tate pairing for the large MOV degrees. ICISC'02, LNCS 2587, 283-297, Springer-Verlag, 2003.
18. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels, Cryptology ePrint Archive, Report 2005/076, 2005.
19. A.J. Menezes. Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, 1993.
20. V. Miller. Short programs for functions on curves, unpublished manuscript, 1986.
21. V. Miller. The Weil pairing, and its efficient calculation, J. Cryptology 17(4), 235-261, 2004.
22. R. Sakai, K. Ohgishiand M. Kasahara. Cryptosystems based on pairing, SCIS'00, no. C20, 2000.
23. M. Scott. Computing the Tate pairing, CT-RSA, Feb., 2005, San Francisco, LNCS 3376, 293-304, Springer-Verlag, 2005.
24. M. Scott, N. Costigan and W. Abdulwahab. Implementing cryptographic pairings on smartcards, Cryptography ePrint Archive, Report 2006/144, 2006.
25. J.H. Silverman. The Arithmetic of Elliptic Curves, GTM 106, Springer-Verlag, Berlin, 1986.

A Provably Secure Blind Signature Scheme

Xiaoming Hu and Shangteng Huang

Department of Computer Science and Engineering,
Shanghai JiaoTong University, Shanghai 200240, China
huxm@sjtu.edu.cn

Abstract. Some blind signature schemes have been constructed from some underlying signature schemes, which are efficient and provably secure in the random oracle. To the best of authors' knowledge, a problem still remains: does the security of the original signature scheme, by itself, imply the security of the blind version? In this paper, we answer the question. We show if the blind factors in the blind version come from hash functions, the design of blind signature scheme can be validated in random oracle model if the original scheme is provably secure. We propose a blind version of Schnorr signature scheme and reduce the security of the proposed scheme to the security of ECDLP. What's more, the complexity of this reduction is polynomial in all suitable parameters in the random oracle.

Keywords: Blind signature, Provably secure, Polynomial reduction, Security arguments.

1 Instruction

Blind signatures, introduced by [1], provide anonymity of users in application such as electronic voting and electronic payment systems. A blind signature scheme is an interactive two-party protocol between a user and a signer. It allows the user to obtain a signature of any given message, but the signer learns neither the message nor the resulting signature. Blind signature plays a central role in building anonymous electronic cash. A lot of work has been done in field of blind signature schemes since Chaum [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19, 20, 21, 22, 23]. Several blind signature schemes [1, 4, 6, 12] have been constructed from some underlying signature schemes [24, 25, 26]. These underlying signature schemes are efficient and have been validated in the so-called random oracle model. However, a problem still remains: the security of the original signature scheme does not, by itself, imply the security of the blind version.

The random oracle [27] model is a popular alternative of provable security. Some blind signature schemes were proposed and proven secure in the model [10, 11, 12, 13, 14, 28]. In [12, 28], Pointcheval and Stern prove the security of several blind digital signatures schemes, including blind variation of the [8], [25], and [5] signature schemes. However, their security proofs, while polynomial in the size of the keys, are poly-logarithmically bounded in the number of blind digital signatures. The authors leaves an open problem that whether

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 171–180, 2007.
© Springer-Verlag Berlin Heidelberg 2007

one can achieve polynomial time both in the number of signatures obtained by the adversary and the size of the keys. Juels, et al. [29] gave a positive answer to show that security and blindness properties for blind signatures could be simultaneously defined and satisfied, assuming an arbitrary one-way trapdoor permutation family. However, their proof was based on complexity. As they had discussed, their schemes should be viewed merely as a proof of existence which pave the way for efficient future implementations. Pointcheval et al. [10] proposed a blind signature scheme based on factorization, unluckily it also need a user to make a poly-logarithmically bounded number of interactions with the signer. The less practical schemes of [11] are provably secure for a polynomial number of synchronized signer interactions, where the synchronization forces the completion of each step for all the different protocol invocations before the next step of any other invocation is started, so some restrictions apply. [13] requires a non-standard strong assumption - namely the RSA chosen-target inversion problem is hard. [14] proposed an efficient three-move blind signature scheme, which provides one more unforgeability with polynomially many signatures. However, the scheme is a specific blind signature scheme which prevents one-more unforgeability after polynomially many interactions with the signer, so it isn't a generic approach. In this paper, by using hash functions to generate the blind factors, we show that the design of blind signature scheme can be validated in random oracle model if the original scheme is provably secure in random oracle model. We propose a blind version of Schnorr signature scheme. Moreover, we show that the proposed blind signature is provably secure if ECDLP is hard, and the complexity of this reduction is polynomial in all suitable parameters.

The rest of the paper is organized as follows. In section 2 we recall some definitions for blind signatures. Section 3 proposes our blind signature scheme. Section 4 gives the proof of nonforgeablility of the proposed scheme. Section 5 concludes this paper.

2 Preliminaries

In this section we review the formal definition and the standard security notion of blind signature schemes [29].

Definition 1. *A blind digital signature scheme is a four-tuple* $(Signer, User, Gen, Verify)$.
- $Gen(1^k)$ *is a probabilistic polynomial-time key-generation algorithm that takes security parameter* k *and outputs a public and secret key pair* (pk, sk).
- $Signer(pk, sk)$ *and* $User(pk, m)$ *are a pair of polynomially-bounded probabilistic Interactive Turing machines, each of which has a public input tape, a private random tape, a private work tape, a private output tape, a public output tape, and input and output communication tapes. The random tape and the input tapes are read-only, and the output tapes are write-only. The private work tape is*

read-write. They are both given pk generated by $Gen(1^k)$ *on theirs public input tapes. Additionally, the private input tape of Signer contains the private key sk, and that for User contains message m. User and Signer engage in the signature issuing protocol and stop in polynomial-time in k. At the end of this protocol, Signer outputs either completed or not-completed on his public output tap, and User outputs either* $\perp$ *or* $\sigma(m)$ *on his private output tap.*
- Verify$(pk, m, \sigma(m))$ *is a deterministic polynomial-time algorithm. On input* $(pk, m, \sigma(m))$ *and outputs accept/reject. with the requirement that for any message m, and for all random choices of key generation algorithm, if both Signer and User follow the protocol then the Signer always outputs completed, and the output of the user is always accepted by the verification algorithm.*

Definition 2. *If Signer and User follow the signature issuing protocol, then with probability of at least* $1 - 1/k^c$ *for every constant c and sufficiently large k, Signer outputs completed and User outputs* $(m, \sigma(m))$ *that satisfies Verify* $(pk, m, \sigma(m))$ *=accepted. The probability is taken over the coin flips of Gen, Signer and User.*
A blind digital signature scheme is secure if it holds the following two properties:

Definition 3. *Let* S^* *be adversarial signer, and* u_0 *and* u_1 *are two honest users.*
- $(pk, sk) \leftarrow Gen(1^k)$.
- $m_0, m_1 \leftarrow S^*(1^k, pk, sk)$.
- Set the input tap of u_0 *and* u_1 *as follows : Let* $b \in_R \{0, 1\}, put\{m_b, m_{1-b}\}$ *on the private input tap of* u_0 *and* u_1*, respectively;Put pk on the public input taps of* u_0 *and* u_1*, respectively;Randomly select the contents of the private random tapes.*
- S^* *engages in the signature issuing protocol with* u_0 *and* u_1.
- If u_0 *and* u_1 *output valid signature* $(m_b, \sigma(m_b))$ *and* $(m_{1-b}, \sigma(m_{1-b}))$*, respectively, then send* $(\sigma(m_b), \sigma(m_{1-b}))$ *to* S^**. Give* $\perp$ *to* S^* *otherwise.*
- S^* *outputs* $b' \in \{0, 1\}$*. If* $b' = b$*, then* S^* *wins.*
A blind signature scheme is blind if all probabilistic polynomial-time algorithm S^**,* S^* *outputs* $b' = b$ *with probability at most* $1/2 + 1/k^c$ *for some constant c and sufficiently large k. The probability is taken over the flips of Gen,* S^**,* u_0 *and* u_1.

Definition 4. *Let* U^* *be adversarial user and S be an honest signer.*
- $(Step1) : (pk, sk) \leftarrow Gen(1^k)$.
- $(Step2)$*:* $U^*(pk)$ *engages in the signature issuing protocol with S in adaptive, parallel and arbitrarily interleaved way. Let L denote the number of executions, where S outputted completed in the end of Step 2.*
- $(Step3)$*:* U^* *outputs a collection* $\{(m_1, \sigma(m_1)), (m_j, (m_j))\}$ *subject to the constraint the all* $(m_i, \sigma(m_i))$ *for* $1 \leq i \leq j$ *are all accepted by verify* $(pk, m_i, \sigma(m_i))$.
A blind signature scheme is nonforgeable if the probability, taken over the coin flips of Gen, U^* *and S, that* $j > l$ *is at most* $1/k^c$ *for some constant c and sufficiently large k.*

3 The Proposed Scheme

In this section, we propose s blind signature scheme. It can be seen as a slight modification of Schnorr Blind Signature Scheme.

Setup of System Parameters. Before the whole scheme can be initialized, the following parameters over the elliptic curve domain must be known.
- A field size p, which is a large odd prime.
- Two parameters $a, b \in F_p$ to define the equation of the elliptic curve E over $F_p(y^2 = x^3 + ax + b(\text{mod } p)$ in the case $p > 3)$, where $4a^3 + 27b^2 \neq 0(\text{mod } p)$. The cardinality of E must be divisible by a large prime because of the issue of security raised by Pohlig and Hellman [30].
- A finite point P whose order is a large prime number in $E(F_p)$, where $P \neq O$, and O denotes infinity.
- The order of P is prime q.
- Two public cryptographically strong hash functions $H_1 : \{0,1\}^* \times E(F_p) \rightarrow Z_q$ and $H_2 : Z_q \times Z_q \rightarrow Z_q$. We remark that H_1 and H_2 will be viewed as random oracles in our security proof.

Setup of a Principal's public/private key. The signer picks a random number $x \in_R Z_q^*$ and computer $Q = xP$. His public key is Q and private key is x. The public keys Q must be certified by the CA.

Signature Generation
- The signer randomly chooses $d, e, r \in Z_q$, computes $U' = rP$, and sends (U', d, e) to the user.
- Blind. The user randomly chooses m_i, α_i' and $\beta_i' \in Z_q$, $1 \leq i \leq n$. He computes $\alpha_i = H_2(\alpha_i', d)$ and $\beta_i = H_2(\beta_i', e)$ as blinding factors, $U_i = U' - \alpha_i P - \beta_i Q$, $h_i = H_1(m_i, U_i)$ and $h_i' = h_i + \beta_i$, $1 \leq i \leq n$, sends to the signer n blinded candidates $h_i', 1 \leq i \leq n$.
- The signer verifies if the user constructs h_i' with the blind factors α or β which are the outputs of the random oracle H_2.The signer randomly chooses a subset of n-1 blinded candidates indices $R = i_j$, $1 \leq i_j \leq n$ for $1 \leq j \leq n$-1 and sends R to the user.
- The user reveals (m_i, α_i' and β_i') to the signer for all i in R.
- The signer verifies h_i' for all i in R. He computes $k_i = H_2(\alpha_i', d)$, $\lambda_i = H_2(\beta_i', e)$, $U_i = U' - k_i P - \lambda_i Q$, $h_i = H_1(m_i, U_i)$ and $H_i' = h_i + \lambda_i$ for all i in R. He accepts h_i' to be a valid blind message if $H_i' = h_i'$. If the signer accepts h_i' for all i in R, then the signer performs the following operations for the rest h_i', i not belong to R, denote by h', and the corresponding parameters are $(m, \alpha, \beta, U, h, h_i)$. The signer stops otherwise.
- Blind Sign. The signer sends back s', where $s' = r - xh'$.
- Unblind. He outputs a signature $\sigma = (s, h)$ where $s = s' - \alpha$. Then σ is the signature of the message m.

Note: The method which the user prepares n blinded candidates but only 1 out of n is finally used, all other n-1 are opened, verified and thrown away, inevitably

cause enormous computational and communication overhead. An alternative to cut down computational and communication overhead is as follows.

The user prepares n blinded candidates and sends them to the signer. The signer randomly chooses $n/2$ out of n to verify them. If all of them pass the verification, then the signer randomly chooses 1 out of the rest $n/2$ blinded candidates to perform blind signature. The signer stops otherwise. It is obvious that the signer's computational and communication overhead is almost half less than the original's, but we will show in Lemma3 that the probability of be caught as the user doesn't generate blind factors from the random oracle H_2, is almost the same with the original's.

Signature Verification. The verifier or recipient of the blind signature accepts the blind signature if and only if $H_1(m, sP + hQ) = h$. Signature verification works correctly because if (s, h) is a valid signature on the message m, then $sP+hQ = (s'-\alpha)P+hQ = (r-xh'-\alpha)P+hQ = rP-(h+\beta)Q-\alpha P+hQ = rP-\alpha P-\beta Q = U'-\alpha P-\beta Q = U$.

It is straightforward to prove our scheme satisfies the blindness property [25]. So in this paper, we will only show that our scheme satisfies nonforgeability property.

4 Security Proofs

In this section, we first show that the adversary should following the protocol. Next, we prove that the security of our scheme can be reduced to security of Schnorr signature scheme and further the security of ECDLP, and the complexity of the reduction is fully polynomial in all suitable parameters.

Lemma 1. *If an adversary constructs h' with α or β which are not the outputs of H_2, the probability the signer accept h' is negligible when the signer verify h'.*

Proof. If signer verify h', the adversary should find a pair(α', β') that satisfies:

$$\alpha = H_2(\alpha', d) \ . \tag{1}$$

$$\beta = H_2(\beta', e) \ . \tag{2}$$

$$U = U' - \alpha P - \beta Q \ . \tag{3}$$

$$h' = H_1(m, U) + \beta \ . \tag{4}$$

Since H_2 is a hash function, the probability he succeeds is negligible. □

Lemma 2. *Let A be the adversary who tries to destroy the requirement that constructs blind candidates with blind factors α or β which are the outputs of the random oracle H2. If there exists 1 out of n blinded candidates $h'_i (1 \le i \le n)$, which blind factors α or β are not the outputs of H_2, then A is caught with probability 1-1/n; If there exists ≥ 2 out of n blinded candidates $h'_i (1 \le i \le n)$, which blind factors α or β are not the outputs of H_2, then A is caught with probability 1.*

Proof. There are n-1 blind candidates to satisfy the requirement of generating blind factors from H_2, so A isn't caught with probability C_{n-1}^{n-1}/C_n^{n-1} , namely $1/n$. Thus, the probability A is caught is 1-$1/n$. Similarly, we can get if there exists ≥ 2 out of n blinded candidates, which blind factors α or β are not the outputs of H_2, then A is caught with probability 1. □

Lemma 3. *Consider the above alternative, namely chooses n out of 2n to verify. Let A be the adversary who tries to destroy the requirement that constructs blind candidates with blind factors α or β which are the outputs of the random oracle H_2. Let ε be the probability of blinded candidates that blind factors α or β are not the outputs of the random oracle H_2, then the signer signs finally on a blind candidate, which blind factors α or β are the outputs of the random oracle H_2, with probability at least 1- $\varepsilon 2^{-2\varepsilon n}$.*

Proof. The number of blinded candidates which satisfy the requirement of generating blind factors from H_2 is $2(1-\varepsilon)n$, so A pass the verification with probability at most

$$C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n} = ((2n-2\varepsilon n)!n!)/((n-2\varepsilon n)!(2n)!) \ . \tag{5}$$

$$C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n} = n(n-1)(n-2\varepsilon n+1))/((2n)(2n-1)(2n-2\varepsilon n+1)) \ . \tag{6}$$

$$C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n} = 1/((1+n/n)(1+n/(n-1))(1+n/(n-2n+1))) \ . \tag{7}$$

Then, the signer randomly chooses a blind candidate from the rest n blind candidates. The probability of getting a blind candidate which blind factors α or β are not the outputs of the random oracle H_2, is 2n/n=. Thus, the signer signs finally on a blind candidates, which blind factors α or β are the outputs of the random oracle H_2, with probability at least $1-\varepsilon C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n}$. It is obvious that $2^{-2\varepsilon n} = 1/(1+1)^{2n\varepsilon} > C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n} > 1/(1+1/(1-2\varepsilon-n^{-1})^{2n\varepsilon}) \Rightarrow C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n} < 2^{-2\varepsilon n} \Rightarrow 1-\varepsilon C_{2(1-\varepsilon)n}^{n}/C_{2n}^{n} > 1-\varepsilon 2^{-2\varepsilon n}$.

So, the probability is at least $1-\varepsilon 2^{-2\varepsilon n}$. $\varepsilon 2^{-2\varepsilon n}$ is negligible when ε is sufficient large. Thus, the signer is assured that he is signing on a blind candidate that blind factors α or β are the outputs of the random oracle H_2, except a negligible probability. □

Lemma 4. *The proposed scheme is secure against one-more forgery assuming Schnorr signature scheme is secure. Concretely, suppose there is a one-more forgery adversary A that has advantage ε against our scheme within running time t. Let H_1, H_2 be random oracles. Assume that A makes at most $q_{H_1} > 0$ hash queries to H_1, $q_{H_2} > 0$ hash queries to H_2, and $q_s > 0$ signing queries to the signer. Then there is an algorithm B that performs a valid forgery against Schnorr signature scheme with probability ε in running time at most $t+t_{H_1}q_{H_1}+t_{H_2}q_{H_2}+(t_s+\tau)q_s$, where t_{H_1} is time for answering an H_1 query, t_{H_2} is time for answering an H_2 query, t_s is time for Schnorr signature scheme to generate a signature, τ and is time for answering an signing query to our scheme.*

Proof. Suppose C is the signer in Schnorr signature scheme. He keeps the secret key sk and publishes the public key pk. We show how to construct a simulator

B that uses A to forge a signature of Schnorr signature scheme for C with advantage ε. B first sends pk to A and claims that pk is his public key. Next, define the queries as follows:
- Sign query. From Lemma 1, Lemma 2 and Lemma 3, we know that A should follow the protocol. When A makes a signing query, B engages in blind signature protocol with A as follows:

B randomly chooses randomly message m' and makes a sign query to C for signature on message m'. C returns a signature (m', U, s, h). B randomly chooses $\alpha, \beta, d, e \in Z_q$ and computes $U = U' + \alpha P + \beta Q$, $s = s' + \alpha$, and sends (U', d, e) to A; A chooses α', $\beta' \in Z_q$, and queries to B for $H_2(\alpha', d)$ and $H_2(\beta', e)$. B returns (α, β) to the adversary A; A computes $U = U' - \alpha P - \beta Q$ and queries B for $H_1(m, U)$. B returns $h = H_1(m, U)$ to the adversary A; A computes $h' = h + \beta$ and sends it to B; B returns $s' = s + \alpha$; A outputs a signature $\sigma = (m, U, h, s)$ where $s = s' - \alpha$.
- Hash query. If A makes H_1 query to B, B sends the query to random oracle H_1 and returns the result from H_1 to A. If A makes H_2 query, B returns μ randomly chosen from Z_q.

It is straightforward to verify that signing query produce "valid" signatures. There is a collisions problem of the query result of H_1 query. In the sign query, B asks C to sign on the message m' which is randomly choose, C returns a valid signature (m', U, h, s). h is the random oracle H_1's answer to query $H_1(m', U)$. B simulates random oracle H_1 and cheats A that $h = H_1(m, U)$. But if A queries the same query $H_1(m', U)$ to B where the query does not come from the sign query, B returns the random oracle H_1's answer h. This may cause some "collision": a query result of H_1 query may produce a value of H_1 that is inconsistent with other query results of H_1. In this case, B just outputs fail and exits. However, since the message m' is randomly choose, the possibility of collisions is negligible. □

Lemma 5. *B simulates the signer C with an indistinguishable distribution.*

Proof. In the signing query, B simulates the signer without the secret key in the blind signature protocol. Furthermore, in the above queries, the answer to H_1 query comes from random oracle H_1 and the answer to H_2 query is randomly choose from Z_q, so they are uniformly random in their respective spaces. Therefore B simulates the signer with an indistinguishable distribution.

Since A is a successful adversary against the proposed scheme and B simulates the signer with an indistinguishable distribution, A will forge a valid signature (m_0, U_0, h_0, s_0). Since (m_0, U_0, h_0, s_0) is not equal to the outputs of the signing query, h_0 must be the right answer to query $H_1(m_0, U_0)$. So (m_0, U_0, h_0, s_0) is a valid signature for C. Thus using A, B forges a valid signature of Schnorr signature scheme for C. Since A has advantage ε within running time t, B succeeds a forgery with advantage ε in running time at most $t + t_{H_1} q_{H_1} + t_{H_2} q_{H_2} + (t_s + \tau) q_s$. □

By the above proof, we know that B makes - query to H_1 and signing queries to Schnorr signature scheme. Again, B has probability ε against Schnorr signature

scheme in running time at most $t + t_{H_1} q_{H_1} + t_{H_2} q_{H_2} + (t_s + \tau) q_s$. According to the Lemma 4 of [12], we obtain the following result:

Theorem 1. *The proposed scheme is secure against one-more forgery assume that ECDLP is hard in groups $E(F_p)$. Concretely, assume that there is an one-more forgery adversary A that has advantage ε against the proposed scheme within running time t. Let H_1, H_2 are random oracles. Assume that A makes at most q_{H_1} ¿ 0 hash queries to H_1, q_{H_2} ¿ 0 hash queries to H_2, and q_s ¿ 0 sign queries to Signer. If $\varepsilon \geq 10(q_s + 1)q_{H_1}/p$, then there is an algorithm C that solves the ECDLP problem in group $E(F_p)$ with probability $\varepsilon' \geq 1/9$ and at most time $t' \leq 23(q_{H_1} - q_s)(t + t_{H_1} q_{H_1} + t_{H_2} q_{H_2} + (t_s + \tau) q_s)/\varepsilon$, where t_{H_1} is time for answering an H_1 query, t_{H_2} is time for answering an H_2 query, t_s is time for Schnorr signature scheme to generate a signature, and τ is time for answering an signing query to the proposed scheme.* □

It is obvious that the complexity of this reduction is fully polynomial in all suitable parameters.

5 Conclusions

In this paper, we present an efficient blind signature scheme which prevents one-more forgery in the random oracle. The proof of security is fully polynomial in all suitable parameters in random oracle model. We also show that using hash functions to make the blind factors, the design of blind signature scheme can be proved secure in random oracle model assume that the original scheme is provably secure. The proposed security reduction can be an efficient technique in the proof of security for blind signature schemes.

Acknowledgment. We thank the anonymous reviewers for comprehensive comments.

References

1. Chaum, D.: Blind signatures for untraceable payments. Advances in Cryptology Crypto'82, LNCS, (1982) 199-203
2. Chaum, D.: Blind signature system. Proceedings of Crypto'83, Plenum, (1983) 153
3. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. Proceedings of Crypto'88, LNCS, (1988) 319-327
4. Chaum,D.: Security without identification. Transaction Systems to Make Big Brother Obsolete. Commun- ications of the ACM 28, (1985)
5. Guillou, L.C., Quisquater, J.J.: A practical zero-knowledge protocol fitter to security microprocessor minimizing both transmission and memory. EUROCRYPT, (1988)
6. Chaum, D.: Privacy protected payments: unconditional payer and/or payee untraceability. In Smartcard 2000, (1989) 69-93
7. Chaum, D., Boen, B., Heyst, E.: Efficient off-line electronic check. In Quisquater J, Vandewalle J, eds. Proceedings of the Eurocrypt'89, LNCS, **434** (1990) 294-301

8. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. CRYPTO, (1992)
9. Camenisch, J. L., Piveteau, J. M., Stadler, M. A.: Blind signatures based on the discrete logarithm problem. Lecture Notes in Computer Science, **950** (1995) 428-432
10. Pointcheval, D., Stern, J.: New blind signatures equivalent to factorization. In ACM SSS, ACM Press, (1997) 92-99
11. Pointcheval, D.: Strengthened security for blind signatures. Eurocrypt'98, LNCS, (1998) 391-405
12. Pointcheval,D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology, **3(13)** (2000) 361-396
13. Bellare, M., Namprempre, C., Pointcheval, D.: The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In Proceedings of Financial Cryptography 01, Springer-Verlag, (2001)
14. Abe, M.: A secure three-move blind signature scheme for polynomially many signature. Eurocrypt'01, LNCS, **2045** (2001) 136-151
15. Zhang, F., Kim, K.: ID-based blind signature and ring signature from pairings. Proc. of Asiacrpt2002, LNCS, **2501** (2002) 533-547
16. Zhang, F., Kim, K.: Efficient ID-Based blind signature and proxy signature from bilinear pairings. ACISP'03, LNCS, **2727** (2003) 312-323
17. Sherman, S.M., Lucas, C.K., Yiu, S.M.: Two improved partially blind signature schemes from bilinear pairings. Available at: http://eprint.iacr.org/2004/108.pdf
18. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. Christian Cachin and Jan Camenisch, editors: Advances in Cryptology EUROCRYPT 2004, Lecture Notes in Computer Science, **3027** (2004) 268-286
19. Camenisch, J., Koprowski, M., Warinschi, B.: Efficient blind signatures without random oracles. Blundo. Security in Communication Networks-SCN 2004, Lecture Notes in Computer Science, Berlin Heidelberg New York, **3352** (2005) 134-148
20. Liao, J., Qi, Y.H., Huang, P.W.: Pairing-based provable blind signature scheme without random oracles. CIS 2005, (2005) 161-166
21. Okamoto, T.: Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, TCC 2006, 3rd Theory of Cryptography Conference, Lecture Notes in Computer Science, New York, NY, USA, March 4-7, **3876** (2006) 80-99
22. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, Advances in Cryptology, CRYPTO 2006, Lecture Notes in Computer Science, Santa Barbara, CA, USA, August 20-24, **4117** (2006) 60-77
23. Galindo, D., Herranz, J., Kiltz, K.: On the generic construction of identity-based signatures with additional properties. ASIACRYPT, (2006) 178-193
24. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM, **2(21)** (1978) 120-126
25. Schnorr, C. P.: Efficient identification and signatures for smart cards. In Crypto '89, **435** (1990) 235-251
26. Schnorr, C. P.: Efficient signature generation by smart cards. Journal of Cryptology, **3(4)** (1991) 161-174
27. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In Proc, of the 1st CSSS, ACM Press, (1993) 62-73
28. Pointcheval, D., Stern, J.: Provably secure blind signature schemes. Asiacrypt, (1996)

29. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures. In Proceedings of Crypto'97, LNCS, **1294** (1997) 150-164
30. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Transactions on Information Theory, **24** (1978) 106-110

Construct Public Key Encryption Scheme Using Ergodic Matrices over GF(2)*

Pei Shi-Hui, Zhao Yong-Zhe**, and Zhao Hong-Wei

College of Computer Science and Technology,
Jilin University, Changchun, Jilin, 130012, PRC
{peish, yongzhe, zhaohw}@jlu.edu.cn

Abstract. This paper proposes a new public key encryption scheme. It is based on the difficulty of deducing x and y from A and $B = x \cdot A \cdot y$ in a specific monoid $(m, \cdot)$ which is noncommutative. So we select and do research work on the certain monoid which is formed by all the $n \times n$ matrices over finite field F_2 under multiplication. By the cryptographic properties of an "ergodic matrix", we propose a hard problem based on the ergodic matrices over F_2, and use it construct a public key encryption scheme.

1 Introduction

Public key cryptography is used in e-commerce systems for authentication (electronic signatures) and secure communication (encryption). The security of using current public key cryptography centres on the difficulty of solving certain classes of problems [1]. The RSA scheme relies on the difficulty of factoring large integers, while the difficulty of solving discrete logarithms provide the basis for ElGamal and Elliptic Curves [2]. Given that the security of these public key schemes relies on such a small number of problems that are currently considered hard, research on new schemes that are based on other classes of problems is worthwhile.

This paper provides a scheme of constructing a one-way(trapdoor)function, its basic thoughts are as follows:

Let $M_{n\times n}^{F_2}$ be the set of all $n \times n$ matrices over F_2, then $(M_{n\times n}^{F_2}, +, \times)$ is a 1-*ring*, here $+$ and $\times$ are addition and multiplication of the matrices over F_2, respectively. We arbitrarily select two nonsingular matrices Q_1, $Q_2 \in M_{n\times n}^{F_2}$, then:

1. $(M_{n\times n}^{F_2}, \times)$ is a monoid, its identity is $I_{n\times n}$.
2. $(\langle Q_1\rangle, \times)$ and $(\langle Q_2\rangle, \times)$ are abelian groups, their identities are $I_{n\times n}$, too.

* This work supported by the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20050183032, and the Jilin Province Education Office Science Foundation Project of China under Grant No.2004150 and No. 2005180.

** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 181–188, 2007.
© Springer-Verlag Berlin Heidelberg 2007

3. for m_1,$m_2 \in M_{n\times n}^{F_2}$, generally we have: $m_1 \times m_2 \neq m_2 \times m_1$, i.e. the operation $\times$ is noncommutative in $M_{n\times n}^{F_2}$.

Let $K = \langle Q_1\rangle \times \langle Q_2\rangle$, then we can construct a function $f : M_{n\times n}^{F_2} \times K \longrightarrow M_{n\times n}^{F_2}$, $f(m,(k_1,k_2)) = k_1 \times m \times k_2$; then f satisfies:

1. knowing $x \in M_{n\times n}^{F_2}$ and $k \in K$, it's easy to compute $y = f(x,k)$.
2. when $|\langle Q_1\rangle|$ and $|\langle Q_2\rangle|$ are big enough, knowing $x, y \in M_{n\times n}^{F_2}$, it's may be hard to deduce $k \in K$ such that $y = f(x,k)$.
3. form $k = (k_1,k_2) \in K$, it's easy to compute $k^{-1} = (k_1^{-1}, k_2^{-1}) \in K$, and for any $x \in M_{n\times n}^{F_2}$, we always have: $f(f(x,k),k^{-1}) = x$.

If 2 is true then by 1 and 2 we know that f has one-way property; by 3, we can take k as the "trapdoor" of the one-way function f, hence we get a one-way trapdoor function.

For $\forall m \in M_{n\times n}^{F_2}$, we know that $Q_1 \times m$ does corresponding linear transformation to every column of m, while $m \times Q_2$ does corresponding linear transformation to every row of m; So, $Q_1 \times m \times Q_2$ may "disarrange" every element of m. This process can be repeated many times, i.e. $Q_1^x m Q_2^y (1 \leq x \leq |\langle Q_1\rangle|, 1 \leq y \leq |\langle Q_2\rangle|)$, to get a complex transformation of m. To increase the quality of encryption(transformation), the selection of Q_1,Q_2 should make the generating set $\langle Q_1\rangle$ and $\langle Q_2\rangle$ as big as possible. And the result, of which Q_1 multiplying a column vector on the left and Q_2 multiplying a row vector on the right, should not be convergent. For this purpose, we put forward the concept of ergodic matrix.

2 Ergodic Matrices over Finite Field F_2

Let F_2^n be the set of all n-dimensional column vectors over finite field F_2.

Definition 1. *Let $Q \in M_{n\times n}^{F_2}$, if for any nonzero n-dimensional column vector $v \in F_2^n \backslash \{0\}, Qv, Q^2v, \ldots, Q^{2^n-1}v$ just exhaust $F_2^n \backslash \{0\}$, then Q is called an "ergodic matrix" over F_2.* $(0 = [0\ 0 \cdots 0]^T)$

For example, select the following matrix $Q \in M_{2\times 2}^{F_2}$:

$$Q = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$then\ \ Q^2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} Q^3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We verify weather Q is an ergodic matrix.
Let $v_1 = [0,1]^T$, $v_2 = [1,0]^T$, $v_3 = [1,1]^T$, then $F_2^2 \backslash \{0\} = \{v_1, v_2, v_3\}$.

To multiply v_1 by Q^1, Q^2, Q^3 respectively, we have:

$$Q^1 v_1 = \begin{bmatrix} 1\,1 \\ 1\,0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = v_2$$

$$Q^2 v_1 = \begin{bmatrix} 0\,1 \\ 1\,1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = v_3$$

$$Q^3 v_1 = \begin{bmatrix} 1\,0 \\ 0\,1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = v_1$$

Their result just exhaust $F_2^2 \backslash \{0\}$. For v_2 and v_3 the conclusion is the same. By Definition 1 Q is an ergodic matrix.

Theorem 1. *$Q \in M_{n\times n}^{F_2}$ is an ergodic matrix iff Q's period, under the multiplication, is $(2^n - 1)$.*

Proof. If $Q \in M_{n\times n}^{F_2}$ is an ergodic matrix, then for $\forall v \in F_2^n \backslash \{0\}$, it must be $Q^{2^n-1} v = v$. Let v respectively be $[1\ 0 \cdots 0]^T, [0\ 1\ 0 \cdots 0]^T, \ldots, [0 \cdots 0\ 1]^T$, then $Q^{2^n-1} = I_{n\times n}$, *i.e.* Q is nonsingular and Q's period divides $(2^n - 1)$ exactly; by Definition 1, Q's period must be $(2^n - 1)$.

If the period of $Q \in M_{n\times n}^{F_2}$ under multiplication is $(2^n - 1)$, then $\langle Q \rangle = \{Q, Q^2, \ldots, Q^{2^n-1} = I_{n\times n}\}$. By Cayley-Hamilton's theorem [3], we have:

$$F_2[Q] = \{p(Q) | p(t) \in F_2[t]\} = \{p(Q) | p(t) \in F_2[t] \wedge deg\ p \leq n - 1\}$$

i.e. $|F_2[Q]| \leq 2^n$; Obviously $\langle Q \rangle \subseteq F_2[Q] \backslash \{0_{n\times n}\}$, so that:

$$F_2[Q] = \{0_{n\times n}, Q, Q^2, \ldots, Q^{2^n-1} = I_{n\times n}\}$$

Arbitrarily selecting $v \in F_2^n \backslash \{0\}$ and $Q^s, Q^t \in \langle Q \rangle$, if $Q^s v = Q^t v$, then$(Q^s - Q^t)v = 0$. Because $(Q^s - Q^t) \in F_2[Q]$ and $v \neq 0$, we have $(Q^s - Q^t) = 0_{n\times n}$, i.e. $Q^s = Q^t$. So, $Qv, Q^2 v, \ldots, Q^{2^n-1} v$ just exhaust $F_2^n \backslash \{0\}$, Q is an ergodic matrix. □

By Cayley-Hamilton's theorem, and finite field theory [4], it's easy to get the following lemmas:

Lemma 1. *If $m \in M_{n\times n}^{F_2}$ is nonsingular, then m's period is equal to or less than $(2^n - 1)$.*

Lemma 2. *If $Q \in M_{n\times n}^{F_2}$ is an ergodic matrix, then $(F_2[Q], +, \times)$ is a finite field with 2^n elements.*

Lemma 3. *If $Q \in M_{n\times n}^{F_2}$ is an ergodic matrix, then Q^T must also be an ergodic matrix.*

Lemma 4. *If $Q \in M_{n\times n}^{F_2}$ is an ergodic matrix, then for $\forall v \in F_2^n \backslash \{0\}$, $v^T Q, \ldots, v^T Q^{2^n-1}$ just exhaust $\{v^T | v \in F_2^n\} \backslash \{0^T\}$.*

Lemma 5. *If $Q \in M_{n\times n}^{F_2}$ is an ergodic matrix, then for $\forall a \in F_2$, $aQ \in F_2[Q]$.*

Lemma 6. *If $Q \in M_{n\times n}^{F_2}$ is an ergodic matrix, then there are just $\varphi(2^n-1)$ ergodic matrices in $\langle Q\rangle$ and we call them being "equivalent" each other. here $\varphi(x)$ is Euler's totient function.*

From above, we know that the ergodic matrices over $M_{n\times n}^{F_2}$ has a maximal generating set, and the result of multiplying a nonzero column vector on the left or multiplying a nonzero row vector on the right by the ergodic matrix is thoroughly divergent; thus it can be used to construct one-way(trapdoor) function.

3 New Public Key Encryption System

3.1 Hard Problem

Problem 1. Let $Q_1,\ Q_2 \in M_{n\times n}^{F_2}$ be ergodic matrices, knowing that $A, B \in M_{n\times n}^{F_2}$, find $Q_1^x \in \langle Q_1\rangle$, $Q_2^y \in \langle Q_2\rangle$ such that $B = Q_1^x A Q_2^y$.

Suppose Eve knows A, B and their relation $B = Q_1^x A Q_2^y$, for deducing Q_1^x and Q_2^y, he may take attacks mainly by [5,6,7]:

1. Brute force attack
   For every $Q_1^s \in \langle Q_1\rangle$, and $Q_2^t \in \langle Q_2\rangle$, Eve computes $B' = Q_1^s A Q_2^t$ until $B' = B$, hence he gets $Q_1^x = Q_1^s$, $Q_2^y = Q_2^t$.
2. Simultaneous equations attack
   Eve elaborately selects $a_1, a_2, \ldots, a_m \in \langle Q_1\rangle$ and $b_1, b_2, \ldots, b_m \in \langle Q_2\rangle$, constructing the simultaneous equations as follows:

$$\begin{cases} B_1 = Q_1^x A_1 Q_2^y \\ B_2 = Q_1^x A_2 Q_2^y \ (Here\ A_k = a_k A b_k\ B_k = a_k B b_k\ are\ know) \\ \quad\vdots \qquad\qquad \vdots \\ B_m = Q_1^x A_m Q_2^y \end{cases}$$

   Thus Eve may possibly deduce Q_1^x and Q_2^y.

But all of these attacks are not polynomial time algorithm. We assume through this paper the Problem 1 are intractable, which means there is no polynomial time algorithm to solve it with non-negligible probability.

3.2 Public Key Encryption Scheme

Inspired by [8,9,10], We propose a new public key encryption scheme as follow:

- Key Generation.
  The key generation algorithm select two ergodc matrices $Q_1,\ Q_2 \in M_{n\times n}^{F_2}$ and a matrix $m \in M_{n\times n}^{F_2}$. It then chooses $s, t \in [0, 2^{n-1}]$, and sets $sk = (s, t)$, $pk = (Q_1, Q_2, m, Q_1^s m Q_2^t)$.

- Encryption.
  On input message matrix X, public key $pk = (Q_1, Q_2, m, Q_1^s m Q_2^t)$, choose $k, l \in [0, 2^{n-1}]$, computer $Z = X + Q_1^k Q_1^s m Q_2^t Q_2^l$, and output the ciphertext $Y = (Z, Q_1^k m Q_2^l)$.
- Decryption.
  On input $sk = (s, t)$, ciphertext $Y = (Z, C)$, output the plaintext $X = Z - Q_1^s C Q_2^t = Z - Q_1^s Q_1^k m Q_2^l Q_2^t = Z - Q_1^k Q_1^s m Q_2^t Q_2^l$.

The security for the public key encryption scheme based on ergodic matrices is defined through the following attack game:

1. The adversary queries a key generation oracle, the key generation oracle computes a key pair (pk, sk) and responds with pk.
2. The challenger gives the adversary a challenge matrix $c \in M_{n\times n}^{F_2}$.
3. The adversary makes a sequence of queries to a decryption oracle. Each query is an arbitrary ciphertext matrix (not include c); the oracle responds with corresponding plaintext.
4. At the end of the game, the adversary output a matrix a.

The advantage of an adversary is: $Adv = Pr[a = Decryption(c, sk)]$.

Definition 2. *A public key encryption scheme is said to be secure if no probabilistic polynomial time adversary has a non-negligible advantage in the above game.*

Theorem 2. *The security of the public key encryption scheme based on ergodic matrices is equivalent to the Problem 1.*

3.3 Example

(1) Key generation: Select two ergodic matrices $Q_1, Q_2 \in M_{23\times 23}^{F_2}$:

$$Q_1 = \begin{bmatrix}
0&0&1&0&0&1&1&1&0&0&0&1&0&0&0&1&1&1&0&1&0&1&1\\
1&1&1&0&0&1&1&1&1&0&1&1&1&1&0&0&1&0&0&1&1&1&0\\
1&0&0&1&0&1&0&0&1&1&0&1&0&1&0&1&1&1&0&0&0&0&1\\
0&1&0&1&1&1&1&0&0&0&0&1&0&0&0&1&0&1&0&1&0&1&1\\
1&0&0&1&1&0&0&1&0&0&0&0&0&1&1&0&1&1&0&1&0&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&1&1\\
1&1&0&1&1&1&0&1&1&0&0&1&0&0&1&0&1&0&1&0&0&0&0&1&0\\
1&1&1&0&0&1&1&1&1&0&1&1&1&1&0&0&0&0&1&1&1&1&1\\
1&1&1&0&0&0&0&1&1&1&1&0&1&1&0&1&1&0&1&0&0&0&0\\
1&1&1&0&1&1&1&0&0&0&0&0&0&0&0&1&0&0&1&1&1&1&0\\
1&1&0&1&1&1&1&1&0&1&1&1&1&0&0&0&1&0&0&0&0&0&1&0\\
0&1&0&0&1&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1\\
1&0&0&1&0&0&1&0&1&1&1&1&1&1&0&1&0&1&1&0&1&1&1&1\\
1&1&0&0&1&0&1&1&0&1&1&0&1&0&0&1&1&1&0&0&1&1&1\\
1&0&0&0&1&0&1&0&1&0&0&0&0&1&0&1&0&0&1&1&1&1&1\\
1&0&0&0&0&1&0&0&0&0&0&0&1&0&1&1&0&1&1&0&0&0&1\\
1&1&0&0&1&0&0&1&1&1&0&0&0&1&0&1&1&1&1&0&1&1&0\\
0&0&0&0&1&1&0&1&0&0&0&1&0&1&1&0&0&0&1&1&0&1&0\\
1&1&0&1&0&1&1&1&1&1&1&0&0&0&1&0&0&0&1&0&0&0&0\\
1&0&1&1&0&0&1&1&1&0&1&1&1&1&0&1&0&1&0&1&0&0&1\\
1&0&1&0&0&1&1&1&1&1&0&1&1&1&0&0&0&1&0&1&0&1&1\\
1&0&0&0&1&0&0&1&0&0&1&1&0&1&1&0&1&1&0&1&0&0&0\\
1&1&0&1&0&0&0&1&0&1&0&0&1&1&1&1&1&1&0&0&0&0&1&1
\end{bmatrix}$$

$$Q_2 = \begin{bmatrix}
1&1&0&1&0&0&0&1&0&0&1&0&1&0&0&1&0&0&1&1&0&0&1\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&1&0&1&0&0&1&1&0\\
1&1&1&1&0&0&0&1&1&0&0&1&0&1&1&1&0&1&1&0&0&0&0\\
1&0&0&0&0&0&1&1&1&0&0&0&1&0&1&0&0&1&0&1&1&1&1\\
1&0&0&1&1&1&1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1&0\\
0&1&1&1&1&0&0&1&0&1&0&0&0&1&1&0&1&0&1&1&0&0&0\\
1&1&1&0&1&0&0&1&0&0&0&0&0&1&1&1&1&1&0&1&0&0&0\\
1&0&1&1&1&0&1&0&0&0&0&1&1&1&0&1&1&1&0&0&0&0&0\\
1&1&0&1&1&1&0&1&0&0&0&1&1&1&1&1&1&0&0&0&1&1&0\\
1&1&0&1&0&0&1&1&1&0&1&1&1&1&0&1&0&1&1&1&1&1&1\\
1&0&1&1&0&0&0&0&1&1&1&0&1&0&1&1&0&0&1&1&0&1&0\\
0&0&0&1&1&0&1&1&0&0&0&0&1&0&0&1&1&0&0&0&1&1&1\\
0&1&1&1&1&1&1&1&1&1&1&0&1&0&1&1&0&0&1&0&0&1&0&1\\
1&0&1&1&0&0&1&1&1&0&0&1&0&0&1&1&0&1&0&1&1&0&1\\
1&1&0&0&0&0&0&0&1&0&0&0&1&0&1&1&1&0&1&0&0&1&0\\
1&0&0&1&1&0&0&0&1&1&1&0&1&0&1&1&1&0&0&1&1&1&1\\
1&1&1&0&1&1&1&1&0&1&1&1&0&1&0&0&0&1&1&0&1&1&0\\
0&0&0&1&1&1&0&0&1&0&1&1&0&0&0&0&1&0&1&1&1&1&0\\
0&0&1&0&0&1&0&1&0&1&1&1&0&0&0&1&0&1&1&1&0&1&0\\
0&1&1&0&0&0&1&1&1&0&0&1&1&0&1&1&0&0&0&1&1&0&1\\
1&0&0&1&0&1&1&1&1&1&1&0&0&1&0&0&1&0&0&1&1&1&1\\
1&0&1&1&0&1&1&1&0&0&0&0&0&1&0&1&0&0&0&0&1&0&0\\
0&0&1&0&1&1&0&1&0&1&1&0&0&0&1&0&0&1&0&0&0&0&1
\end{bmatrix}$$

the ergodic matrices Q_1 and Q_2 can be generated using following algorithm:

1. select a random matrix $m \in M_{n\times n}^{F_2}$;
2. if $Rank(m) < n$, goto 1;
3. if $m^{2^n-1} \neq I_{n\times n}$, goto 1;
4. m is a ergodic matrix.

Moveover, we need select a matrices $m \in M_{23\times 23}^{F_2}$ randomly:

$$m = \begin{bmatrix}
0&0&1&0&1&1&0&0&0&0&0&0&1&1&0&0&0&1&0&1&0&1&0\\
1&0&0&1&0&1&1&1&1&0&0&0&1&1&0&0&1&1&0&1&0&1&1\\
1&1&1&0&0&0&1&1&1&0&1&1&0&1&0&1&1&1&0&0&0&1&1\\
0&0&0&1&1&1&1&1&0&0&1&0&1&0&0&0&1&0&0&0&1&1&1\\
1&1&1&0&1&0&0&1&0&0&1&1&1&1&0&1&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&0&1&1&0&0&0&0&0&1&0&0&1&0&1&1&1&1\\
0&1&1&1&1&1&1&1&1&0&1&0&1&0&0&0&1&0&1&0&1&1&1\\
0&1&1&1&1&1&0&0&1&0&0&1&1&0&0&1&1&0&1&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&1&0&1&1&0&0&1&1&0&1&0&0&0&1&0&1&1&0&0\\
0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&1&0&1&0&0&0&0&1\\
0&0&1&1&0&1&1&0&0&0&0&0&1&1&1&0&1&1&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&1&0&0&1&1&0&1&1&0&0&1&0&1&0&0&1&0&1&0\\
1&1&0&1&0&1&1&1&0&0&0&0&0&1&0&0&1&0&0&0&0&1&1\\
0&1&1&0&0&1&0&1&0&0&1&0&1&1&1&0&1&0&0&0&0&0&0\\
0&1&0&1&1&0&0&0&1&0&1&1&1&1&0&0&1&0&0&1&0&0&1\\
1&1&0&1&0&0&0&1&1&0&1&0&1&0&1&1&1&1&0&0&0&0&1\\
0&1&0&1&0&1&0&0&0&0&1&1&0&1&1&1&0&0&0&1&0&1&1\\
1&0&0&0&0&1&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&1\\
1&0&1&0&1&1&0&0&1&0&1&0&0&1&1&1&1&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

then select private key: $s = 1367235$ $t = 2480563$, and compute:

$$Q_1^{1367235} m Q_2^{2480563} = \begin{bmatrix} 1&0&0&1&1&0&1&1&0&1&1&1&1&1&1&1&1&1&0&1&0&1 \\ 0&0&0&1&0&1&1&0&1&0&0&0&0&0&0&1&0&0&0&1&1&1&1 \\ 1&1&1&1&1&1&1&1&1&0&1&0&0&0&1&1&1&1&0&0&1&1&0&0 \\ 1&1&0&1&1&1&1&1&0&0&1&0&1&0&1&1&1&0&1&1&1&1&1&0 \\ 1&0&0&1&0&1&1&0&0&0&1&0&0&1&0&1&0&1&1&1&0&0&1&1 \\ 1&1&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0 \\ 0&0&1&1&0&0&1&1&1&0&0&0&0&1&1&1&0&1&1&0&0&0&1 \\ 1&1&0&0&0&0&1&1&0&1&0&0&0&0&1&0&1&1&0&1&0&1&1 \\ 1&1&0&1&1&0&1&1&0&1&1&1&0&0&1&1&0&0&0&0&1&0&1 \\ 0&1&1&0&1&0&1&0&0&1&0&1&0&0&0&1&1&1&0&0&1&1&0 \\ 1&0&1&1&0&0&1&0&0&0&1&0&0&0&1&1&1&1&0&0&0&0&1&1 \\ 0&1&1&0&1&1&1&1&1&0&1&1&1&0&1&1&1&1&0&1&0&1&0&0 \\ 1&1&0&0&1&0&0&1&0&0&0&1&1&0&0&1&0&0&1&0&0&1&1 \\ 1&0&0&0&0&1&0&1&0&1&0&0&0&1&0&1&1&0&1&0&0&1&1 \\ 0&0&1&1&1&0&1&1&0&0&1&1&1&1&1&0&0&0&0&0&0&1&1 \\ 0&0&1&0&0&0&1&0&0&0&1&1&1&1&1&1&0&1&1&0&1&1&1 \\ 1&1&1&1&1&1&0&1&1&1&0&0&0&1&1&0&1&0&0&1&1&1&0 \\ 1&1&0&1&0&0&1&1&1&0&0&1&1&0&0&1&1&0&0&1&0&1&1 \\ 0&0&0&1&1&0&0&0&1&0&0&1&0&1&0&1&0&1&0&1&0&1&0 \\ 0&1&0&0&0&1&1&0&1&0&0&1&1&1&1&0&1&1&1&1&1&0&1 \\ 1&1&0&0&0&1&0&0&0&0&1&1&0&1&0&0&1&0&0&0&1&1&1 \\ 1&0&0&0&0&1&0&1&0&0&1&0&0&1&0&0&0&1&1&0&1&0&0 \\ 0&1&1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&0 \end{bmatrix}$$

so the public key is $pk = (Q_1, Q_2, m, Q_1^s m Q_2^t)$.

(2)In the process of Encryption, select two random integers: $k = 4321506$, $l = 3493641$.

because $Q_1^k Q_1^s m Q_2^t Q_2^l = Q_1^s Q_1^k m Q_2^l Q_2^t$, i.e.

$Q_1^{4321506} Q_1^{1367235} m Q_2^{2480563} Q_2^{3483641} = Q_1^{1367235} Q_1^{4321506} m Q_2^{3483641} Q_2^{2480563}$,
the result is:

$$\begin{bmatrix} 1&1&1&1&0&0&0&1&0&0&1&1&1&1&1&0&1&0&1&0&0&1&1 \\ 0&0&1&0&0&1&0&0&0&1&0&0&1&1&0&0&1&0&1&0&0&0&0 \\ 1&0&1&0&1&1&1&1&0&1&1&0&0&1&0&0&0&0&0&1&1&1&0 \\ 0&1&0&1&0&0&1&1&1&1&0&0&0&1&0&0&1&1&0&1&0&1&0 \\ 0&1&0&0&1&0&0&0&1&1&1&1&1&1&0&0&1&0&1&0&0&0&0 \\ 1&0&1&0&0&0&0&1&0&0&1&1&1&1&1&1&1&0&0&0&1&1&0 \\ 0&1&1&1&1&0&0&0&0&1&1&1&0&1&1&1&0&1&0&0&0&1&0 \\ 0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&0&1&0&0&1&1&0&1 \\ 1&0&0&1&1&0&1&1&1&1&0&0&1&0&1&0&1&1&1&0&1&1&0 \\ 0&1&0&1&0&1&0&1&1&0&0&0&0&0&1&1&0&1&1&1&0&1&0 \\ 1&1&1&0&0&0&1&1&0&0&0&0&1&0&1&0&0&1&0&0&0&1&0 \\ 1&1&1&1&0&0&0&0&1&0&0&0&1&1&0&0&1&1&0&1&0&0&0 \\ 1&0&0&1&1&0&1&0&0&1&1&0&0&0&0&0&1&1&0&0&1&1&0 \\ 1&1&0&0&1&1&0&0&1&0&0&0&0&0&1&0&0&1&0&0&1&0&0 \\ 0&1&0&0&0&0&1&0&1&0&0&1&0&1&1&1&0&0&0&0&0&0&0 \\ 1&0&0&0&1&1&0&0&1&0&0&1&1&0&0&0&0&1&1&1&0&0&1 \\ 1&1&1&0&1&0&0&1&0&1&1&1&0&0&0&1&0&0&0&0&1&1&0 \\ 1&1&0&0&1&0&0&1&1&1&0&1&1&1&0&1&0&0&0&0&0&1&1 \\ 0&1&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&1&1&1&1&1 \\ 0&1&0&0&1&1&0&1&0&0&0&1&0&1&1&1&1&1&0&1&0&1&0 \\ 1&0&1&0&0&1&0&0&0&0&1&0&1&1&0&1&0&1&1&0&0&1&1 \\ 1&0&0&0&1&0&1&0&0&1&1&0&1&0&0&1&1&0&1&1&1&1&0 \\ 1&1&0&1&1&0&0&1&0&1&0&0&0&0&1&1&0&1&1&0&0&1&0 \end{bmatrix}$$

It is easy to verify the process of encryption and decryption.

4 Conclusions

The ergodic matrices over $M_{n\times n}^{F_2}$ has a maximal generating set, and the result of multiplying a nonzero column vector on the left or multiplying a nonzero row vector on the right by the ergodic matrix is thoroughly divergent; thus it can be used to construct one-way(trapdoor) function. In this paper, we propose a new hard problem based on the ergodic matrices over F_2, by which, we implement a public key encryption scheme. Different from the previous approaches, we adopt matrix to represent plaintext, which can encrypt more information once a time.

We plan to give the theoretical proof on the hard problem based on the ergodic matrices over F_2. Additional research is also required to compare the security and performance with other public key encryption schemes such as RSA and Elliptic Curves.

References

1. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone: Handbook of Applied Cryptography, New York: CRC Press,1997.
2. Bruce Schneier: Applied Cryptography: protocols, algorithms, and source code in C, USA, John Wiley & Sons, Inc. 1996.
3. F.R. Gantmacher: The Theory of Matrices, Vol.2, New York:Chelsea, 1974
4. R. Lidl, H. Niederreiter: Introduction to Finite Fields and Their Applications, Cambridge: Univ. Press, 1994.
5. Y. Zhao, L. Wang L, W. Zhang: Information-Exchange Using the Ergodic Matrices in GF(2), Proc. 2th International Conference on Applied Cryptography and Network Security (ACNS 2004). ICISA PRESS, 2004: 388-397.
6. Y. Zhao, S. Huang, Z. Jiang: Ergodic matrices over GF(2k) and their properties, Mini-Micro Systems, 2005, 26(12): 35-39.
7. Y. Sun, Y. Zhao, Y. Yang, R. Li: Scheme to construct one-way(trapdoor)functions based on ergodic matrice, Journal of Jilin University, 2006, 24(5):554-560.
8. T. ElGamal: A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inform. Theory, 31(4):469-472, 1985.
9. M. Gerard, M. Chris, R. Joachim: A public key cryptosystem based on actions by semigroups, IEEE International Symposium on Information Theory-Proceedings, 2002, p.266-289
10. R. Arash, H. Anwar: A new construction of Massey-Omura parallel multiplier over GF(2^m), IEEE Transactions on Computers, v51(5), 2002:511-520.

New Left-to-Right Radix-r Signed-Digit Recoding Algorithm for Pairing-Based Cryptosystems

Fanyu Kong[1,2], Jia Yu[3], Zhun Cai[1,2], and Daxing Li[1,2]

[1] Institute of Network Security, Shandong University, 27 Shanda Nanlu Road, Jinan 250100, China
fanyukong@sdu.edu.cn
[2] Key Laboratory of Cryptographic Technology and Information Security, Ministry of Education, Jinan 250100, China
[3] College of Information Engineering, Qingdao University, Qingdao 266071, China

Abstract. In pairing-based cryptosystems, radix-r signed-digit representations are used to speed up point multiplication over supersingular elliptic curves or hyper-elliptic curves in characteristic r. We propose a left-to-right radix-r signed-digit recoding algorithm, which can obtain a new signed-digit representation from left to right. It is proved that its average non-zero density is asymptotically $\frac{1}{2} - \frac{2r+3}{2r(r+1)^2}$, which is reduced by 20%-50% compared with the previous left-to-right radix-r signed-digit representations. The proposed algorithm can be applied to efficient implementations of pairing-based cryptosystems over supersingular elliptic curves or hyper-elliptic curves.

Keywords: Elliptic curve cryptosystems, pairing-based cryptosystems, point multiplication, signed-digit number representations.

1 Introduction

The bilinear pairings over supersingular elliptic curves or hyper-elliptic curves have been applied to construct an attractive family of pairing based cryptosystems, such as tripartite Diffie-Hellmann scheme [1], ID-based encryption [2], short digital signature [3], etc. However, the computation of the bilinear pairing such as Weil and Tate pairing is computationally expensive. The fundamental algorithm for computing bilinear pairings is developed by V. S. Miller [4]. Several efficient algorithms for pairing based cryptosystems over supersingular elliptic curves in characteristic three have been developed in [5, 6, 7]. At ASIACRYPT 2003, Duursam and Lee proposed an efficient implementation of Tate pairing over hyper-elliptic curves in general characteristic r [8].

The radix-r signed-digit number representations play an important role in the efficient implementation of pairing-based cryptosystems over supersingular or hyper-elliptic curves in characteristic r. Signed-digit binary representations of integers were firstly introduced to speed up parallel arithmetics such as multipliers, adders and so on, by A. D. Booth [9] and G. W. Reitwiesner [10]. Furthermore, the generalized radix-r ($r \geq 2$) non-adjacent form (GNAF) [11, 12]

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 189–198, 2007.
© Springer-Verlag Berlin Heidelberg 2007

was proposed and its average density of non-zero digits is asymptotically $\frac{r-1}{r+1}$, which is minimal among all radix-r signed-digit representations with the digit set $\{0, \pm 1, \ldots, \pm(r-1)\}$. A very interesting problem introduced by J. A. Solinas [13] and V. Müller [14] is how to develop left-to-right signed-digit recoding algorithms, which are desirable for point multiplication over elliptic curves with a smaller space requirement. Some excellent work [15,16,17,18] have proposed several left-to-right signed-digit binary recoding algorithms. For radix-r signed-digit representations, M. Joye and S.-M. Yen [19] proposed a left-to-right generalized star form (GSF) with the same weight as the generalized non-adjacent form (GNAF). Recently, J. A. Muir [20] gave another left-to-right radix-r signed-digit recoding with the same weight as the GNAF and the GSF. In [21], F. Kong analyzed the expected on-line efficiency of the left-to-right GNAF recoding algorithm. At ISC 2004, T. Takagi, S.-M. Yen and B.-C. Wu [22] proposed the radix-r non-adjacent form (rNAF), whose average non-zero density is asymptotically $\frac{r-1}{2r-1}$ in contrast with the non-zero density $\frac{r-1}{r+1}$ of the GNAF.

In this paper, we propose a new left-to-right radix-r signed-digit recoding algorithm and by the performance analysis, its average non-zero density is asymptotically $\frac{1}{2} \times \left(1 - \frac{1}{r^2} - \frac{1}{(r+1)^2} - \frac{r-1}{r^2(r+1)^2}\right)$, i.e. $\frac{1}{2} - \frac{2r+3}{2r(r+1)^2}$, which is close to that of the rNAF. By the comparison, its average non-zero density is reduced by 20%-50% compared with the previous left-to-right radix-r signed-digit representations such as the GNAF and the GSF. This algorithm can be applied to speed up point multiplications over supersingular elliptic curves or hyper-elliptic curves in characteristic r in pairing-based cryptosystems.

The rest of this paper is organized as follows. In Section 2, we summarize radix-r signed-digit number representations and their applications in elliptic curve cryptosystems (ECC). In Section 3, we propose a new left-to-right radix-r signed-digit recoding algorithm. In Section 4, we analyze the proposed algorithm and compare them with others, such as the GSF and so on. Finally, in Section 5 we conclude the paper.

2 Preliminaries

2.1 Notations

If an integer $n = \sum_{i=0}^{k-1} n_i \times r^i$, where $n_i \in \{0, 1, \ldots, r-1\}$ and $n_{k-1} \neq 0$, we call $(n_{k-1}, \ldots, n_1, n_0)$ the radix-r standard representation of n. Moreover, the number of non-zero digits is called the Hamming weight of the representation $(n_{k-1}, \ldots, n_1, n_0)$. If $n = \sum_{i=0}^{k-1} n_i \times r^i$ with $n_i \in \{0, \pm 1, \ldots, \pm(r-1)\}$ and $n_{k-1} \neq 0$, we call $(n_{k-1}, \ldots, n_1, n_0)$ a radix-r signed-digit representation of n.

2.2 Radix-r Signed-Digit Number Representations

Now we give several definitions and properties of various radix-r signed-digit representations such as the GNAF and the rNAF, which have been described in detail in [11, 12, 19, 20, 21, 22, 23, 24, 25].

Definition 1. *[11,12] A signed radix-r representation* $(n_{k-1}, \ldots, n_1, n_0)$*, where* $n_i \in \{0, \pm 1, \ldots, \pm(r-1)\}$ *and* $0 \le i \le k-1$*, is called a generalized non-adjacent form (GNAF) of n if and only if*

(1) $|n_i + n_{i+1}| < r$*, where* $0 \le i \le k-2$*, and*

(2) $|n_i| < |n_{i+1}|$ *if* $n_i \cdot n_{i+1} < 0$*, where* $0 \le i \le k-2$*.*

Theorem 1. *[11, 12] The GNAF has the following properties:*

(1) Every integer n has a unique GNAF.

(2) The GNAF is an optimal signed-digit representation with the non-zero density $\frac{r-1}{r+1}$*.*

The properties and algorithms of the GSF and J. A. Muir's representation can be seen in the literature [19,20]. At ISC 2004, T. Takagi et al. [22] proposed the radix-r non-adjacent form (rNAF), whose average non-zero density is asymptotically $\frac{r-1}{2r-1}$. The definition and properties of the rNAF are given as follows.

Definition 2. *[22] Suppose that an integer* $n = \sum_{i=0}^{k-1} n_i \times r^i$*, where for* $0 \le i \le k-1$*,* $n_i \in \{0, \pm 1, \ldots, \pm\lfloor \frac{r^2-1}{2} \rfloor\} \setminus \{\pm 1r, \pm 2r, \ldots, \pm\lfloor \frac{r-1}{2} \rfloor r\}$ *and* $n_{k-1} > 0$*,* $(n_{k-1}, \ldots, n_1, n_0)$ *is called radix-r non-adjacent form (rNAF) if it satisfies that* $n_i \cdot n_{i-1} = 0$ *for all* $i = 1, 2, \ldots, k-1$*.*

Theorem 2. *[22] The rNAF has the following properties:*

(1) Every integer n has a unique rNAF representation.

(2) The rNAF is an optimal radix-r signed-digit representation with the minimal non-zero density $\frac{r-1}{2r-1}$*.*

The rNAF representation of an integer n can be generated from the least significant digit to the most significant digit, i.e. from right to left. The generation algorithm for the rNAF representation [22] is described as follows. The notation 'mods' denotes the signed modulo, namely x mods r^2 is equal to $(x \bmod r^2) - r^2$ if $(x \bmod r^2) \ge r^2/2$, otherwise $(x \bmod r^2)$.

Algorithm 1. The generation algorithm for the rNAF representation [22]

Input: The integer $n = (n_{k-1}, \ldots, n_1, n_0) = \sum_{i=0}^{k-1} n_i \times r^i$, where
$n_i \in \{0, 1, \ldots, r-1\}$ and $n_{k-1} \neq 0$.
Output: The representation $n = (n'_k, \ldots, n'_1, n'_0) = \sum_{i=0}^{k} n'_i \times r^i$, where
$n_i \in \{0, \pm 1, \ldots, \pm\lfloor \frac{r^2-1}{2} \rfloor\} \setminus \{\pm 1r, \pm 2r, \ldots, \pm\lfloor \frac{r-1}{2} \rfloor r\}$.
1. $i \leftarrow 0; x \leftarrow n;$
2. while $x > 0$ do
   2.1 {if $(x \bmod r)$=0 then $n'_i \leftarrow 0$;
   2.2 else $\{n'_i \leftarrow x$ mods r^2; $x \leftarrow x - n'_i;\}$
   2.3 $x \leftarrow x/r; i \leftarrow i + 1;\}$
3. end.

2.3 Left-to-Right Recoding Algorithm and Its Application in ECC

The computation of point multiplication $n \cdot P$ (where n is a scalar and P is a point) over elliptic curves is analogous operation as an exponentiation g^k on multiplicative groups. Various signed-digit number representations and sliding window algorithms, which are surveyed in detail in [24,25], can be used to speed up this operation. The point multiplication $n \cdot P$ can be carried out from left to right or from right to left, which are shown as follows.

Algorithm 2. The point multiplication algorithms for computing $n \cdot P$

Input:The integer $n = \sum_{i=0}^{k-1} n_i \times r^i$ and a point P.
Output: $Q = n \cdot P$.

(a) left-to-right algorithm

1. $Q \leftarrow n_{k-1} \cdot P$;
2. for $i = k-2$ downto 0 do
   { $Q \leftarrow r \cdot Q$;
   $Q \leftarrow Q + n_i \cdot P$; }
3. end.

(b) right-to-left algorithm

1. $Q \leftarrow O$; $S \leftarrow P$;
2. for $i = 0$ to $k-1$ do
   { $Q \leftarrow Q + n_i \cdot S$;
   $S \leftarrow r \cdot S$; }
3. end.

Note that the left-to-right point multiplication algorithms such as sliding window methods, the w-NAF and radix-r representations, are superior to the right-to-left counterparts since that $n_i \cdot P$ can be precomputed and stored (See [13,14,15,16,17,18,19,20,21]). However, this can not be implemented in right-to-left algorithms. Indeed, various improvement techniques such as sliding window methods, the w-NAF and radix-r representations, can't work efficiently in right-to-left point multiplication algorithms. Therefore, we should develop left-to-right signed-digit recoding algorithms, otherwise we have to require more memory space and time to firstly generate the signed-digit representations for the sake of the coming point multiplications.

3 New Left-to-Right Radix-r Signed-Digit Recoding Algorithm

3.1 The Basic Idea of Our Algorithm

Now we propose a heuristic left-to-right radix-r signed-digit recoding algorithm, which obtains a new radix-r signed-digit representation. Note that the rNAF, which was proposed by T. Takagi et al. [22], is probably the most efficient radix-r signed-digit representation in spite of its large digit set. However, the rNAF can only be computed from right to left.

The basic idea of our algorithm is to try to develop a new radix-r signed-digit representation with the non-zero density close to that of the rNAF, which can be generated from left to right. We can process two adjacent digits (n_i, n_{i-1}) in one recurrence from left to right and obtain the two corresponding signed digits (n'_i, n'_{i-1}). However, there is a problem to be resolved that is how

to deal with the carries produced during the computation. When we process the digits $(n_i,\ n_{i-1})$, it is probable that a carry is produced to make us adjust the previous digits $(n'_{i+2},\ n'_{i+1})$. Generally, this maybe come into a long carry chain, which makes it impossible to generate the signed-digit representation from left to right. For the sake of simplicity, we resolve this problem by extending the digit set of the rNAF to $\{0, \pm 1, \ldots, \pm\lfloor\frac{r^2-1}{2}\rfloor, \pm(\lfloor\frac{r^2-1}{2}\rfloor + 1)\} \setminus \{\pm 1r, \ldots, \pm\lfloor\frac{r-1}{2}\rfloor r, \pm\left(\lfloor\frac{r-1}{2}\rfloor + 1\right) r\}$. When the radix r is even, the digit set does not be increased since that $\pm(\lfloor\frac{r^2-1}{2}\rfloor + 1)$ is equal to $\pm\left(\lfloor\frac{r-1}{2}\rfloor + 1\right) r$. When the radix r is odd, only two digits $\pm(\lfloor\frac{r^2-1}{2}\rfloor + 1)$ are added into the digit set (Indeed, this requires only one more stored point for point multiplication over elliptic curves since that the inverse of a point is easily computed.)

Now we can obtain the two signed digits $(n'_{i+2},\ n'_{i+1})$ as soon as we calculate the carry of $(n_i,\ n_{i-1})$. The main operations are seen from Step 3.1 to 3.3.4 in the following Algorithm 3. Furthermore, we can improve this method by reducing the non-zero digits. There are two kinds of cases that can be processed further. One kind of case is that two adjacent signed digits $(n'_{i+3},\ n'_{i+2})$ are both non-zero. The other kind of case is that four adjacent signed digits $(n'_{i+5},\ n'_{i+4},\ n'_{i+3},\ n'_{i+2})$ may be recoded into the representations with less non-zero digits. These improvements can be seen from Step 3.4.1 to 3.4.6 in Algorithm 3. Finally, we need to deal with the last two signed digits separately, which are implemented from Step 4 to 5 in Algorithm 3.

3.2 The Proposed Algorithm

The function $sign(x)$ denotes the sign of the integer variable x such that $sign(x)$ is equal to $x/|x|$, where $|x|$ denotes the absolute value of an integer x. The proposed algorithm is described in the following Algorithm 3.

4 Analysis and Comparison

4.1 Analysis

We can prove the following theorem on the average non-zero density of the radix-r signed-digit representation in Algorithm 3.

Theorem 3. *The average non-zero density of the radix-r signed-digit representation in Algorithm 3 is asymptotically* $\frac{1}{2} \times \left(1 - \frac{1}{r^2} - \frac{1}{(r+1)^2} - \frac{r-1}{r^2(r+1)^2}\right)$, *i.e.* $\frac{1}{2} - \frac{2r+3}{2r(r+1)^2}$.

Proof. Algorithm 3 scans the radix-r representation $(n_{k-1}, n_{k-2}, \ldots, n_0)$ of the integer n from left to right. When two adjacent digits (n_i, n_{i-1}) are processed, the signed digits (n'_{i+2}, n'_{i+1}) are outputted, where at least one of (n'_{i+2}, n'_{i+1}) is 0 (Note that this property does not mean the Non-adjacent Form since that $i = (k-1) - 2j$, where j is positive integer.). Hence the non-zero density of $(n'_k, n'_{k-1}, \ldots, n'_0)$ must be less than 1/2. Indeed, for some cases, the adjacent

Algorithm 3. New left-to-right radix-r signed-digit recoding algorithm

Input: The integer $n = (n_{k-1}, \ldots, n_1, n_0) = \sum_{i=0}^{k-1} n_i \times r^i$, where $n_i \in \{0, 1, \ldots, r-1\}$ and $n_{k-1} \neq 0$.

Output: The representation $n = (n'_k, n'_{k-1}, \ldots, n'_1, n'_0) = \sum_{i=0}^{k} n'_i \times r^i$, where $n_i \in \{0, \pm 1, \ldots, \pm\lfloor \frac{r^2-1}{2} \rfloor, \pm(\lfloor \frac{r^2-1}{2} \rfloor + 1)\} \setminus \{\pm 1r, \pm 2r, \ldots, \pm\lfloor \frac{r-1}{2} \rfloor r, \pm\left(\lfloor \frac{r-1}{2} \rfloor + 1\right) r\}$.

1. $\Delta \leftarrow 0$;
   /* *The Δ stores the value of $(n'_{i+2} \times r + n'_{i+1})$ to be processed.* */
2. $i \leftarrow k - 1$;
3. while $i \geq 1$ do
   /* *Note: From Step 3.1 to 3.3.4, complete the main operations.* */
   3.1 { $temp \leftarrow n_i \times r + n_{i-1}$;
   3.2 if $temp \leq \lfloor \frac{r^2-1}{2} \rfloor$ then
   3.2.1 { if $r|\Delta$ then $(n'_{i+2}, n'_{i+1}) \leftarrow (\Delta/r, 0)$;
   3.2.2 else $(n'_{i+2}, n'_{i+1}) \leftarrow (0, \Delta)$;
   3.2.3 $\Delta \leftarrow temp$; }
   3.3 else
   3.3.1 {$\Delta \leftarrow \Delta + 1$;
   3.3.2 if $r|\Delta$ then $(n'_{i+2}, n'_{i+1}) \leftarrow (\Delta/r, 0)$;
   3.3.3 else $(n'_{i+2}, n'_{i+1}) \leftarrow (0, \Delta)$;
   3.3.4 $\Delta \leftarrow temp - r^2$; }
   end if
   /* *Note: From Step 3.4.1 to 3.4.6, reduce the non-zero digits further.* */
   3.4 if $(n'_{i+3} \neq 0)$ and $(n'_{i+2} \neq 0)$ then
   3.4.1 { $temp \leftarrow n'_{i+3} \times r + n'_{i+2}$;
   3.4.2 if $(|temp| \leq \lfloor \frac{r^2-1}{2} \rfloor + 1)$ then
   3.4.3 $(n'_{i+3}, n'_{i+2}) \leftarrow (0, temp)$;
   3.4.4 else if $n'_{i+5} = -sign(temp)$ then
   3.4.5 { $(n'_{i+5}, n'_{i+4}) \leftarrow (0, n'_{i+5} \times r + sign(temp))$;
   3.4.6 $(n'_{i+3}, n'_{i+2}) \leftarrow (0, temp - r^2)$;}
   end if
   end if
   3.5 $i \leftarrow i - 2$;}
   end while
   /* *From Step 4 to Step 5.3.1, process the last two digits n'_1 and n'_0.* */
4. if $r|\Delta$ then
   4.1 $(n'_{i+2}, n'_{i+1}) \leftarrow (\Delta/r, 0)$;
   4.2 else $(n'_{i+2}, n'_{i+1}) \leftarrow (0, \Delta)$;
5. if $i = 0$ then
   5.1 { $n'_0 \leftarrow n_0$;
   5.2 $\Delta \leftarrow n'_1 \times r + n'_0$;
   5.3 if $(n'_1 \neq 0)$ and $(n'_0 \neq 0)$ and $(|\Delta| \leq \lfloor \frac{r^2-1}{2} \rfloor + 1)$ then
   5.3.1 $(n'_1, n'_0) = (0, \Delta)$;}
6. return $(n'_k, n'_{k-1}, \ldots, n'_1, n'_0)$.

signed digits (n'_{i+2}, n'_{i+1}) may be both zero. Therefore, we need to count the probability that (n'_{i+2}, n'_{i+1})=(0, 0). We assume that each digit n_i of the standard radix-r representation of the integer n is randomly distributed in the digit set $\{0, 1, \ldots, r-1\}$, which satisfies that $\Pr[n_i = 0]=\Pr[n_i = 1]=\ldots=\Pr[n_i = r-1]=1/r$. There are four kinds cases, which are described as follows. In Case 1 and 2, (n'_{i+2}, n'_{i+1})=(0, 0) may occur in Step 3.2.1 and 3.3.2 in Algorithm 3. In Case 3 and Case 4, (n'_{i+4}, n'_{i+3})=(0, 0) or (n'_{i+6}, n'_{i+5})=(0, 0) may occur in Step 3.4.3 and 3.4.5 respectively in Algorithm 3.

Case 1: When $(n_{i+2}, n_{i+1}) = (0, 0)$ and $(n_i, n_{i-1}) \leq \lfloor \frac{r^2-1}{2} \rfloor$, we have that $(n'_{i+2}, n'_{i+1}) = (0, 0)$ since that no carry is produced by (n_i, n_{i-1}). Since that $\Pr[(n_{i+2}, n_{i+1}) = (0, 0)] = \frac{1}{r^2}$, we obtain that $\Pr[(n_i, n_{i-1}) \leq \lfloor \frac{r^2-1}{2} \rfloor] = \frac{1}{2}$. Therefore, $\Pr[(n_{i+2}, n_{i+1}) = (0, 0)$ and $(n_i, n_{i-1}) \leq \lfloor \frac{r^2-1}{2} \rfloor] = \frac{1}{2r^2}$.

Case 2: When $(n_{i+2}, n_{i+1}) = (r-1, r-1)$ and $(n_i, n_{i-1}) \geq \lfloor \frac{r^2-1}{2} \rfloor + 1$, a carry is outputted and $(n'_{i+2}, n'_{i+1}) = (0, 0)$. Similar as Case 1, we have $\Pr[(n_{i+2}, n_{i+1}) = (r-1, r-1)$ and $(n_i, n_{i-1}) \leq \lfloor \frac{r^2-1}{2} \rfloor] = \frac{1}{2r^2}$.

Case 3: When $(n'_{i+4}, n'_{i+3}) = (0, \alpha), (n'_{i+2}, n'_{i+1}) = (\beta, 0)$ and $|\alpha \cdot r + \beta| \leq \lfloor \frac{r^2-1}{2} \rfloor + 1$, we have $(n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (0, 0, \alpha \cdot r + \beta, 0)$.

Let $x = \lfloor \frac{r^2-1}{2} \rfloor$ and $y = \lfloor \frac{r-1}{2} \rfloor$. It can be obtained that $\Pr[(n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (0, \alpha, \beta, 0)$, where $\alpha \neq 0$ and $\beta \neq 0]=\frac{x-y}{x} \times \frac{y}{x} = \frac{xy-y^2}{x^2}$. If $|\alpha \cdot r + \beta| \leq \lfloor \frac{r^2-1}{2} \rfloor + 1$ holds, we have $|\alpha| \leq \lfloor \frac{r-1}{2} \rfloor$ and $|\beta| \leq \lfloor \frac{r-1}{2} \rfloor$ with the probability $\frac{y}{x-y}$. Therefore, we obtain that

$$\Pr[(n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (0, \alpha, \beta, 0) \text{ and } |\alpha \cdot r + \beta| \leq \lfloor \tfrac{r^2-1}{2} \rfloor + 1]$$
$$= \frac{xy-y^2}{x^2} \times \frac{y}{x-y} = \left(\frac{\lfloor \frac{r-1}{2} \rfloor}{\lfloor \frac{r^2-1}{2} \rfloor} \right)^2 = \frac{1}{(r+1)^2}.$$

Case 4: When $(n'_{i+5}, n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (-sign(\alpha \cdot r + \beta), 0, \alpha, \beta, 0)$ and $|\alpha \cdot r + \beta| \geq \lfloor \frac{r^2-1}{2} \rfloor + 2$, we have $(n'_{i+5}, n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (0, n'_{i+5} \times r + sign(temp), 0, temp - r^2)$.

Note that $\Pr[(n'_{i+5} = -sign(\alpha \cdot r + \beta) = \pm 1] = \frac{1}{r^2}$. Let $x = \lfloor \frac{r^2-1}{2} \rfloor$ and $y = \lfloor \frac{r-1}{2} \rfloor$. According to Case 3, we have $\Pr[(n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (0, \alpha, \beta, 0)$, where $\alpha \neq 0$ and $\beta \neq 0]=\frac{x-y}{x} \times \frac{y}{x} = \frac{xy-y^2}{x^2}$. If $|\alpha \cdot r + \beta| \geq \lfloor \frac{r^2-1}{2} \rfloor + 2$ holds, we have $|\alpha| \geq \lfloor \frac{r-1}{2} + 1$ with the probability $1 - \frac{y}{x-y}$. Therefore, we obtain that

$$\Pr[(n'_{i+5}, n'_{i+4}, n'_{i+3}, n'_{i+2}, n'_{i+1}) = (-sign(\alpha \cdot r + \beta), 0, \alpha, \beta, 0) \text{ and } |\alpha \cdot r + \beta| \geq \lfloor \tfrac{r^2-1}{2} \rfloor + 2] = \frac{xy-y^2}{x^2} \times \left(1 - \frac{y}{x-y}\right) \times \frac{1}{r^2} = \frac{r-1}{r^2(r+1)^2}.$$

According to Case 1, 2, 3 and 4, we obtain that the probability of two adjacent zeros is $\left(\frac{1}{r^2} + \frac{1}{(r+1)^2} + \frac{r-1}{r^2(r+1)^2}\right)$. Therefore, the average non-zero density of the radix-r signed-digit representation obtained by Algorithm 3 is asymptotically $\frac{1}{2} \times \left(1 - \frac{1}{r^2} - \frac{1}{(r+1)^2} - \frac{r-1}{r^2(r+1)^2}\right)$, i.e. $\frac{1}{2} - \frac{2r+3}{2r(r+1)^2}$. □

4.2 Comparison

We give a comparison of various radix-r signed-digit recoding representations such as the GSF [19], Muir's algorithm [20] and the rNAF [22] in Table 1. Note that the rNAF recoding algorithm is the only right-to-left algorithm while the others are left-to-right ones.

When the radix r=3, the average non-zero density of Algorithm 3 is 0.406, which is close to that of the rNAF and reduced by nearly 20% compared with the GSF and Muir's algorithm. When $r \to \infty$, the average non-zero density of Algorithm 3 and the rNAF is nearly 0.5 in contrast with 1 of the GSF and Muir's algorithm.

Table 1. Comparison of left-to-right radix-r signed-digit recoding algorithms

Representation	Left-to-right	Non-zero density (radix-r)	radix-3
GSF	YES	$\frac{r-1}{r+1}$	0.5
Muir's	YES	$\frac{r-1}{r+1}$	0.5
rNAF	NO	$\frac{r-1}{2r-1}$	0.4
Algorithm 3	YES	$\frac{1}{2} - \frac{2r+3}{2r(r+1)^2}$	0.406

5 Conclusion

We have propose a new left-to-right radix-r signed-digit recoding algorithm, which has the expected non-zero density $\frac{1}{2} \times \left(1 - \frac{1}{r^2} - \frac{1}{(r+1)^2} - \frac{r-1}{r^2(r+1)^2}\right)$, i.e. $\frac{1}{2} - \frac{2r+3}{2r(r+1)^2}$, close to the $\frac{r-1}{2r-1}$ of the rNAF. Moreover, while the rNAF representation is computed from right to left, our proposed algorithm can be processed from left to right and reduce the space requirement in the implementation of pairing-based cryptosystems. An interesting problem is how to develop a left-to-right radix-r signed-digit recoding algorithm, which can attain the same non-zero density $\frac{r-1}{2r-1}$ as the rNAF.

Acknowledgments. Supported by National 863 High-tech Research and Development Program of China (2003AA141120, 2004AA001260); National 973 Grand Fundamental Research Program of China (G1999035802). The authors would like to thank Professor Shaohan Ma and three graduates Baodong Qin, Ming Li and Guangqing Zhang for their interesting discussions. The authors are very grateful to the anonymous referees for their valuable comments and suggestions.

References

1. A. Joux, "A one-round protocol for tripartite Diffie-Hellman," Algorithm Number Theory Symposium - ANTS IV, Lecture Notes in Computer Science 1838, pp. 385-394, Springer-Verlag, 2000.
2. D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," Advances in Cryptology - CRYPTO 2001, Lecture Notes in Computer Science 2139, pp.213-229, Springer-Verlag, 2001.

3. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," ASIACRYPT 2001, Lecture Notes in Computer Science 2248, pp.514-532, Springer-Verlag, 2002.
4. V. S. Miller. Short programs for functions on curves. Unpublished manuscript, 1986. Available at: `http://crypto.stanford.edu/miller/miller.pdf`.
5. P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems," CRYPTO 2002, Lecture Notes in Computer Science 2442, pp.354-368, Springer-Verlag, 2002.
6. S. Galbraith, K. Harrison, and D. Soldera, "Implementing the Tate pairing," ANTS V, Lecture Notes in Computer Science 2369, pp.324-337, Springer-Verlag, 2002.
7. N. Smart, and J.Westwood, "Point Multiplication on Ordinary Elliptic Curves over Fields of Characteristic Three," Applicable Algebra in Engineering, Communication and Computing, Vol.13, No.6, pp.485-497, 2003.
8. I. Duursma and H -S .Lee, "Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$," ASIACRYPT 2003, Lecture Notes in Computer Science 2894, pp.111-123, Springer-Verlag, 2003.
9. A. D. Booth, "A Signed Binary Multiplication Technique," Q. J. Mech. Appl. Math., Vol. 4, No. 2, pp.236-240, 1951.
10. G. W. Reitwiesner, "Binary arithmetic", Advances in Computers, Vol. 1, pp. 231-308, 1960.
11. W. Clark and J. Liang, "On Arithmetic Weight for a General Radix Representation of Integers," IEEE Transaction on IT, IT-19, pp.823-826, 1973.
12. S. Arno and F. S. Wheeler, "Signed digit representations of minimal hamming weight," IEEE Transactions on Computers, Vol. 42, No. 8, pp.1007-1010, 1993.
13. J. A. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," CRYPTO 1997, Lecture Notes in Computer Science 1294, pp.357-371. Springer-Verlag, 1997.
14. V. Müller, "Fast Multiplication on Elliptic Curves over Small Fields of Characteristic Two," Journal of Cryptology, No.11, pp. 219-234, 1998.
15. M. Joye, and S.-M. Yen, "Optimal left-to-right binary signed-digit recoding". IEEE Trans. on Comp, Vol. 49, No. 7, pp.740-748, 2000.
16. R. M. Avanzi, "A Note on the Signed Sliding Window Integer Recoding and a Left-to-Right Analogue," SAC 2004, Lecture Notes in Computer Science 3357, pp.130-143, Springer-Verlag, 2005.
17. Katsuyuki Okeya, Katja Schmidt-Samoa, Christian Spahn, Tsuyoshi Takagi, "Signed Bi-nary Representations Revisited", CRYPTO 2004, Lecture Notes in Computer Science 3152, pp.123-139, Springer-Verlag, 2004.
18. J. A. Muir, and D. R. Stinson, "New Minimal Weight Representations for Left-to-Right Window Methods," CT-RSA 2005, Lecture Notes in Computer Science 3376, pp.366-383, Springer-Verlag , 2005.
19. M. Joye and S. -M. Yen, "New Minimal Modified Radix-r Representation with Applications to Smart Cards," PKC 2002, Lecture Notes in Computer Science 2274, pp.375-384, Springer-Verlag, 2002.
20. J. A. Muir, "A Simple Left-to-Right Algorithm for Minimal Weight Signed Radix-r Representations," IEEE Transactions on Information Theory, Vol.53, No.3, pp.1234-1241, 2007.
21. F. Kong, J. Yu, Z. Cai and D. Li, "Left-to-right Generalized Non-adjacent Form Recoding for Elliptic Curve Cryptosystems," The First International Conference on Hybrid Information Technology, IEEE Press, pp.299-303, 2006.
22. T. Takagi, S.-M. Yen, B.-C. Wu, "Radix-r Non-Adjacent Form," ISC 2004, Lecture Notes in Computer Science 3225, pp.99-110, Springer-Verlag, 2004.

23. F. Kong, D. Li, "A Note on Signed Binary Window Algorithm for Elliptic Curve Cryptosystems," Cryptology and Network Security-CANS 2005, Lecture Notes in Computer Science 3810, pp.223-235, Springer-Verlag, 2005.
24. D. M. Gordon, "A Survey of Fast Exponentiation Methods," Journal of Algorithms, Vol. 27 , pp.129-146,1998.
25. B. Phillips and N. Burgess, "Minimal Weight Digit Set Conversions," IEEE Transactions on Computers, Vol.53, No.6, pp.666-677, 2004.

The Strongest Nonsplitting Theorem

Mariya Ivanova Soskova* and S. Barry Cooper**

University of Leeds, Leeds, LS2 9JT, UK

1 Introduction

Sacks [14] showed that every computably enumerable (c.e.) degree $\geq \mathbf{0}$ has a c.e. splitting. Hence, relativising, every c.e. degree has a Δ_2 splitting above each proper predecessor (by 'splitting' we understand 'nontrivial splitting'). Arslanov [1] showed that $\mathbf{0}'$ has a d.c.e. splitting above each c.e. $\mathbf{a} < \mathbf{0}'$. On the other hand, Lachlan [9] proved the existence of a c.e. $\mathbf{a} > \mathbf{0}$ which has no c.e. splitting above some proper c.e. predecessor, and Harrington [8] showed that one could take $\mathbf{a} = \mathbf{0}'$. Splitting and nonsplitting techniques have had a number of consequences for definability and elementary equivalence in the degrees below $\mathbf{0}'$.

Heterogeneous splittings are best considered in the context of cupping and noncupping. Posner and Robinson [13] showed that every nonzero Δ_2 degree can be nontrivially cupped to $\mathbf{0}'$, and Arslanov [1] showed that every c.e. degree $> \mathbf{0}$ can be d.c.e. cupped to $\mathbf{0}'$ (and hence since every d.c.e., or even n-c.e., degree has a nonzero c.e. predecessor, every n-c.e. degree $> \mathbf{0}$ is d.c.e. cuppable.) Cooper [2] and Yates (see Miller [11]) showed the existence of degrees noncuppable in the c.e. degrees. Moreover, the search for relative cupping results was drastically limited by Cooper [3], and Slaman and Steel [15] (see also Downey [7]), who showed that there is a nonzero c.e. degree $\mathbf{a}$ below which even Δ_2 cupping of c.e. degrees fails.

We prove below what appears to be the strongest possible of such nonsplitting and noncupping results.

Theorem 1. *There exists a computably enumerable degree* $\mathbf{a} < \mathbf{0}'$ *such that there exists no nontrivial cuppings of c.e. degrees in the* Δ_2 *degrees above* $\mathbf{a}$.

In fact, if we consider the extended structure of the enumeration degrees, Theorem 1 is a corollary of the even stronger result:

Theorem 2. *There exists a* Π_1 *e-degree* $\mathbf{a} < \mathbf{0}'_e$ *such that there exist no nontrivial cuppings of* Π_1 *e-degrees in the* Δ_2 *e-degrees above* $\mathbf{a}$.

This would appear to be the first example of a structural feature of the Turing degrees obtained via a proof in the wider context of the enumeration degrees (rather than the other way round).

Notation and terminology below is based on that of [5].

* The first author was partially supported by the Marie Curie Early Training Site MATHLOGAPS (MEST-CT-2004-504029).

** The second author was supported by EPSRC grant No. GR /S28730/01, and by the NSFC Grand International Joint Project, No. 60310213, *New Directions in the Theory and Applications of Models of Computation.*

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 199–211, 2007.
© Springer-Verlag Berlin Heidelberg 2007

2 Requirements and Strategies

We assume a standard listing of all quadruples (Ψ, Θ, U, W) of enumeration operators Ψ and Θ, Σ_2 sets U and c.e. sets W. We will construct Π_1 sets A and E to satisfy the corresponding list of requirements:

$$N_\Psi: \quad E \neq \Psi^A,$$
$$P_{\Theta,U,W}: \quad E = \Theta^{U,\overline{W}} \wedge U \in \Delta_2 \Rightarrow (\exists \Gamma, \Lambda)[\overline{K} = \Gamma^{U,A} \vee \overline{K} = \Lambda^{\overline{W},A}],$$

where $\Gamma^{U,A}$, for example, denotes an e-operator enumerating relative to the data enumerated from two sources U and A. We describe the basic strategy to satisfy these requirements, only using $U \in \Delta_2$ for satisfying P in the case of a successful Γ-strategy.

2.1 The Naive N_Ψ-Strategy

Select a witness x for N_Ψ and wait for $x \in \Psi^A$. Then extract x from E while restraining each $y \in A \upharpoonright use(\Psi, A, x)$ (the use function $use(\Psi, A, x)$ is defined in the usual way by $use(\Psi, A, x) = \mu y[x \in \Psi^{A\upharpoonright y}]$).

2.2 The Naive P_Θ-Strategy

Definition 1. *Let Φ be an enumeration operator and A a set. We will consider a generalised use function φ defined as follows:*

$$\varphi(x) = \max\left\{use(\Phi, A, y) | (y \leq x) \wedge (y \in \Phi^A)\right\}$$

We progressively try to rectify Γ_Θ at each stage by ensuring that $z \in \overline{K} \Leftrightarrow z \in \Gamma^{U,A}$ for each z below $l(E, \Theta^{U,\overline{W}})$. The definition of the enumeration operator Γ involves axioms with two types of markers $u(z) \in U$ and $\gamma(z) \in A$ - the generalised use functions for the operator Γ. Given a suitable choice of a marker $\gamma(z) \in A$ if $z \nearrow \overline{K}$, Γ can be rectified via A-extraction.

2.3 N_Ψ Below P_Θ

In combining these two strategies the A-restraint for N_Ψ following the extraction of x from E conflicts with the need to rectify Γ_Θ. We try to resolve this by choosing a threshold d for N_Ψ, and try to achieve $\gamma(z') > use(\Psi, A, x)$ for all $z' \geqslant d$ at a stage previous to the imposition of the restraint. We try to maintain $\theta(x) < u(d)$, in the hope that after we extract x from E, each return of $l(E, \Theta^{U,\overline{W}})$ will produce an extraction from $U \upharpoonright \theta(x)$ which can be used to avoid an A-extraction in moving $\gamma(d)$.

In the event that some such attempt to satisfy N_Ψ ends with a $\overline{W} \upharpoonright \theta(x)$-change, then we must implement the $\Lambda_{\Theta,\Psi}$-strategy which is designed to allow lower priority N-requirements to work below the Γ_Θ-activity, using the $\overline{W} \upharpoonright \theta(x)$-changes thrown up by Γ_Θ to move $\Lambda_{\Theta,\Psi}$-markers. Each time we progress the

$\Lambda_{\Theta,\Psi}$-strategy, we cancel the current witness of (N_Ψ, Γ), and if this happens infinitely often, N_Ψ might not be satisfied. This means that N_Ψ must be accompanied by an immediately succeeding copy (N_Ψ, Λ), say, designed to take advantage of the improved strategy for N_Ψ without any other $P_{\Theta'}$ intervening between (N_Ψ, Γ) and (N_Ψ, Λ).

2.4 The Approximations

During the construction we approximate the given sets at each stage. We need to choose these approximating sequences very carefully. Consider a $P_{\Theta,U,W}$ requirement.

Definition 2. *We inductively say that a stage $s+1$ is P_Θ-expansionary if and only if $l(E[s+1], \Theta^{U,\overline{W}}[s+1])$ attains a greater value at stage $s+1$ than at any previous P_Θ-expansionary stage.*

If the length of agreement is bounded, $P_{\Theta,U,W}$ is trivially satisfied and we do not have to act on its behalf. The strategies act only on $P_{\Theta,U,W}$- expansionary stages. It is essential that if $P_{\Theta,U,W}$ turns out to be equal to E we have infinitely many expansionary stages. We will work with a good approximating sequence to $U \oplus \overline{W}$ (basically one with sufficient thin stages, in the sense of Cooper [4]) as defined in [10]:

Consider a Δ_2 approximating sequence $\{U'_s\}$ to U and the standard approximating sequence $\{W'_s\}$ to the c.e. set W. We define $\left\{\overline{W}^*_s\right\}$ to be the Δ_2 approximating sequence to $\overline{W}$ constructed in the following way: $\overline{W}^*_s = \overline{W'_s} \upharpoonright s$. Joining the two Δ_2 approximating sequences, we get $\left\{U'_s \oplus \overline{W}^*_s\right\}$ – that is, a Δ_2 approximating sequence to $U \oplus \overline{W}$. It follows from [10] that we can construct a good approximating sequence to $U \oplus \overline{W}$ in the following way: $(U \oplus \overline{W})_s = U'_s \oplus \overline{W}^*_s \upharpoonright (\mu n[U'_s \oplus \overline{W}^*_s(n) \neq U'_{s+1} \oplus \overline{W}^*_{s+1}(n)])$. The resulting good approximating sequence has the following properties:

1. $\forall n \exists s (U \oplus \overline{W} \upharpoonright n \subseteq (U \oplus \overline{W})_s \subseteq U \oplus \overline{W})$. Such stages are called good stages and hence there are infinitely many of them.
2. $\forall n \exists s_0 \forall s > s_0 (U \oplus \overline{W} \upharpoonright n = (U \oplus \overline{W})_s \upharpoonright n)$
3. If G is the set of all good stages of the approximation, then $\forall n \exists s_0 \forall s > s_0 (s \in G \Rightarrow \Theta_s((U \oplus \overline{W})_s) \upharpoonright n = \Theta(U \oplus \overline{W}) \upharpoonright n)$. This is also a result from [10].

From these properties and the fact that E is a Π_1 set, we can conclude that if $\Theta(U \oplus \overline{W}) = E$ then there are infinitely many expansionary stages.

We will use more information about the sequence $\left\{(U \oplus \overline{W})_s\right\}$ – it will be considered as a pair $((U \oplus \overline{W})_s, ap_s) = (U'_s \oplus \overline{W}^*_s \upharpoonright (\mu n[U'_s \oplus \overline{W}^*_s(n) \neq U'_{s+1} \oplus \overline{W}^*_{s+1}(n)]), \mu n[U'_s \oplus \overline{W}^*_s(n) \neq U'_{s+1} \oplus \overline{W}^*_{s+1}(n)])$.

We will modify the definition of expansionary stages to incorporate the trueness of the approximations.

Definition 3. *We inductively say that a stage $s+1$ is P_Θ-expansionary if and only if $l(E[s+1], \Theta^{U,\overline{W}}[s+1])$ attains a greater value at stage $s+1$ than at any previous P_Θ-expansionary stage and $ap_{s+1} > l(E[s+1], \Theta^{U,\overline{W}}[s+1])$.*

Note that if U is a properly Σ_2 set, then we can still obtain a modified approximation to it in the way described above, but will not need to satisfy its requirement P in that case.

2.5 The Basic Module for One P_Θ- and One N_Ψ- Requirement

The (P_Θ, Γ)-strategy tries to maintain the equality between $\overline{K}$ and $\Gamma^{U,A}$ at expansionary stages. It scans elements $n < l(\Theta^{U,\overline{W}}, E)$ fixing their axioms as appropriate.

Every strategy works below a right boundary R, assuming that as the stages grow the right boundary will grow unboundedly.

(P_Θ, Γ) builds an operator Γ by defining marker $u_s(n)$ and $\gamma_s(n)$ and corresponding axioms for elements $n \in \overline{K}$ of the form $\langle n, U_s \upharpoonright u_s(n), A_s \upharpoonright \gamma_s(n)\rangle$ at stage s.

It may happen that the two strategies (P_Θ, Γ) and (P_Θ, Λ) influence each other by extracting markers from A. In order to prevent that we define two nonintersecting infinite computable sets A_G and A_L for the possible values of A-markers for (P_Θ, Γ) and (P_Θ, Λ) respectively. Each time (P_Θ, Γ) defines a new marker for some n, it defines $\gamma(n)$ big (bigger than any number that appeared in the construction until now) and $\gamma(n) \in A_G$.

Each time (P_Θ, Λ) defines a new marker for some n, it defines $\lambda(n)$ big and $\lambda(n) \in A_L$.

We will describe the modules in a more general way, so that we can use them later in the construction involving all requirements.

The (P_Θ, Γ) - Strategy

1. Wait for an expansionary stage. $(o = l)$
2. Choose $n < l(\Theta^{U,\overline{W}}, E)$ in turn $(n = 0, 1, \dots)$ and perform the following actions:
   - If $u(n)\uparrow$, then define it anew as $u(n) = u(n-1)+1$ (if $n = 0$, then define $u(n) = 1$). If $u(n)$ is defined, but $ap_s < u(n)$ skip to the next element.
   - If $n \in \overline{K}$:
     - If $\gamma(n)\uparrow$, then define it anew, and define an axiom $\langle n, (U \upharpoonright u(n)+1, A \upharpoonright \gamma(n)+1)\rangle \in \Gamma$.
     - If $\gamma(n)\downarrow$, but $\Gamma^{(U,A)}(n) = 0$ (due to a change in U or in A), then enumerate the old axiom in a special parameter $Old(n)$ – this being a collection of all axioms that might later on turn out to be valid. The element enumerated in $Old(n)$ is of the form $(\gamma(n), \langle n, (U \upharpoonright u(n)+1, A \upharpoonright \gamma(n)+1)\rangle)$ – the pair of the old marker and old axiom. Then define $\gamma(n)$ anew and define an axiom $\langle n, (U \upharpoonright u(n)+1, A \upharpoonright \gamma(n)+1)\rangle \in \Gamma$.

– If $n \notin \overline{K}$, but $n \in \Gamma^{(U,A)}$ then look through all axioms defined for n in $Old(n)$ and extract the $\gamma(n)$ markers for any axiom that is valid.

Module for (N_Ψ, Γ). The basic module acts only at P_Θ- expansionary stages. If there are only finitely many expansionary stages, then P_Θ is trivially satisfied and N_Ψ moves to a truer path through the tree of outcomes.

At the beginning of each stage we check if the thresholds are correct, i.e. if $\overline{K} \upharpoonright d$ has not changed since the last true stage. If so we initialize all strategies below this one and start from initialization.

– **Initialization**

1. If a threshold has not yet been defined or is cancelled, choose a new threshold d bigger than any defined until now.
2. If a witness has not yet been defined or is cancelled, choose a new witness $x > d$, $x \in E$.
3. Wait for $x < l(E, \Theta^{U,\overline{W}})$. $(o = w)$
4. Extract all markers $\gamma(d)$ – old and new – and empty the list $Old(n)$ for $n \geq d$. Define $u(d)$ anew, bigger than $\theta(x)$. This gives us control over any axiom enumerated in Γ for the elements we are monitoring.
5. For every element $y \leq x$, $y \in E$, enumerate into the list *Axioms* the current valid axiom from Θ that has been valid longest.

– **Honestification**

Scan the list *Axioms*. If for any element $y \leq x$, $y \in E$, the listed axiom is not valid anymore, then update the list *Axioms*, let (o = h) and

1. Extract $\gamma(d)$ from A for all markers of axioms – the current one and the old ones. Empty the list $Old(d)$. Redefine $u(d) = \max(\theta(x), u(d)) + 1$.
2. Cancel all markers $u(n)$ for $n > d$ and $n \in \overline{K}$. Empty the list $Old(n)$. Notice that the extraction of all markers $\gamma(d)$ guarantees that the old axioms for elements $n > d$ will never again be valid. Hence at the next expansionary stage $u(n)$ and $\gamma(n)$ will be defined anew, bigger than $\theta(x)$. This ensures the following property: for all elements $z \geq d$, $z \in \overline{K}$, the U-parts of axioms both old and new in Γ include the U-parts of all axioms listed in *Axioms* for elements $y \leq x$, $y \in E$.

Otherwise go to:

– **Waiting**

If Γ is honest, i.e. $u(d) > \theta(x)$, and all the axioms enumerated in *Axioms* have remained unchanged since the last stage, then wait for $x \in \Psi^A$ with $use(\Psi, A, x) < R$, returning at each successive stage to Honestification (o = w).

– **Attack**

1. If $x \in \Psi^A$ and $u(d) > \theta(x)$, then extract x from E and restrain A on $use(\Psi, A, x)$. (o = g)
2. Wait until the length of agreement has returned and $ap_s > u(d)$.

3. Result –
   Let $x' \leq x$ be the least element that has been extracted from E during the stage of the Attack. When the length of agreement returns $x' \notin \Theta^{U,\overline{W}}$. Hence all axioms for x' in Θ are not valid, in particular the one enumerated in $Axioms$, say $\langle x', U_{x'}, \overline{W}_{x'} \rangle$.

   If $\overline{W}_{x'} \subset \overline{W}_s$ then the attack is successful and the activity at (P_Θ, Γ) lifts the γ-markers of all elements greater than d above the restraint to maintain $A \upharpoonright \psi(x)$. Note that this change affects not only the current axiom, but also all axioms enumerated in Old, because we insured that all possibly valid axioms in Γ – old and current – contain as a subset $U_{x'}$. If the change in U_x is permanent, then this will lead to success for N_Ψ. Otherwise the attack is unsuccessful, and we are forced to capriciously destroy Γ by extracting markers $\gamma(d)$ from A, and to start over with a bigger witness.
4. Successful attack: Then all valid axioms in Γ for $n \geq d$ have $\gamma(n) > use(\Psi, A, x)$. (o = f) Return to Result at the next stage. Note that the Σ_2-nature of the set U can trick us into believing that an attack is successful, whereas in fact it later turns out not to be. This is why we keep monitoring the witness and trying to prove that the attack is unsuccessful.
5. Unsuccessful attack: Extract all γ- markers $\gamma(d)$ from A for both the current and the old axioms. Empty $Old(n)$ for $n \geq d$. Remove the restraint on A. Cancel the current witness x. Return to Initialization at the next stage (choosing a new big enough witness) (o = g).

Analysis Of Outcomes: P_Θ has two possible outcomes:

[l] – there is a stage after which $l(\Theta^{U,\overline{W}}, E)$ remains bounded by its previous expansionary value say L. Then P_Θ is trivially satisfied as if U is Δ_2, then $\Theta^{U,\overline{W}} \neq E$. In this case we implement a simple 'Friedberg- Muchnik' strategy for N_Ψ working below $(R = \infty)$.

[e] – infinitely many expansionary stages, on which (N_Ψ, Γ) acts:

The possible outcomes of the (N_Ψ, Γ)- strategy are:

[w] – There is an infinite wait at Waiting for $\Psi^A(x) = 1$. Then N_Ψ is satisfied because $E(x) = 1 \neq \Psi^A(x)$ and the Γ_Θ-strategy remains intact. Successive strategies work below $R = \infty$)

[f] – There is a stage after which the last attack remains successful. At sufficiently large stages $\overline{K} \upharpoonright d$ has its final value. So there is no injury to the outcomes below f, $\Psi^A(x) = 1$, N_Ψ is satisfied, leaving the Γ_Θ- strategy intact. Successive strategies work below $(R = \infty)$

[h] – There are infinitely many occurrences of Honestification, precluding an occurrence of Attack. Then there is a permanent witness x, which has unbounded $lim_{sup}\theta(x)$. This means that $\Theta^{U,V}(y) = 0$ for some $y \leq x$,$y \in E$. Thus P_Θ is again satisfied. In this case we also implement a simple Friedberg–Muchnik strategy for N_Ψ working below $(R = \gamma(d))$.

[g] – We implement the unsuccessful attack step infinitely often. As anticipated, we must activate the $\Lambda_{\Theta,\Psi}$-strategy for P_Θ. N_Ψ is not satisfied, but we have a copy of N_Ψ designed to take advantage of the switch of strategies for P_Θ below N_Ψ. It works below $(R = x)$.

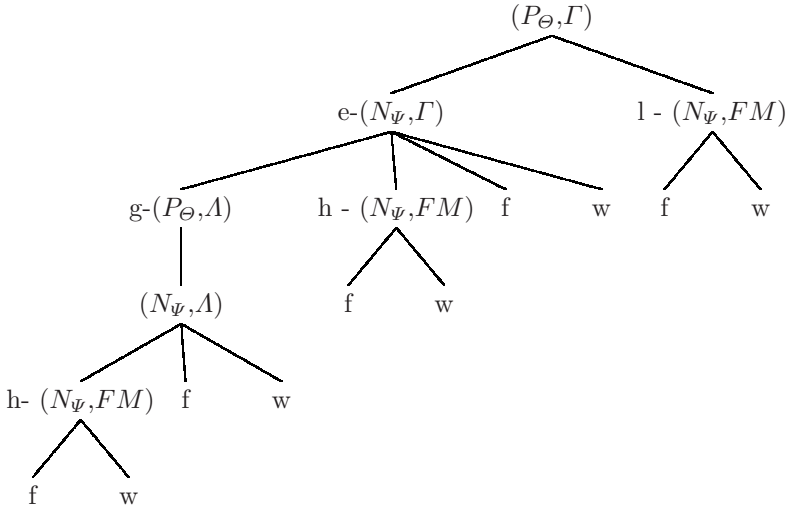
Module for the Strategies Below Outcome *g*: Notice that the outcome g is visited in two different cases – at the beginning of an attack and when the attack turns out to be unsuccessful. The first case starts a nonactive stage for the subtree below g, allowing the other N-strategies to synchronize their attacks. The second case starts an active stage for the strategies in the subtree below g.

The (P_Θ, Λ) acts only on active stages in a similar but less complicated way than (P_Θ, Γ). Namely it does not have a list Old as any change in $\overline{W} \restriction ap_s$ is permanent.

The (N_Ψ, Λ)-strategy is again similar to the (N_Ψ, Γ)-strategy. It has its own threshold $\widehat{d} > d$ and witness $\widehat{x} < x$. It has the Initialization, Honestification and Waiting modules which it executes on active stages. The corresponding outcomes are h and w.

It attacks on nonactive stages. The next stage at which this strategy is accessible after an attack will be an active stage – an unsuccessful attack for (N_Ψ, Γ). Note that the least element extracted during the attack is $x' \leq \widehat{x} < x$. So we have a $\overline{W} \restriction \theta(\widehat{x})$ - change. Hence there will be no unsuccessful attacks and no outcome g, but only successful attacks and outcome f.

The tree of outcomes at this point looks as follows:



It is worth noticing that the outcomes on the tree, strictly speaking, are outcomes relating to strategies, rather than outcomes telling us exactly how the requirement is satisfied, and these subsume the "not Δ_2" case of the P-requirements. The properly Σ_2 case only needs to be specially factored in when one considers in the verification what the strategies deliver.

2.6 All Requirements

When all requirements are involved the construction becomes more complicated. We will start by describing the tree of outcomes.

The requirements are ordered in the following way:

$$N_0 < P_0 < N_1 < P_1 \dots$$

Each P-requirement has at least one node along each path in the tree. Each N-requirement has a whole subtree of nodes along each path, the size of which depends on the number of P-requirements of higher priority.

Consider the requirement N_i. It has to clear the markers from A of i P-requirements $P_0, P_1, \dots P_{i-1}$. Each of them can follow one of the three strategies (N_Ψ, Γ_i), (N_Ψ, Λ_i) or (N_Ψ, FM_i). There will be nodes for each of the possible combinations in the subtree.

We distinguish between the following strategies:

1. For every P_i-requirement we have two different strategies: (P_i, Γ) with outcomes $e <_L l$ and (P_i, Λ) with one outcome s.
2. For every N_i-requirement, where $i > 0$, we have strategies of the form $(N_i, S_0, \dots S_{i-1})$, where $S_j \in \{\Gamma_j, \Lambda_j, FM_j\}$. They are all equipped with working boundaries (L, R). The requirement N_0 has one strategy (N_0, FM) with $(L = 0, R = \infty)$. The outcomes are f, w and for each $j < i$ if $S_j \in \{\Gamma_j, \Lambda_j\}$ there is an outcome h_j, if $S_j = \Gamma_j$, there is an outcome g_j. They are ordered according to the following rules:
   - For all j_1 and j_2, $g_{j_1} <_L h_{j_2} <_L f <_L w$
   - If $j_1 < j_2$ then $g_{j_2} <_L g_{j_1}$ and $h_{j_1} <_L h_{j_2}$.

   Let $\mathbb{O}$ be the set of all possible outcomes and $\mathbb{S}$ be the set of all possible strategies.

Definition 4. *The tree of outcomes is a computable function $T : D(T) \subset \mathbb{O}^* \to \mathbb{S}$ which has the following properties:*

1. $T(\emptyset) = (N_0, FM)$

2. $T(\alpha) = S$ and O_S is the set of outcomes for the strategy S, then for every $o \in O_S$, $\alpha\hat{}o \in D(T)$.

3.If $S = (N_i, S_0, S_1, \dots, S_{i-1})$, then

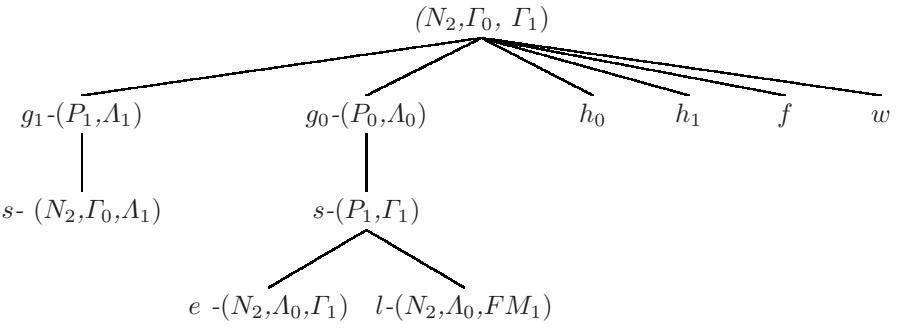
$T(\alpha\hat{}g_j) = (P_j, \Lambda_j)$ and $T(\alpha\hat{}g_j\hat{}s) = (P_{j+1}, \Gamma_{j+1}) \dots T(\alpha\hat{}g_j\hat{}s\hat{}o_{j+1}\hat{}o_{i-2}) = (P_{i-1}, \Gamma_{i-1})$, where $o_k \in \{e_k, l_k\}$ for $j + 1 \leq k \leq i - 2$.

All this means that, under an outcome g_j the strategy P_j starts its work on building the second possible functional Λ_j, and all strategies P_k for $k > j$ start their work from the beginning, i.e., start building the functional Γ_k again.

$T(\alpha\hat{}g_j\hat{}s\hat{}o_{j+1}\hat{}o_{i-1}) = (N_i, S_0, \dots, \Lambda_j, \dots, S_{i-1})$, where $S_k = \Gamma_k$ if $o_k = e_k$ and $S_k = FM_k$ if $o_k = l_k$ for every k such that $j < k < i$.

Then there is a copy of the strategy N_i which starts work with the old strategies S_l for $l < j$ and the new strategies S_k for $k \geq j$.

To illustrate this complicated definition here is a picture of this part of the tree in the simpler case of only two P- requirements.



The tree under outcome h_j is built in a similar fashion.

$T(\alpha\hat{}h_j) = (P_{j+1},\Gamma_{j+1})\ldots T(\alpha\hat{}h_j\hat{}o_{j+1}\hat{}o_{i-2}) = (P_{i-1},\Gamma_{i-1})$, *where* $o_k \in \{e_k,l_k\}$ *for* $j+1 \le k \le i-2$.

Hence all strategies S_k for $k > j$ start their work from the beginning, building a new functional Γ_k.

$T(\alpha\hat{}h_j\hat{}o_{j+1}\hat{}o_{i-1}) = (N_i,S_0,\ldots,FM_j,\ldots,S_{i-1})$, *where* $S_k = \Gamma_k$ *if* $o_k = e_k$ *and* $S_k = FM_k$ *if* $o_k = l_k$ *for every* k *such that* $j < k < i$.

$T(\alpha\hat{}f) = (P_i,\Gamma_i)$

$T(\alpha\hat{}w) = (P_i,\Gamma_i)$

Say $S = (P_i,\Gamma)$, *and* $\alpha = \alpha'\hat{}f$ *or* $\alpha = \alpha'\hat{}w$, *and* $T(\alpha') = (N_i,S_0,S_1,\ldots,S_{i_1})$. *It follows that this is not the case described and* (P_i,Γ) *appears for the first time, and then*

$T(\alpha\hat{}e) = (N_{i+1},S_0,\ldots,S_{i-1},\Gamma_i)$

$T(\alpha\hat{}l) = (N_{i+1},S_0,\ldots,S_{i-1},FM_i)$.

Interaction Between Strategies: In order to prevent unwanted interaction between the different strategies on different nodes we will do the following:

Different P-strategies define and extract different A-markers at stages of the construction. Extraction of markers for one P-strategy may influence the validity of axioms for another P-strategy. Again we deal with this problem by separating the A-markers for the different possible strategies. We have countably many different nodes in the tree of outcomes, whose values are P-strategy. For each such node α we define an infinite computable set A_α, from which the strategy $T(\alpha)$ can choose A-markers. If $\alpha \neq \beta$ then $A_\alpha \cap A_\beta = \emptyset$.

Similarly we define separate nonintersecting sets D_α and X_α for the different nodes on the tree which are labelled with N- strategies, from which they choose their thresholds and witnesses.

As usual we give higher priority to nodes that are to the left or higher up in the tree of strategies. This is achieved via two forms of initialization.

1. On each stage initialization is performed on all nodes that are bigger than the last node visited on that stage.
2. The second case in which initialization is performed is when the thresholds are not correct:

Every strategy α with $T(\alpha) = (N_i, S_1, \ldots, S_j, \ldots, S_{i-1})$ has a threshold d_j for each strategy S_j. If $\overline{K} \upharpoonright (d_j + 1)$ changes, then all successors of α that assume that d_j does not change infinitely many times are initialized. These are strategies γ such that $\gamma \supseteq \alpha\hat{}g_k$ for $k \leq j$ or $\gamma \supseteq \alpha\hat{}o$, where $o \in \{h_l, s, w | l < j\}$. And hence are all strategies below and to the right of outcome g_j.

If α has not yet started an attack, then it continues from the Initialization step.

If α has started an attack, and this change injures the equation that α is trying to preserve, then α will choose a new witness and start from Initialization. If the equation is not injured, then α will continue to restrain A in the hope that no later change in K will ever destroy its work.

The Construction: At each stage s of the construction we build inductively a string $\delta_s \in D(T)$ of length s, by visiting nodes from the tree and acting according to their corresponding strategies.

$\delta_s(0) = \emptyset$.

Let $\delta_s \upharpoonright n = \alpha$.

1. $T(\alpha) = (P_i, \Gamma)$ – on active stages we perform the actions as stated in the main module. $\delta_s(n+1) = l$ at nonexpansionary stages. At expansionary stages $\delta_s(n+1) = e$ and $R_{\alpha\hat{}\delta_s(n+1)} = R_\alpha$.

   At nonactive stages no actions are performed. The strategy will have the same outcome as it did on the previous active stage.
2. $T(\alpha) = (P_i, \Lambda)$ – on active stages we perform the actions as stated in the main module. $\delta(n+1) = s$ and $R_{\alpha\hat{}\delta_s(n+1)} = R_\alpha$.

   At nonactive stages no actions are performed, $\delta(n+1) = s$.
3. $T(\alpha) = (N_i, S_0, \ldots, S_{i-1})$ –

   Let $Z_j = U$ if $S_j = \Gamma$ and $Z_j = \overline{W}$ if $S_j = \Lambda$.

   – **Initialization**

   On active stages:

   Each strategy $S_j \neq FM_j$ picks a threshold if it is not already defined. The different thresholds must be in the following order:

$$L < d_{i-1} < d_{i-2} < \cdots < d_0 < R$$

   Strategy S_j picks its threshold so that it is bigger than any threshold it has picked before and such that its marker has not yet been defined.

   After all thresholds have been chosen, the strategy picks a witness x, bigger than any witness used until now and such that $d_0 < x$ and waits until $l(E, \Theta_j^{U_j, \overline{W}_j}) > x$ for all $j < i$. $\delta(n+1) = w$, working below $(R = R_\alpha)$.

   On the first stage on which $l(E, \Theta_j^{U_j, \overline{W}_j}) > x$ for all $j < i$, extract all markers for all axioms old and new for all thresholds d_j, cancel all markers $z_j(n)$ for $n \geq d_j$ and let $z_j(d_j) > \theta_j(x)$.

For every element $y \leq x$, $y \in E$ enumerate in the list $Axioms_j$ the current valid axiom from Θ_j, that has been valid the longest.

Go to honestification at the next stage. Notice that this guarantees that any axiom $\langle n, Z_n, A_n \rangle$ enumerated in S_j for an element $n \geq d_j$, $n \in \overline{K}$ will have the property that for any $y \leq x$, $x \in E$ with axiom $\langle y, Z_y, V_y \rangle \in Axioms_j$, we will have that $Z_y \subset Z_n$.
$\delta(n+1) = w$, working below $(R = R_\alpha)$.

– **Honestification**

On active stages:
Scan all strategies from the list $S_0 \ldots S_{i-1}$ in turn $(j = 0, 1, \ldots i-1)$. Perform $Honestification_j$ from the main module for each $S_j \neq FM_j$. If the outcome of $Honestification_j$ is w go on to the next strategy. If it is h, then extract all markers old and new $s_k(d_k)$ for $k > j$ from A and empty their corresponding lists $Old_k(n)$ for elements $n \geq d_k$.

The outcome is $\delta(n+1) = h_j$ working below $(R = \min(R_\alpha, s_j(d_j)))$. Start from $Honestification$ at the next stage.

– **Waiting**

If all outcomes of all $Honestification_j$-modules are w, i.e all enumeration operators are honest, then wait for $x \in \Psi_i^A$ with $use(\Psi, A, x) < R_\alpha$. $f(n+1) = w$, working within below $R = R_\alpha$). Return to $Honestification$ at the next stage.

– **Attack**

(a) Let β be the biggest node such that $\alpha \supseteq \beta \hat{} g$. If there is such a node β, then wait for a β-nonactive stage.

(b) If $x \in \Psi^A$, $use(\Psi, A, x) < R_\alpha$ and all operators are honest, then extract x from E and restrain A on $use(\Psi, A, x)$. This starts a nonactive stage for the strategies below the most recently visited outcome g_j (if none has been visited until now, then g_0).

(c) Result –
Wait until the length of agreement has returned for all strategies and they have been visited at an expansionary stage s with $ap_s^{U_j, W_j} > u(d_j)$.

Scan all strategies $S_0, \ldots S_{i-1}$ in turn, starting with S_0 and perform the corresponding $Result_j$ from the main module on each.

- If the attack was successful for S_j, continue scan with S_{j+1}. If all the strategies are scanned, then $\delta(n+1) = f$, working below $(R = R_\alpha)$ go to Result at the next stage.
- If the attack was unsuccessful for S_j, hence $S_j = \Gamma_j$, then $\gamma_j(d_j)$ has been extracted from A during $Result_j$ and the corresponding markers have been moved. In addition cancel the thresholds d_k, for $k < j$, cancel all markers $s_l(d_l)$ old and new for $l > j$ and extract them from A, emptying the corresponding lists Old_l. Cancel the witness. Start from Initialization at the next stage. $\delta(n+1) = g_j$, working below $(R = \min(x, R_\alpha))$.

Proof. The true path f is defined to be the leftmost path on the tree that is visited infinitely many times. Such a path exists, because the tree is finitely branching. We prove that the strategies along the true path satisfy their requirements.

Lemma 1. *For every n there is a stage s_n such that $f \upharpoonright n$ does not get initialized after stage s_n.*

Lemma 2. *1. Let $\alpha \subset f$ be the biggest (P_j, Γ)-strategy and assume U_j is Δ_2. If $\gamma(n)$ moves off to infinity, then the following condition holds: If $\Theta_j^{U,\overline{W}} = E$ and then there is an outcome g_j along the true path.*
2. Let $\alpha \subset f$ be the biggest P_j-strategy. It builds a function M. If $m(n)$ moves off to infinity then $\Theta_j^{U,\overline{W}} \neq E$.

Corollary 1. *Every P_i-requirement is satisfied.*

Proof. If $\Theta_i^{U_i,\overline{W}_i} \neq E$ or U_i is properly Σ_2, then the requirement is trivially satisfied. Otherwise let $\alpha \subset f$ be the biggest (P_i, M) strategy along the true path. The properties of the approximation guarantee that there are infinitely many expansionary stages. According to the previous lemma all markers m used to build the operator M are bounded, hence for each element n, there are finitely many axioms in M.

Now it is easy to prove via induction on n that $\overline{K}(n) = M^{Z,A}(n)$.

Lemma 3. *Let $\alpha \subset f$ be an N_i requirement along the true path. And let s be a stage after which α is not initialized. Then*

1. *None of the nodes to the right or to the left of α extract elements from A that are less than R_α after stage s.*
2. *None of the N_j-nodes above α extract elements from A that are less than R_α after stage s.*
3. *Suppose $\beta \subset \alpha$ is a P_j node such that there is another P_j node β', with $\beta \subset \beta' \subset \alpha$. Then β does not extract elements from A that are less than R_α after stage s.*

Hence the strategies that can injure a restraint imposed by an N_i-strategy α along the true path are the active P_j-strategies at α and α itself.

Lemma 4. *Every N_i-requirement is satisfied.*

Proof. Let α be the last N_i requirement along the true path. We will prove that it satisfies N_i.

α has true outcome w or f, or else there will be a successive copy of N_i along the true path.

In the first case, α waits forever for $\Psi^A(x) = 0$. Hence $\Psi^A(x) \neq E(x)$.

Let the true outcome be f. And let $s > s_{\alpha^\frown f}$. After stage s, $\overline{K} \upharpoonright d_1$ will not change anymore. Hence the active P_j-strategies can only extract markers of elements $n > d_j$.

If $S_j = \Gamma_j$ and the restraint is injured, then α would have outcome to the left, because this would mean that the attack for x is unsuccessful. Suppose $S_j = \Lambda_j$. Then α timed its attack, with some β, such that $\beta^\frown g_j \subset \alpha$. Note that the entries in both lists $Axioms_j$ at α and β are the same. Hence an unsuccessful β-attack would mean that α gets the right $\overline{W}$-permission.

References

1. M. M. Arslanov, *Structural properties of the degrees below* $\mathbf{0'}$, Dokl. Akad. Nauk. SSSR **283** (1985), 270–273.
2. S. B. Cooper, *On a theorem of C. E. M. Yates*, (handwritten notes), 1974.
3. S. B. Cooper, *The strong anti-cupping property for recursively enumerable degrees*, J. Symbolic Logic **54** (1989), 527–539.
4. S. B. Cooper, *Enumeration reducibility, nondeterministic computations and relative computability of partial functions*, in *Recursion Theory Week, Proceedings Oberwolfach 1989* (K. Ambos-Spies, G. H. Müller, G. E. Sacks, eds.), Lecture Notes in Mathematics no. 1432, Springer-Verlag, Berlin, Heidelberg, New York, 1990, pp. 57–110.
5. S. B. Cooper, *Computability Theory*, Chapman & Hall/CRC Mathematics, Boca Raton, FL, New York, London, 2004.
6. S. B. Cooper, A. Sorbi, A. Li and Y. Yang, *Bounding and nonbounding minimal pairs in the enumeration degrees*, J. Symbolic Logic **70** (2005), 741–766.
7. R. G. Downey, Δ^0_2 *degrees and transfer theorems*, Illinois J. Math. **31** (1987), 419–427.
8. L. Harrington, *Understanding Lachlan's monster paper* (handwritten notes), 1980.
9. A. H. Lachlan, *A recursively enumerable degree which will not split over all lesser ones*, Ann. Math. Logic **9** (1975), 307–365.
10. A. H. Lachlan and R. A. Shore, *The n-rea enumeration degrees are dense*, Arch. Math. Logic **31** (1992), 277–285.
11. D. Miller, *High recursively enumerable degrees and the anti-cupping property*, in *Logic Year 1979–80: University of Connecticut* (M. Lerman, J. H. Schmerl, R. I. Soare, eds.), Lecture Notes in Mathematics No. 859, Springer-Verlag, Berlin, Heidelberg, Tokyo, New York, 1981.
12. P. G. Odifreddi, *Classical Recursion Theory, Volume II*, North-Holland/Elsevier, Amsterdam, Lausanne, New York, Oxford, Shannon, Singapore, Tokyo, 1999.
13. D. B. Posner and R. W. Robinson, *Degrees joining to* $\mathbf{0'}$, J. Symbolic Logic **46** (1981), 714–722.
14. G. E. Sacks, *On the degrees less than* $\mathbf{0'}$, Ann. of Math. **77** (2) (1963), 211–231.
15. T. A. Slaman and J. R. Steel, *Complementation in the Turing degrees*, J. Symbolic Logic **54** (1989), 160–176.
16. R. I. Soare, *Recursively enumerable sets and degrees*, Springer-Verlag, Berlin, Heidelberg, London, New York, Paris, Tokyo, 1987.

There is an Sw-Cuppable Strongly c.e. Real*

Yun Fan

[1] Department of Mathematics, Nanjing University, China
[2] Department of Mathematics, Southeast University, China

Abstract. The strong weak truth table (sw) reducibility was suggested by Downey, Hirschfeldt and LaForte as a measure of relative randomness. In this paper, in order to discuss the structure of sw-degrees further, we introduce the definition of sw-cuppable for c.e. reals. For c.e reals, it is natural to conclude that there exist sw-cuppable c.e. reals. The main result of this paper is that there exists an sw-cuppable strongly c.e. real.

1 Introduction

In the very long and widely circulated manuscript [3] (a fragment of which appeared in [4]), Solovay carefully investigated relationships between Martin-Löf-Chaitin prefix-free complexity, Kolmogorov complexity, and properties of random languages and reals. Solovay discovered that several important properties of Ω are shared by another class of reals he called Ω-like, whose definition is model-dependent. To define this class, Solovay reducibility was introduced between c.e. reals.

Although Solovay reducibility is a powerful tool in studying relative randomness, it was not sufficient, especially as far as non-c.e. reals are concerned. Motivated by certain shortcomings of Solovay reducibility, Downey, Hischfeldt, and Laforte in [1] introduced a strengthening of the weak truth table reducibility as a measure of relative complexity, which is called *strong weak truth table reducibility* or *sw-reducibility* for short.

We work with reals between 0 and 1, identifying a real with its binary expansion, and hence with the set of natural numbers whose characteristic function is the same as that expansion. Our main concern will be *computably enumerable reals* and *strongly computably enumerable reals*, whose definitions are given as follows.

Definition 1.1 (Soare [2]). *A set A is nearly computably enumerable if there is a computable approximation $\{A_s\}_{s\in w}$ such that $A(x) = \lim_s A_s(x)$ for all x and $A_s(x) > A_{s+1}(x) \Rightarrow \exists y < x(A_s(y) < A_{s+1}(y))$.*

Definition 1.2. *A real α is computably enumerable (c.e) if $\alpha = 0.\chi_A$ where A is a nearly c.e. set. A real α is strongly computably enumerable (strongly c.e.) if $\alpha = 0.\chi_A$ where A is a c.e. set.*

* This work is supported by DFG (446 CHV 113/240/0-1) and NSFC (10420130638).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 212–221, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Sw-reducibility is more explicitly derived from the idea of initial segment complexity. It is quite natural since it occurs in many proofs in classical computation theory: it follows when we apply simple permitting for the construction of a set "below" a given one.

Definition 1.3 (Downey, Hischfeldt, and Laforte in [1]). *Let $A, B \subseteq N$. We say that B is sw-reducible to A, written $B \leq_{sw} A$, if there is a turning functional Γ and a constant c such that $B = \Gamma^A$ and the use of this computation on any argument x is bounded by $x + c$. For reals $\alpha = 0.\chi_A$ and $\beta = 0.\chi_B$, we say that β is sw-reducible to α, and write $\beta \leq_{sw} \alpha$ if $B \leq_{sw} A$.*

As *sw*-reducibility is reflexive and transitive, we can define the *sw-degree* $deg_{sw}(\alpha)$ of a real α to be its *sw*-equivalence class. In this paper it further intends to analyze the structure of *sw*-degrees.

Since Yu and Ding [5] showed that there is no *sw*-complete c.e. reals, the definition of *cuppable* under *sw*-reducibility will be a little different from the definition of *cuppable* under Turing reducibility. It is defined as follows:

Definition 1.4. *A c.e.real α is sw-cuppable if there is a c.e. real β such that there is no c.e. real above both of them respect to sw-reducibility.*

From the definition of *sw*-cuppable, every computable real is non-*sw*-cuppable. So there exist non-*sw*-cuppable (strongly) c.e. reals. In [5], Yu and Ding's result can be restated as following.

Theorem 1.5 (Yu and Ding [5]). *There exist sw-cuppable c.e. reals.*

To analyze the property of *sw*-cuppable, it is natural to ask for whether there is an *sw*-cuppable strongly c.e. real. In this paper, we answer it positively as follows:

Theorem 1.6. *There is an sw-cuppable strongly c.e. real.*

In order to analyze further property of *sw-cuppable*, the follows question is waiting for answer.

Question 1.7. Is there a non-computable strongly c.e. real which is non-*sw*-cuppable?

The notations and terminology are quite standard, which all contained in Soare [2].

2 A Corollary

There is an interesting corollary deduced from Theorem 1.5 which was proved by Barmpalias and Lewis [6].

Corollary 2.1 (Barmpalias and Lewis [6]). *There is a non-random c.e. real which cannot be sw-computed by any c.e. random real.*

Proof. By Theorem 1.5, there is a strongly c.e. real α and a c.e. real β so that no c.e. real can sw-compute both of them. Then β must not be random since Downey, Hirschfeldt and LaForte proved (see Theorem 13.7.1 in [1]) that every c.e. random real sw-computes all strongly c.e. reals. Furthermore, no c.e. random real can sw-compute β by the same reason above.

3 Procedure $P(n, k)$

From now on all reals will be c.e reals except explicitly stated otherwise. In this section, we do some preparation for proof of theorem 1.6.

We want to construct a strongly c.e. real α_0 and a c.e. real α_1 in stages such that any β which tries to sw-compute α_0 and α_1 simultaneously fails to do that. In other words, β is unable to cope with α_0 and α_1 sw-coding needed.

The requirements are:

$$R_{\Phi,\Psi,\beta} : \alpha_0 \neq \Phi^\beta \wedge \alpha_1 \neq \Psi^\beta,$$

where Φ and Ψ run over all partial computable sw-functionals and β over all c.e. reals in the unit interval.

Considering a single requirement, we picture β having to cope with the instructions : every change at x-th digit from α_i should be sw-coded in β. If the use of Φ and Ψ is the identity plus a constant c, the instruction will be 'change a digit of β below $x + c$'. But at this moment we assume $c = 0$ for simplicity.

As to express our idea clearly, some descriptions and definitions in Barmpalias and Lewis [6] will be introduced .

Definition 3.1. *The positions on the right of the decimal point in the binary expansion are numbered as* 1, 2, 3,... *from left to right. The first position on the left of the decimal point is* 0. *We say interval* $[h, l]$ *means positions from* h *to* l.*'before position* x*'means positions 'at* x *or higher ',i.e.* $\leq x$.

We can think of the requirement above as a game between players A and B, where A controls α_i $(i = 0, 1)$ and B controls β. A winning strategy for A means that the instructions from an enumeration of α_i make β fails to sw-compute them. A winning strategy for B means that β satisfy all instructions from α_i. Once having a winning strategy for A, we can apply the same idea to deal with all requirements.

The following lemma which was proved by Yu and Ding [5], and then made be clearer by Barmpalias and Lewis [6], will simplify things in designing a winning strategy for A.

Lemma 3.2. *In the game described above between* A *and* B, *the best strategy for* B *is to increase* β *by the least amount needed to satisfy* A*'s instruction(i.e. a change of a digit in* β *below a certain level) when the request is put forward by* A. *In other words, any other different strategy for* B *produces* β' *which at every stage* s $\beta_s \leq \beta'_s$.

The observation above shows that player A forces β to be larger and larger, which provide us a winning strategy for A. Once the forcing from α_i is strong enough, player B has to give up it's wining strategy. Otherwise even β from the *best B-strategy* will exit the unit interval, which contradicts to 'β is in $[0,1]$.'

Assuming the *best B-strategy* of Lemma 3.2, every A-strategy corresponds to a single B-strategy and so a single β.

Since α_0 is a strongly c.e. real, any digit of α_0 cannot return back to 0 if it changes from 0 to 1 during the game. Yu-Ding's procedure in [5] guarantees the whole proof. Compared with the proof in [5], it is impossible to perform their procedure to produce the requests for sw-coding from a strongly c.e. α_0.

The natural solution of our proof for theorem 1.6 is to design a finer procedure which can utilize the requests from α_i more effectively and takes any strong forcing to β. We say 'an n forcing procedure' means that 'player A forces $\beta = n$ after running this procedure'.

Definition 3.3. *For arbitrary binary natural number n, k. We say that there is a $P(n,k)$ procedure when*

- *Player A forces β to be equal to n if A performs this procedure on interval $[1, l_{n,k}]$ of α_i*
- *For all positions x between the decimal point and k, $\alpha_0(x) = 0$*

Remark 3.4. $\diamond$ More precisely, $P(n,k)$ stands for a subclass of all procedures. Every element in $P(n,k)$ satisfies orders in definition 3.3. Without confusion, we consider $P(n,k)$ as a concrete procedure.

$\diamond$ Procedure $P(n,k)$ just describes the whole program which meets with A's winning strategy when it runs at interval $[1,l]$. As a program, $P(n,k)$ can be used by player A at any interval $[j, l+j]$. But once A performs $P(n,k)$ on $[j,l+j]$, player B controls β to be '$n \times 2^{-l}$'.

$\diamond$ Without confusion, $l_{n,k}$ is written as l.

Example 3.5. For $n = 1$, it is easy to give a $P(1,k)$ procedure for any k. In figure 1, the first $k+1$ digits of α_i, β are shown in consecutive stages as the arrow indicates. In other words, A performs $P(1,k)$ on interval $[1, k+1]$ of α_i, β and one can see the action of A along with the response of B.

	$\longrightarrow$				
α_0	$00\cdots0$	$000\cdots0$	$\cdots\cdots$	$000\cdots0$	$.000\cdots1$
α_1	$10\cdots0$	$110\cdots0$		$111\cdots1$	$.111\cdots1$
β	$10\cdots0$	$110\cdots0$		$111\cdots1$	$1.000\cdots0$
	$\underbrace{\quad}_{k+1}$				

Fig. 1.

For simplicity, we substitute consecutive 0 or 1 for '$\cdots$' in reals. In all figures, the left position of decimal point (and decimal point)will be omitted if no changes are involved in them.

Observing Figure 1, till last stage player A only changes α_1 from higher position 1 to lower position $k+1$ continuously. At the last stage, the action which A changes α_0 at position $k+1$ forces $\beta = 1$.

More precisely, all positions between the decimal point and k of α_0 is still not used for A's winning.

Therefore, there is a $P(1,k)$ procedure.

For case $n > 1$, the construction of $P(n,k)$ will be more sophisticated. Before giving the general method for producing $P(n,k)$, we discuss two simple cases for $n = 10, k = 0$ and $n = 10, k = 1$ as concrete examples, which help to explain the idea of dealing with general cases.

Example 3.6. Case $n = 10$*,* $k = 0$: A $P(10,0)$ procedure is shown at the interval $[1,3]$ of α_i in consecutive stages by the following figure and one can see the action of A along with the response of B.

	→						
α_0	000	000	001	.001	.101	.101	.111
α_1	010	011	011	.100	.100	.110	.110
β	010	011	100	1.000	1.100	1.110	10.000
		$P(1,1)$ *at* $[2,3]$				$P(1,0)$ *at position* 2	

Fig. 2.

Looking at stage 1-3, player A forces $\beta = 0.1$ by remark 3.4 when running $P(1,1)$ in interval $[2,3]$. Fortunately running $P(1,1)$ assures the second position of α_0 still be empty .

In stage 4, A takes a 0.1 forcing to β by using the first position of α_1 meanwhile it makes next two lower positions clear and wait again.

In stage 5, by using the first position of α_0, it takes a 0.1 forcing to β.

In stage 6, 7, since both the second positions of α_i can be used, run $P(1,0)$ on it which makes a 0.1 forcing to β again. Hence, $\beta = 0.1 + 0.1 + 0.1 + 0.1 = 10$.

Therefore, this is a $P(10,0)$ procedure.

Remark 3.7. This example indicates a general method how to get a stronger forcing $P(n+1,0)$ procedure to β from two weaker forcing $P(n,k)$ procedures when player A carries out this procedure on interval $[1,l]$. We explain its idea by case 10.

The existence of $P(1,k)$ provides a good foundation for constructing a stronger forcing procedure $P(10,0)$. Whenever A use the forcing from the first positions of α_i, it always cause β to plus 1 compared with before using. In order to force $\beta = 10$, the increasing amount of β from the other forcing of α_i should be equal to 1. Considering that higher position's change can bring lower position clear for c.e. real, the interval $[2,l]$ of α_1 can be used as often as two times. By remark 3.4, the actions which any procedure $P(1,k)$ carried out on $[2,l]$ can only force

β to plus 0.1. So the solution to get $P(10,0)$ is to find an interval which permits A to apply $P(1,k)$ to $[2,l]$ two times often, written as $P(1,k_1)$ and $P(1,k_2)$.

Since α_0 is strongly c.e real, $k_1 \neq k_2$. Suppose that $k_1 < k_2$. Run $P(1,k_2)$ from the second position of $[1,l]$, and k_2 stands for the length of the unused interval from position 2 of α_0. After using clear forcing from the first position of α_1, both intervals $[2,k_2+1]$ of α_i can be used again. If the length $1+k_1$ of the interval needed by carrying out $P(1,k_1)$ do not surpass k_2, do $P(1,k_1)$ on $[2,k_2+1]$ instantly and bring a 0.1 forcing to β again. The 0.1 forcing from the first position of α_0 can be arranged at any stage. . Considering all forcing to β, $\beta = 0.1+0.1+0.1+0.1+0.1 = 10$.

$P(10,0)$ procedure is not unique and interval $[1,l]$ only needs long enough to apply those actions mentioned above. Since the shortest interval which permits 1 forcing to β is $P(1,0)$, choose $k_1 = 1$ and then $k_2 \geqslant k_1 + 1 = 2$. Let $k_2 = 2$, and set $l = 3$, we get the $P(10,0)$ procedure in above example.

The cases for $n = 11, 100, \cdots$ can follow in similar way.

During the construction of a stronger forcing procedure, it is most important that the first weaker forcing procedure could provide a sufficient unused interval of α_0 to wait for another weaker forcing procedure.

Considering case $n = 11$, it asks for an $n = 10$ weaker forcing procedure which can provide such a sufficient interval to use $P(10,0)$. Since three positions are needed for applying $P(10,0)$, looking for a $P(10,11)$ is our task.

Example 3.8. Case $n = 10$*,* $k = 1$: A $P(10,1)$ procedure is shown at the interval $[1,5]$ of α_i in consecutive stages by the following figure and one can see the action of A along with the response of B.

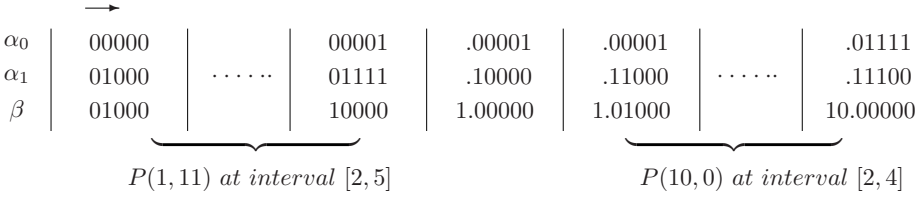


Fig. 3.

Looking at the stages for applying $P(1,3)$ to interval $[2,5]$. A forces $\beta = 0.1$ by remark 3.4. Fortunately running $P(1,3)$ assures interval $[2,4]$ of α_0 still be empty.

In the following stage, A takes a 0.1 forcing to β by using the first position of α_1 meanwhile it makes next four positions clear and wait again.

Looking at next stages for applying $P(10,0)$, since both intervals $[2,4]$ of α_i can be used, run $P(10,0)$ on it which makes a 1.0 forcing to β again by remark 3.4. Hence, $\beta = 0.1+0.1+1.0 = 10$.

More precisely, first position of α_0 keeps '0' during the whole program.

Therefore, this is a $P(10,1)$ procedure.

Remark 3.9. This example indicates a general method how to get a $P(n, k+1)$ procedure from a $P(n, k)$ procedure and some weaker forcing procedure $P(n - 0.1, m)$ when player A carries out this procedure on interval $[1, l]$. We explain its idea by case $n = 10$.

The first position of α_0 can not be used forever. Whenever A use the forcing from the first position of α_1, it always cause β to plus 0.1. In order to force $\beta = 10$, the increasing amount of β from the other forcing of α_i should be equal to 1.1. Considering that higher position's change can bring lower position clear for c.e. real, the interval $[2, l]$ of α_1 can be used as often as two times. By remark 3.4, the actions which some procedure $P(1, m)$ carry out on $[2, l]$ can only force β to plus 0.1. So the solution to get $P(10, k+1)$ is to find an interval which permits A to apply $P(2, k)$ to $[2, l]$.

Run $P(1, m)$ from the second position of $[1, l]$, and m stands for the length of the unused interval from position 2 of α_0. After using clear forcing from the first position of α_1, both intervals $[2, m+1]$ of α_i can be used again. If the length of the interval needed by carrying out $P(2, k)$ do not surpass m, do $P(2, k)$ on $[2, m+1]$ instantly which brings a 1.0 forcing to β again and at the same time keeps interval $[2, k+1]$ empty. During the whole program, it is not necessary to utilize the first position of α_0. Therefore, for all positions x between decimal point and $k+1$, $\alpha_0(x) = 0$. Considering all forcing to β, $\beta = 0.1 + 0.1 + 1.0 = 10$.

$P(10, 0)$ procedure is not unique and interval $[1, l]$ only needs long enough to apply those actions mentioned above. Since the length of the shortest interval of $P(1, 0)$ given by example 3.6 is 3, $m \geqslant 3$. Let $m = 3$. Since the length of interval $P(10, 1)$ procedure mentioned in example 3.5 is 5, let $l = 5$, and we get the $P(10, 0)$ procedure in above example.

By iteration, $P(10, 1)$ procedures $P(10, 10)$ and $P(10, 10)$ procedures $P(10, 11)$.

After generalizing those methods where example 3.6 and example 3.8 indicates, we get a general conclusion in next lemma.

Lemma 3.10. *There exists a procedure $P(n, k)$ for arbitrary number (n, k).*

Proof. For $(1, k)$: $P(1, k)$ is given by example 3.6.

For $(n+1, 0)$ $(n > 1)$: Suppose that $P(n', k)$ have been given for any $n' < n$, set $l_{n+1,0} = l_{n,l_{n,0}} + 1$. Then we describe a $P(n+1, 0)$ procedure in consecutive stages when A runs it in interval $[1, l_{n+1,0}]$ of α_i as follows:

- ◇ *run* $P(n, l_{n,0})$ *in interval* $[2, l_{n+1,0}]$
- ◇ *change* α_0 *at position* 1
- ◇ *change* α_1 *at position* 1 *and clear next lower positions*
- ◇ *run* $P(n, 0)$ *in interval* $[2, l_{n,0} + 1]$

Looking at the stages for applying $P(n, l_{n,0})$, player A produces a $n/2$ forcing to β by remark 3.4. Fortunately running $P(n, l_{n,0})$ assures interval $[2, l_{n,0} + 1]$ of α_0 still be empty.

In the following two stages, by using both first positions of α_i, it takes a $0.1 + 0.1$ forcing to β.

Looking at next stages for applying $P(n,0)$. Since both intervals $[2, l_{n,0}+1]$ of α_i can be used, run $P(n,0)$ on it which takes an $n/2$ forcing to β again by remark 3.4.

Hence, $\beta = n/2 + 0.1 + 0.1 + n/2 = n$.

Therefore, this is a $P(n+1,0)$ procedure.

For $(n > 1, k+1)$: Suppose that $P(n',k)$ have been given for any $n' < n$ and $n' = n, k' \leq k$, set $l_{n,k+1} = l_{n-1,l_{n,k}} + 1$. Then we describe a $P(n,k+1)$ procedure in consecutive stages when A runs it in interval $[1, l_{n,k+1}]$ of α_i as follows:

$\diamond$ *run* $P(n-1, l_{n,k})$ *in* $[2, l_{n,k+1}]$
$\diamond$ *change* α_1 *at position* 1 *and clear next lower positions.*
$\diamond$ *run* $P(n,k)$ *in* $[2, l_{n,k}+1]$

Looking at the stages for applying $P(n-1, l_{n,k})$, A produces a $(n-1)/2$ forcing to β by remark 3.4. Fortunately running $P(n-1, l_{n,k})$ assures interval $[2, l_{n,k}+1]$ of α_0 still be empty.

In the following stage, by using the first position of α_1, it takes a 0.1 forcing to β.

Looking at next stages for applying $P(n,k)$. Since both intervals $[2, l_{n,k}+1]$ of α_i can be used, run $P(n,k)$ on it which takes an $n/2$ forcing to β again by remark 3.4.

Hence, $\beta = (n-1)/2 + 0.1 + n/2 = n$.

More precisely, all positions between decimal point and $k+1$ α_0 keeps '0' during the whole program.

Therefore, this is a $P(n,k+1)$ procedure.

We make inductions on n and k respectively in above lemma. The only purpose of induction on k is to assure the assumption when making induction on n. During the construction of procedure $P(n,k)$ for any n,k, the finial result we want is to find a procedure which could take any stronger forcing to β when running it in some interval of α_i. In other words, a $P(n,0)$ procedure for any n is what we really want.

Recall that so far we assumed that the use of the *sw*-functional which computes α_i from β is the identity. In general it will be $x+c$ on argument x and so for arbitrary c we consider the modification $P_c(n,0)$ of $P(n,0)$ which produces the same n forcing to β.

Corollary 3.11. *Suppose that* α_i *force* β *with the use function* $x+c$. *There is a procedure* $P_c(n,0)$ *which produces an* n *forcing to* β *when it runs in interval* $[1, l_{n,k}]$.

Proof. *Set* $P_c(n,0) = P(2^c \times n, 0)$.

4 Proof of Theorem 1.6

Assume an effective list of all requirements

$$R_{1,1}, R_{1,2}, R_{2,1}, \cdots$$

based on an effective list (Φ, Ψ, β) of computable sw-functional and c.e. reals in $(0, 1)$, where

$$R_{e,j} : \alpha_0 \neq \Phi_j^{\beta_e} \wedge \alpha_1 \neq \Psi_j^{\beta_e}$$

Now we adopt a priority list of all $R_{e,j}$ based on the following priority relation:

$$R_{e_1,j_1} < R_{e_2,j_2} \Longleftrightarrow \langle e_1, j_1\rangle < \langle e_2, j_2\rangle$$

where $\langle , \rangle$ is a standard pairing function.

To each $R_{e,j}$, we assign an interval I_j^e where α_i can apply some $P_c(n, 0)$ procedure, here c is the constant in the use of sw-functionals Φ_j and Ψ_j. Suppose that the intervals for higher priority requirements have been defined and that x_0 is the least number larger than all numbers in these intervals. The winning strategy for $R_{e,j}$ will be provided by running $P_c(2^{x_0}, 0)$ in I_j^e.

Lemma 4.1. *Any β which follows the instructions of a $P_c(2^{x_0}, 0)$ procedure in interval I_j^e has to be $\geqslant 1$.*

Proof. Assume that β comes from the best strategy for B. By corollary 3.11 and remark 3.4, running $P_c(2^{x_0}, 0)$ in interval I_j^e produces a $2^{x_0} \times 2^{-x_0} = 1$ forcing to β. By lemma 3.2, it is proved.

Construction: We give the construction in stages.
At stage $s = 0$. Set $\alpha_{0,0} = \alpha_{1,0} = 0$.
At stage s. Assign an interval for requirement $R_{e,j}$ when $\langle e, j\rangle = s$. $R_{e,j}$ requires attention at stage s if $\alpha_0 = \Phi_j^{\beta_e} \wedge \alpha_1 = \Psi_j^{\beta_e}$ currently holds on all arguments up to the largest number in interval I_j^e. Choose the least $\langle e, j\rangle \leq s$ that $R_{e,j}$ requires attention. Then perform the next step of its winning strategy and end stage s.We say that $R_{e,i}$ *receives attention* at stage s.
If no $R_{e,j}$ requires attention, end stage s instantly.

Verification:
Considering how to assign interval I_j^e, we work in a fixed interval for each requirement. One thing to note is that there is no interaction amongst all winning strategies. It means the lack of induction in the verification.

Fixing e, j we will show that $R_{e,j}$ is satisfied. Once α_i is successfully being coded into β, it follows the instructions in a $P_c(2^{x_0}, 0)$ procedure. According to lemma 3.12, if α_i uses up all positions of I_j^e, then $\beta \geqslant 1$, a contradiction.

Through a tricky procedure, we complete proof of theorem 1.6.

References

1. Rodney G. Downey, Denis Hirschfeldt. Randomness and reducibility. *Springer-Verlag Monographs in Computer Science. Preparation.*
2. Robert I. Soare. *Recursively Enumerable Sets and Degrees.* Springer-Verlag, Berlin, 1987.
3. R. M. Solovay. Draft of a paper (or series of papers) on chaitin's work... *done for the most part during the period of Sept.-Dec. 1974 (May 1975), unpublished manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 215 pages.*
4. R. M. Solovay. On random r.e. sets, in A. Arruda, N.Da Costa, and R.Chuaqui(eds), Non-Classical Logics, Model Theory, and Computability. *Proceedings of the Third Latin-American Symposium on Mathematical Logic, Campinas, July 11-17, 1976, vol. 89 of studies in Logic and the Foundations of Mathematics (North-Holland, Amsterdam, 1977) 283-307.*
5. Liang Yu and Decheng Ding. There is no SW-complete c.e. real. *J. Symbolic Logic* 69 (2004), no. 4, 1163–1170.
6. Barmpalias, George; Lewis, Andrew E. M. A c.e. real that cannot be SW-computed by any Ω number. *Notre Dame J. Formal Logic* 47 (2006), no. 2, 197–209

On Computation Complexity of the Concurrently Enabled Transition Set Problem

Li Pan, Weidong Zhao, Zhicheng Wang, Gang Wei, and Shumei Wang

Tongji University, Shanghai 200092, China
CAD Research Center
panli12345@sohu.com

Abstract. In this paper, we propose a new decision problem, called the concurrently enabled transition set problem, which is proved to be NP-complete by reduction from the independent set problem. Then, we present a polynomial-time algorithm for the maximal concurrently enabled transition set problem, and prove that some special subproblems are in P by the proposed algorithm.

1 Introduction

Petri nets [1,2] are a powerful formal model for describing and studying concurrent and distributed systems. In these systems, transitions (actions, events, etc.) usually depend on a limited set of places (conditions, resources, etc.), which could be called the local environment. Petri nets model transitions by the change of their environment, and transitions having disjoint locality can occur concurrently.

In Petri nets, two types of execution strategies for the concurrency of transitions are usually adopted: interleaving execution and step execution. The interleaving execution is interpreted as firing concurrent transitions one by one in any sequence, while the step execution as firing a concurrently enabled transition set by a step. It is shown that step execution can efficiently reduce the state space explosion problem due to the execution of concurrency by interleaving.

The execution of a step first needs to determine concurrently enabled transition sets of Petri nets. However, the number of concurrently enabled transition sets, in general, is exponential magnitude of the number of enabled transitions. Therefore, whether the decision problem w.r.t concurrently enabled transition sets is NP-complete will directly determine the computation complexity of the step execution.

Although the concept of concurrently enabled transition set or step has been introduced or used in papers [3,4,5,6,7,8], none of these studies were done on the computation complexity as a decision problem. In [3], Jeffrey et al. proposed the concept of concurrently enabled transition set for describing a HCLGN-Based parallel execution model. In [4], a step transition systems generated by Petri nets was characterized in terms of concurrent step by Mukund. In [5], the definition and computation of concurrency-degree was presented by Jucan and Vidraşcu.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 222–233, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In recent years, the concept of the transition step has been utilized to reduce state space explosion in verification methods by Vernadat et al. [6,7]. And in [8], concurrently enabled transition sets were adopted to analyze the reachability of time Petri nets by Pan et al.

On the other hand, the research on NP-complete problems for Petri nets focuses on several aspects as follow. (i) The deadlock problem, for safe free-choice nets, is NP-complete [9]. (ii) The reachability problem, for live and safe free-choice nets, acyclic nets, conflict-free nets, and normal and sinkless nets, is NP-complete [10,11,12,13]. (iii) The liveness problem, for free-choice nets, is NP-complete [9]. (iv) The legal firing sequence problem, for general Petri nets and some free-choice nets, is NP-complete [14], [15]. (v) The synthesis problem, for elementary net systems, is NP-complete [16]. In addition, the decidability and complexity about the verification problem in Petri nets are also discussed in [17].

This paper proposes for Petri nets a new decision problem, called the concurrently enabled transition set problem, and this problem is proved to be NP-complete. Furthermore, a polynomial-time algorithm for the maximal concurrently enabled transition set problem is presented, and some subproblems solvable in polynomial time are also analyzed.

The rest of the paper is organized as follows. Section 2 defines the Petri net and the concurrently enabled transition set. Section 3 presents concurrently enabled transition set problem and discusses its complexity. Section 4 analyzes some special subproblems solvable in polynomial time. Finally, Section 5 concludes the paper.

2 Basic Definitions

2.1 Petri Nets

Definition 1. *A Petri Net (place/transition net) is a 4-tuple = (P, T, Pre, Post), where:*

(1) $P = \{p_1, p_2, \ldots, p_m\}$ *is a finite nonempty set of places;*
(2) $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite nonempty set of transitions;*
(3) $Pre \in N^{P \times T}$ *is the forward incidence matrix;*
(4) $Post \in N^{P \times T}$ *is the backward incidence matrix.*

$Pre(p,t) = i$ $(i > 0)$ if and only if there is an arc with weight i from place p to transition t; $Pre(p,t) = 0$ if and only if there is no arc from place p to transition t. $Pre(t) \in N^P$ denotes the multi-set of input places of transition t, corresponding to the vector of column t in the forward incidence matrix. ${}^\bullet t = \{p \mid Pre(p,t) > 0\}$ denotes the set of input places of transition t. $p^\bullet = \{t \mid Pre(p,t) > 0\}$ denotes the set of output transitions of place p. Similarly, $Post(p,t)$, $Post(t)$, $t^\bullet$ and ${}^\bullet p$ are defined.

Note that the capacities of places are not described explicitly in the above definition, and then the default value ∞ is assigned automatically.

Definition 2. *A marking of a Petri net Σ is a multi-set $M \in N^P$. The ith component $M(p_i)$ of marking M is the number of tokens in place p_i.*

A Petri net system is a pair (Σ, M_0) where Σ is a Petri net and M_0 is an initial marking. But the definition only describes the static specification of a net system, i.e., the net structure and the initial resource distribution. So we also need to define the dynamic behaviors of a net system, which are given by the evolution of the marking that is driven by the following rules.

Definition 3 (Enabling rule). *A transition t is enabled at marking M in a Petri net Σ, abbreviated $M[t\rangle_\Sigma$, if and only if $\forall p \in P$: $Pre(p,t) \leq M(p)$.*

Let $En_\Sigma(M) = \{t | M[t\rangle_\Sigma\}$ be the set of transitions enabled at M in Σ. The notation "$[\cdot\rangle_\Sigma$" and "$En_\Sigma(\cdot)$" will be simplified to "$[\cdot\rangle$" and "$En(\cdot)$" whenever Σ is understood from context. Let $d_M(t) = min\{M(p)/Pre(p,t) \mid p \in {}^\bullet t\}$ denote the enabling degree of transition t at marking M. A transition t is multi-enabled at marking M if $d_M(t) \geq 2$.

In addition, the following marking arithmetic will be used later: $M_1 \leq M_2$ iff $\forall p \in P$: $M_1(p) \leq M_2(p)$; $M_1 + M_2$ iff $\forall p \in P$: $M_1(p) + M_2(p)$; $k{\cdot}M$ iff $\forall p \in P$: $k{\cdot}M(p)$; and if $M_2 \leq M_1$, then $M_1 - M_2$ iff $\forall p \in P$: $M_1(p) - M_2(p)$.

Definition 4 (Firing rule). *If a transition t is enabled at marking M in a Petri net Σ, then t may be fired yielding a new marking M', abbreviated $M[t\rangle_\Sigma M'$, defined by $M' = M + Post(t) - Pre(t)$.*

Below we introduce several classical subclasses of Petri net.

Definition 5. *Let $\Sigma = (P, T, Pre, Post)$ be a Petri net.*

(1) *Σ is an ordinary net, if all of its arc weights are 1's. i.e., $\forall p \in P, \forall t \in T$: $Pre(p,t) \in \{0,1\} \wedge Post(p,t) \in \{0,1\}$.*

(2) *Σ is a free-choice net, if it is an ordinary net such that $\forall p_1, p_2 \in P$: ${p_1}^\bullet \cap {p_2}^\bullet \neq \emptyset \rightarrow |{p_1}^\bullet| = |{p_2}^\bullet| = 1$.*

(3) *Σ is an extended free-choice net, if it is an ordinary net such that $\forall p_1, p_2 \in P$: ${p_1}^\bullet \cap {p_2}^\bullet \neq \emptyset \rightarrow {p_1}^\bullet = {p_2}^\bullet$.*

(4) *Σ is an asymmetric choice net, if it is an ordinary net such that $\forall p_1, p_2 \in P$: ${p_1}^\bullet \cap {p_2}^\bullet \neq \emptyset \rightarrow {p_1}^\bullet \subseteq {p_2}^\bullet \vee {p_2}^\bullet \subseteq {p_1}^\bullet$.*

2.2 Concurrently Enabled Transition Set

The behaviors of a transition exclusively depend on its locality, which is defined as the totality of its input and output objects (places and arcs) together with the transition itself. Transitions having disjoint locality can occur independently (concurrently). Because all places' capacities are defined as infinite, the concurrency of transitions does not depend on their output objects any more.

Definition 6. *Transitions t_1 and t_2 are said to be concurrently enabled at marking M, denoted by $t_1||_M t_2$, if $\forall p \in P$: $Pre(p,t_1) + Pre(p,t_2) \leq M(p)$.*

If $d_M(t) \geq 2$, then $\forall p \in P$: $2Pre(p,t) \leq M(p)$, and then $t||_M t$. Therefore, the multi-enabledness of a transition can be considered as being concurrently enabled of the transition with itself. In order to simplify the treatment of the problem discussed in this paper, we do not consider multi-enabledness of transitions. But this does not influence the essence of the problem[1].

Assumption 1. *From now on, we assume that all Petri nets have no multi-enabledness.*

Definition 7. *A transition set $C \subseteq T$ is called a concurrently enabled transition set at marking M, if $\sum_{t \in C} Pre(t) \leq M$.*

Let $ConSet_\Sigma(M)$ denote the set of all concurrently enabled transition sets at marking M in a Petri net Σ. Obviously, the empty set and singletons are concurrently enabled transition sets, i.e., $\emptyset \in ConSet_\Sigma(M)$, and $\forall t \in En_\Sigma(M)$: $t \in ConSet_\Sigma(M)$. If C is a concurrently enabled transition set, then all of transitions in C may be fired by a step.

Assume that $T' \subseteq En_\Sigma(M)$ is a set of enabled transitions. If $\forall t_1, t_2 \in T'$: $t_1||_M t_2 \wedge {}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$, then $\sum_{t \in T'} Pre(t) \leq M$, thus T' is a concurrently enabled transition set.

Definition 8. *A transition set $C \in ConSet_\Sigma(M)$ is a maximal concurrently enabled transition set, if $\neg \exists C' \in ConSet_\Sigma(M)$: $C \subset C'$.*

Let $MaximalCS_\Sigma(M)$ be the set of all maximal concurrently enabled transition set at marking M in a Petri net Σ.

Definition 9. *A transition set $C \in MaximalCS_\Sigma(M)$ is a maximum concurrently enabled transition set, if $\forall C' \in MaximalCS_\Sigma(M)$: $|C| \geq |C'|$.*

Let $MaximumCS_\Sigma(M)$ be the set of all maximum concurrently enabled transition set at marking M in a Petri net Σ. The cardinality of a maximum concurrently enabled transition set is said to be the concurrent number of (Σ, M), denoted by $\lambda_\Sigma(M)$. Consider a Petri net system (Σ, M) in Fig.1. It is not difficult to see that $\{t_1\}$, $\{t_1, t_3\}$, and $\{t_1, t_3, t_4\}$ are concurrently enabled transition sets, $\{t_2, t_4\}$ and $\{t_1, t_3, t_4\}$ are maximal concurrently enabled transition sets, $\{t_1, t_3, t_4\}$ is maximum concurrently enabled transition set, hence $\lambda_\Sigma(M) = 3$.

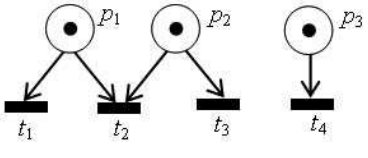


Fig. 1. Concurrently enabled transition sets

[1] The multi-enabledness of transitions just leads us to consider concurrently enabled transition multi-sets.

Contrary, in some sense, to concurrency is the notion of conflict. Formally, We have a situation of (effective) conflict if and only if the set of all transitions enabled at a given marking is not a concurrently enabled transition set.

Definition 10. *Two enabled Transitions t_1 and t_2 are said to be in effective conflict at marking M, denoted by $t_1 \uparrow_M t_2$, if $\exists p \in P$: $Pre(p, t_1) + Pre(p, t_2) > M(p)$.*

Definition 11. *Transitions t_1 and t_2 are said to be in structural conflict at marking M, if they have at least one common input place, i.e., ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$.*

It is important to remark the difference between structural conflicts and effective conflicts which depends on the marking. Two transitions are in structural conflict when they are in effective conflict, but a structural conflict does not guarantee the existence of an effective conflict.

Proposition 1. *Let (Σ, M) be a Petri net system. If one of the following conditions is satisfied*:

(1) *Σ is an extended free-choice net, or*
(2) *Σ is an ordinary net and $\forall p \in P$: $M(p) \in \{0, 1\}$, or*
(3) *$\forall t \in T, \forall p \in {}^\bullet t$: $M(p) < 2Pre(p, t)$,*

then $\forall t_1, t_2 \in En_\Sigma(M)$: $t_1 || _M t_2$ if and only if ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$.

Proof. ($\Leftarrow$). If ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$, then $\neg\exists p \in P$: $Pre(p, t_1) > 0$ and $Pre(p, t_2) > 0$. If $t_1, t_2 \in En_\Sigma(M)$, then $\forall p \in P$: $Pre(p, t_1) \leq M(p)$ and $Pre(p, t_2) \leq M(p)$. So $\forall p \in P$: $Pre(p, t_1) + Pre(p, t_2) \leq M(p)$. By Definition 6, we have $t_1 || _M t_2$.

($\Rightarrow$). (1). Assume that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, then ${}^\bullet t_1 = {}^\bullet t_2$ and $\forall p \in P$: $Pre(p, t_1) = Pre(p, t_2) = 1$ by the definition of extended free-choice nets. If $t_1 || _M t_2$, then $\forall p \in P$: $Pre(p, t_1) + Pre(p, t_2) \leq M(p)$. Thus we obtain $\forall p \in P$: $2Pre(p, t_1) \leq M(p)$, this is a contradiction to Assumption 1.

(2). Assume that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, then $\exists p' \in {}^\bullet t_1 \cap {}^\bullet t_2$: $Pre(p', t_1) = Pre(p', t_2) = 1$ by the definition of ordinary nets. If $t_1 || _M t_2$, then $\forall p \in P$: $Pre(p, t_1) + Pre(p, t_2) \leq M(p)$. So $\exists p' \in P$: $M(p') \geq Pre(p', t_1) + Pre(p', t_2) = 2$, this is a contradiction to the condition (2).

(3). Assume that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, then $\exists p' \in {}^\bullet t_1 \cap {}^\bullet t_2$: $Pre(p', t_1) > 0$ and $Pre(p', t_2) > 0$. If $t_1 || _M t_2$, then $\forall p \in P$: $Pre(p, t_1) + Pre(p, t_2) \leq M(p)$. Without loss of generality, assume that $Pre(p', t_1) \leq Pre(p', t_2)$. So we obtain $M(p') \geq 2Pre(p', t_1)$, this is a contradiction to the condition (3). □

3 Computational Complexity

3.1 Independent Set Problem

Definition 12. *Let $G = (V, E)$ be a simple, undirected graph, where V is a finite set of vertices and E is a set of edges. Two vertices v, w are adjacent if $(v, w) \in E$. A set I of vertices is independent if $(v, w) \notin E$ for all $v, w \in I$. An independent set I is a maximal independent set if $I \cup \{v\}$ is not independent for any $v \notin I$. An independent set I is a maximum independent set if $|I| \geq |I'|$ for any independent set I'.*

In computational complexity theory, a decision problem is a question in some formal system with a yes-or-no answer. Below we introduce several important complexity classes of decision problems.

Definition 13. P *is the set of decision problems which can be solved by a deterministic Turing machine in polynomial time.* NP *is the set of decision problems that can be solved by a non-deterministic Turing machine in polynomial time. A decision problem is* NP-complete *if it is in* NP *and every other problem in* NP *is reducible to it.*

Definition 14 (Independent Set Problem, IS). *Given a simple, undirected graph* $G = (V, E)$ *and a positive integer* $k \leq |V|$*, is there an independent set* $I \subseteq V$ *in* G *such that* $|I| \geq k$*?*

Lemma 1. *If* Y *is an* NP-complete *problem, and* X *is a problem in* NP *with the property that* Y *is polynomial-time reducible to* X*, then* X *is* NP-complete*.*

Proof. See reference [18].

Lemma 2. IS *is* NP-complete*.*

Proof. See reference [18].

3.2 Concurrently Enabled Transition Set Problem

Definition 15 (Concurrently Enabled Transition Set Problem, CETS). *Given a Petri net system* (Σ, M) *and a positive integer* $k \leq |T|$*, is there a concurrently enabled transition set* $C \subseteq T$ *in* (Σ, M) *such that* $|C| \geq k$*?*

Theorem 1. CETS *is* NP-complete*.*

Proof. Consider a nondeterministic algorithm (see Algorithm 1).

Algorithm 1. A nondeterministic algorithm for CETS

Input: $(P, T, Pre, Post), M, k$
begin
Nondeterministically guess a transition set T_1, and verify that
(i) $|T_1| \geq k$
(ii) $\sum_{t \in T_1} Pre(t) \leq M$
If all these conditions hold, then accept
end

Assume that $|P| = m, |T| = n$, and v be the greatest integer found over the incidence matrix $Pre, Post$ and the marking M, then the sizes of $P, T, Pre, Post$, M and k are $O(logm), O(logn), O(mnlogv), O(mnlogv), O(mlogv)$ and $O(logn)$ respectively, hence the total input size belongs to $O(mnlogv)$. The length of T_1 is $O(n)$, thus verifying (i) needs time $O(n)$. And verifying (ii) needs time

$O(mn^2logv)$. So the time efficiency of this algorithm is in $O(mn^2logv)$. This is a polynomial-time nondeterministic algorithm. Therefore, CETS $\in$ NP.

To prove the NP-completeness, we shall reduce IS to CETS. Given an instance of IS, note that it consists of a graph $G = (V, E)$ and an integer s. The reduction function maps such instance to the following instance of CETS:

$v_i \in V$ if and only if $t_i \in T$

$(v_i, v_j) \in E$ if and only if $p_{ij} \in P \wedge Pre(p_{ij}, t_i) = 1 \wedge Pre(p_{ij}, t_j) = 1 \wedge M(p_{ij}) = 1$

v_i is an isolated vertex if and only if $p_i \in P \wedge Pre(p_i, t_i) = 1 \wedge M(p_i) = 1$

$k = s$

It is easy to see that a transition t_i in Petri net system (Σ, M) corresponds to a vertex v_i in graph G, and every transition is guaranteed to be enabled at M. Two enabled transitions share a common place, if and only if their corresponding vertices are adjacent. A place has only an output transition, if and only if the vertex corresponding to this output transition is isolated. Fig. 2 shows an example of the construction from graph G to Petri net system (Σ, M). Graph G includes an independent set $\{v_1, v_3, v_4\}$ in Fig. 2(a). And by using the above map, a Petri net system (Σ, M) is obtained in Fig 2(b), which also includes a concurrently enabled transition set $\{t_1, t_3, t_4\}$.

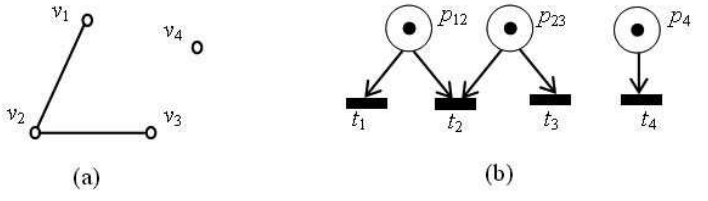


Fig. 2. An example of construction from graph G to Petri net system (Σ, M)

Assume that I is an independent set with $|I| \geq s$ in graph G, then we have $\forall v_i, v_j \in I$: $(v_i, v_j) \notin E$ by Definition 12, and this holds if and only if $\forall t_i, t_j \in C$: ${}^\bullet t_i \cap {}^\bullet t_j = \emptyset$ according to the reduction function. Further, this holds if and only if $\forall t_i, t_j \in C$: $t_i || _M t_j$ by Proposition 1(2), and this means that C is a concurrently enabled transition set with $|C| \geq k$ in Petri net system (Σ, M) by Definition 7. Thus, I is an independent set with $|I| \geq s$ in G if and only if C is a concurrently enabled transition set with $|C| \geq k$ in (Σ, M).

Now we prove that the map is indeed a polynomial-time reduction from IS to CETS.

Let $|V| = n$, then the input sizes of V, E and s are respectively $O(logn), O(n^2)$ and $O(logn)$. Constructing transition set T from vertex set V is only in $O(n)$, and determining place set P from edge set E needs time $O(n^2)$. Since the weight of every arc and the number of token in every place are 1, incidence matrices Pre and $Post$ are constructed in time $O(n^3)$ and marking M in time $O(n^2)$. Therefore, this is a polynomial-time reduction.

As IS has been proved NP-complete, and CETS is in NP, by Lemma 1 it follows that CETS is NP-complete. □

3.3 Maximal Concurrently Enabled Transition Set Problem

In computational complexity theory, a function problem [19] is a problem other than a decision problem, that is, a problem requiring a more complex answer than just YES or NO.

Definition 16. FP *is the set of function problems which can be solved by a deterministic Turing machine in polynomial time.*

Definition 17 (Maximal Concurrently Enabled Transition Set Problem, MCETS). *Given a Petri net system (Σ, M), to find a maximal concurrently enabled transition set C.*

Obviously, CETS is a decision problem, while MCETS is a function problem.

Theorem 2. MCETS $\in$ FP

Proof. Consider a deterministic algorithm (see Algorithm 2).

Algorithm 2. A deterministic algorithm for MCETS

```
Input: Σ, M
begin
    C ← ∅
    for each transition t in T
      if (Pre(t) ≤ M) then
        C ← C ∪ {t}
        M ← M − Pre(t)
    return C
end
```

This algorithm begins with an arbitrary transition t in T. If t is enabled at M, then it is added to transition set C, and $Pre(t)$ is subtracted from M; otherwise, the next transition is selected. This process continues until all transitions in T are visited. At this time the returned set is a maximal concurrently enabled transition set.

Let C_i and M_i be the value of variables C and M on the ith while iteration, respectively. We prove by mathematical induction that C_i is a concurrently enabled transition set.

For the basis case ($i = 0$), $M_0 = M$ and $C_0 = \emptyset$. Because of $\emptyset \in ConSet_\Sigma(M)$, the assertion holds for $i = 0$.

Now assume that the assertion holds for $i \leq k$, i.e. C_k is a concurrently enabled transition set and $M_k = M - \sum_{t \in C_k} Pre(t)$.

Consider $i = k+1$. Assume that t_{k+1} is selected. If $t_{k+1} \in En(M_k)$, by induction assumption,then $Pre(t_{k+1}) \leq M_k = M - \sum_{t \in C_k} Pre(t)$, i.e., $\sum_{t \in C_k} Pre(t) + Pre(t_{k+1}) \leq M$. By Definition 7, $C_{k+1} = C_k \cup \{t_{k+1}\}$ is a concurrently enabled

transition set. Otherwise, if $t_{k+1} \notin En(M_k)$, then $C_{k+1} = C_k$ is a concurrently enabled transition set by induction assumption.

Next we prove that C must be a maximal concurrently enabled transition set when the algorithm stops.

Assume that C_n and M_n are the value of variables C and M when the algorithm terminates. If C_n is not a maximal concurrently enabled transition set, then this implies that there must exist a concurrently enabled transition set C' such that $C_n \subset C'$. Without loss of generality, assume that $t' \in C'$ and $t' \notin C_n$, thus we have $C_n \cup \{t'\} \subseteq C'$. Since C' is a concurrently enabled transition set, by Definition 7, we obtain $\sum_{t \in C_n} Pre(t) + Pre(t') \leq \sum_{t \in C'} Pre(t) \leq M$, i.e., $Pre(t') \leq M - \sum_{t \in C_n} Pre(t) = M_n$. According to the condition of the while iteration, the algorithm is not terminated. This is a contradiction.

Since n iterations are made and each iteration's time efficiency is in $O(mlogv+n)$, where $m = |P|, n = |T|$ and v is the greatest integer found over the incidence matrix $Pre, Post$ and the marking M, the running time of the algorithm is $O(mnlogv + n^2)$.

Therefore, MCETS can be computed in polynomial time. □

4 Subproblem Analysis

In this section, we will analyze some special subproblems solvable in polynomial time.

Theorem 3. *Let $\Sigma = (P, T, Pre, Post)$ be an asymmetric choice net and $R = \{(t_1, t_2) | t_1, t_2 \in T \wedge {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset\}$ a binary relation over T, then R is an equivalence relation.*

Proof. $\forall t \in T$: ${}^\bullet t \cap {}^\bullet t \neq \emptyset$, then $(t, t) \in R$. Hence R is reflexive.

$\forall t \in T$: $t_1, t_2 \in T$, if $(t_1, t_2) \in R$, then ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, i.e., ${}^\bullet t_2 \cap {}^\bullet t_1 \neq \emptyset$, thus $(t_2, t_1) \in R$. So R is symmetric.

$\forall t \in T$: $t_1, t_2, t_3 \in T$, if $(t_1, t_2) \in R$ and $(t_2, t_3) \in R$, then ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ and ${}^\bullet t_2 \cap {}^\bullet t_3 \neq \emptyset$. Without loss of generality, assume that $p_{12} \in {}^\bullet t_1 \cap {}^\bullet t_2$ and $p_{23} \in {}^\bullet t_2 \cap {}^\bullet t_3$, then $\{t_1, t_2\} \in p_{12}{}^\bullet$ and $\{t_2, t_3\} \in p_{23}{}^\bullet$, thus $t_2 \in p_{12}{}^\bullet \cap p_{23}{}^\bullet$. Since Σ is an asymmetric choice net, we obtain $p_{12}{}^\bullet \subseteq p_{23}{}^\bullet$ or $p_{23}{}^\bullet \subseteq p_{12}{}^\bullet$, i.e., $\{t_1, t_2, t_3\} \subseteq p_{12}{}^\bullet$ or $\{t_1, t_2, t_3\} \subseteq p_{23}{}^\bullet$. And then ${}^\bullet t_1 \cap {}^\bullet t_3 \neq \emptyset$, i.e., $(t_1, t_3) \in R$. So R is transitive.

Therefore, R is an equivalence relation. □

Note that Theorem 3 also holds for free-choice nets and extended free-choice nets, because they are both the subclasses of asymmetric choice nets [1].

Let $[t] = \{t' | (t, t') \in R\}$ be the equivalence class of a transition t in T and $T/R = \{[t] | t \in T\}$ the quotient set of T by R. Furthermore, we denote by R_M the restriction of R to a subset $En(M)$ of T. It is easy to see that R_M is also an equivalence relation, thus we can define $[t]_M = \{t' | (t, t') \in R_M\}$ and $T/R_M = \{[t]_M | t \in En(M)\}$.

Theorem 4. *Let* (Σ, M) *be a Petri net system and* $R = \{(t_1, t_2) | t_1, t_2 \in T \wedge {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset\}$ *a binary relation over* T*. If the following conditions are satisfied:*
(1) R *is an equivalence relation, and*
(2) $\forall t_1, t_2 \in En_\Sigma(M)$: $t_1 || _M t_2$ *if and only if* ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$,
then $MaximumCS_\Sigma(M) = MaximalCS_\Sigma(M)$.

Proof. We only need to prove that $|C| = |T/R_M|$ for all $C \in MaximalCS_\Sigma(M)$.

Let $f : C \to T/R_M$ be a (total) function such that $f(t) = [t]_M$ for all $t \in C$. We will prove that f is a bijection.

According to Definition 7, $\forall t_1, t_2 \in C (t_1 \neq t_2)$: $t_1 || _M t_2$. Then ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$ by condition (2). And then $(t_1, t_2) \notin R$ and $[t_1]_M \neq [t_2]_M$ by condition (1). Therefore, f is an injection.

Assume that $\exists [t]_M \in T/R_M, \forall t_i \in C$: $f(t_i) = [t_i]_M \neq [t]_M$, then $\forall t_i \in C$: $(t_i, t) \notin R \wedge {}^\bullet t_i \cap {}^\bullet t = \emptyset$ by condition (1), and then $\forall t_i \in C$: $t_i || _M t$ by condition (2). Because C is a concurrently enabled transition set, so is $C \cup \{t\}$ by Definition 7. Thus C is not a maximal concurrently enabled transition set by Definition 8. This is a contradiction. Therefore, f is a surjective.

Because C and T/R_M are both finite sets and there must exists a bijection between them, we obtain $\forall C \in MaximalCS_\Sigma(M)$: $|C| = |T/R_M|$. Therefore, $MaximumCS_\Sigma(M) = MaximalCS_\Sigma(M)$. □

Theorem 5. *Assume that* (Σ, M) *is a Petri net system. If* $MaximumCS_\Sigma(M) = MaximalCS_\Sigma(M)$*, then* CETS $\in$ P.

Proof. Assume that we have a black box FIND-MCETS(Σ, M) that finds a maximal concurrently enabled transition set using Algorithm 2. Now consider a deterministic algorithm for CETS (See Algorithm 3).

Algorithm 3. A deterministic algorithm for CETS

```
Input: Σ, M, k
begin
    S ← FIND-MCETS(Σ, M)
    if |S| ≥ k then accept
    else reject
end
```

According to Theorem 2, a maximal concurrently enabled transition set S can be returned by FIND-MCETS(Σ, M) in polynomial time. And it is also a maximum concurrently enabled transition set because of $MaximumCS_\Sigma(M) = MaximalCS_\Sigma(M)$. And then we can obtain a yes-or-no answer only by comparing the size of S with k. Therefore, CETS can be solved in polynomial time by the deterministic algorithm. □

Corollary 1. *Assume that* (Σ, M) *is an extended free choice net system, then* CETS $\in$ P.

Proof. According to Theorem 3, Proposition 1(1), Theorem 4 and Theorem 5, it follows that CETS is in P. □

Corollary 2. *Assume that* (Σ, M) *is an asymmetric choice net system. If* $\forall p \in P$: $M(p) \in \{0, 1\}$, *then* CETS $\in$ P.

Proof. According to Theorem 3, Proposition 1(2), Theorem 4 and Theorem 5, it follows that CETS is in P. □

Corollary 3. *Assume that* (Σ, M) *is a Petri net system and* $R = \{(t_1, t_2) | t_1, t_2 \in T \wedge {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset\}$ *is an equivalence relation over* T. *If* $\forall t \in T, \forall p \in {}^\bullet t$: $M(p) < 2Pre(p, t)$, *then* CETS $\in$ P.

Proof. According to Proposition 1(3), Theorem 4 and Theorem 5, it follows that CETS is in P. □

5 Conclusions

This paper has presented the concurrently enabled transition set problem (CETS), and determined the exact complexity of this problem. CETS turns out to be NP-complete. The NP-hardness proof is a straightforward reduction from the independent set problem. A good result that CETS for extended free choice nets and asymmetric choice nets belongs to P is also obtained in this paper. This work will be quite useful to further study the concurrency of Petri nets.

Our future research will aim at developing approximation or heuristic algorithms to solve the problem.

References

1. T. Murata. Petri nets, properties, analysis and applications, Proc. IEEE 77(4) (1989) 541-580.
2. C. Girault and R. Valk. Petri Nets for Systems Engineering A Guide to Modeling, Verification, and Applications. Springer Verlag, 2003.
3. J. Jeffrey, J. Lobo and T. Murata. A High-Level Petri Net for Goal-Directed Semantics of Horn Clause Logic. IEEE Transactions on Knowledge and Data Engineering., Vol. 8, No. 2, pp.241 - 259, 1996.
4. M. Mukund. Petri Nets and Step Transition Systems. International Journal of Foundations of Computer Science 3, 4, (1992) 443-478.
5. T. Jucan and C. Vidraşcu. Concurrency-degrees for Petri nets. In Proc. of the 1st Conference on Theoretical Computer Science and Informatics Technologies - CITTI 2000, Ovidius University of Constanta, Romania, May 25-27, 2000. pages 108-114.
6. F. Vernadat, P. Azéma and F. Michel. Covering step graph. In: Proc. of the 17th Intel Conf. on Application and Theory of Petri Nets 96. LNCS 1091, Osaka: Springer-Verlag, 1996. 516-535.
7. P. O. Ribet, F. Vernadat and B. Berthomieu. On combining the persistent sets method with the covering steps graph method. In: Proc. of the FORTE 02. Springer Verlag, LNCS 2529, 2002. pages 344-359.

8. L. Pan, D. Chen, and W. Li. Reachability Analysis of Time-Independent Choice TPN. In the 10th Joint International Computer Conference, Kunming, China, Novermber 2004. 629-634.
9. A. Cheng, J. Esparza and J. Palsberg. Complexity Results for 1-safe Nets. Theoret. Comput. Sci.147 (1995) 117-136.
10. J. Esparza. Reachability in live and safe free-choice Petri nets is NP-complete. Theoretical Computer Science, Volume 198, Issue 1-2, May 1998, Pages: 211 - 224.
11. I. A. Stewart. On the reachablility problem for some classes of Petri nets. Research Report, University of Newcastle upon Tyne, 1992.
12. R. Howell and L. Rosier. Completeness results for conflict-free vector replacement system. Journal of Computer and System Science 37(1988), 349-366.
13. R. Howell, L. Rosier and H. Yen. Normal and Sinkless Petri Nets. Journal of Computer and System Sciences 46(1993), 1-26.
14. T. Watanabe, et al. Time complexity of legal firing sequences and related problems of Petri nets, Trans. IEICE of Japan, 1989, 72(12): 1400-1409.
15. T. Watanabe, Y. Mizobata and K. Onaga. Legal firing sequences and minimum initial markings for Petri nets. IEEE International Symposium on Circuits and Systems, 1989, vol.1, page(s): 323-326.
16. E. Badouel, L. Bernardinello and P. Darondeau. The synthesis problem for elementary net systems is NP-complete. Theoretical Computer Science 186(1997):107-l34.
17. J. Esparza. Decidability and complexity of Petri net problems - an introduction, Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Lecture Notes in Computer Science, Vol.1491 (Springer Verlag, 1998) 374-428.
18. M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.
19. C. H. Paradimitriou. Computational Complexity. Addison-Wesley Publishing Company, 1994.

Synchronization of Some DFA

A.N. Trahtman

Bar-Ilan University, Dep. of Math., 52900, Ramat Gan, Israel
trakht@macs.biu.ac.il

Abstract. A word w is called synchronizing (recurrent, reset, directable) word of deterministic finite automaton (DFA) if w brings all states of the automaton to an unique state. Černy conjectured in 1964 that every n-state synchronizable automaton possesses a synchronizing word of length at most $(n-1)^2$. The problem is still open.

It will be proved that the minimal length of synchronizing word is not greater than $(n-1)^2/2$ for every n-state ($n > 2$) synchronizable DFA with transition monoid having only trivial subgroups (such automata are called *aperiodic*). This important class of DFA accepting precisely star-free languages was involved and studied by Schŭtzenberger. So for aperiodic automata as well as for automata accepting only star-free languages, the Černý conjecture holds true.

Some properties of an arbitrary synchronizable DFA and its transition semigroup were established.

http://www.cs.biu.ac.il/∼trakht/syn.html

Keywords: Deterministic finite automata, synchronization, aperiodic semigroup, Černy conjecture.

1 Introduction

The natural problem of synchronization of DFA draws quite often the attention and various aspects of this problem were touched upon the literature. The synchronization makes the behavior of an automaton resistant against input errors since, after detection of an error, a synchronizing word can reset the automaton back to its original state, as if no error had occurred.

An important problem with a long story is the estimation of the shortest length of synchronizing word of DFA. Best known as Černy's conjecture, it was raised independently by distinct authors. Jan Černy found in 1964 [1] an n-state automaton with minimal length synchronizing word of $(n-1)^2$. He conjectured that this is the maximum length of the shortest synchronizing word for any DFA with n states. The conjecture is valid for big list of objects, but in general the question still remains open. The best known upper bound is now equal to $(n^3-n)/6$ [3,5,7]. By now, this simple looking conjecture with rich and intriguing story of investigations [4,7,10,12] is one of the most longstanding open problems in the theory of finite automata.

The existence of some non-trivial subgroup in the transition semigroup of the automaton is essential in many investigations of Černy conjecture [2,7,8].

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 234–243, 2007.
© Springer-Verlag Berlin Heidelberg 2007

We use an opposite approach and consider transition semigroups without non-trivial subgroups. This condition distinguishes a wide class of so-called aperiodic automata that, as shown by Schützenberger [11], accept precisely star-free languages (also known as languages of star height 0). Star-free languages play a significant role in the formal language theory.

It will be established that the synchronizable DFA has a synchronizing word of length not greater than $(n-1)^2/2$ $(n > 2)$ for automata with transition semigroup having only trivial subgroups (*aperiodic* automata) and therefore the Černy conjecture holds true for such DFA.

The necessary and sufficient conditions of synchronizability of an arbitrary automaton are presented below in the following form:

An automaton with transition graph Γ is synchronizable iff Γ^2 has sink state. (see also [1] for another wording). In the case of aperiodic automata holds:

An aperiodic automaton with sink state is synchronizable.

Some properties of an arbitrary synchronizable DFA were found by help of some new concepts such as almost minimal SCC, m-cycle and set of 2-reset words.

2 Preliminaries

We consider a complete DFA $\mathcal{A}$ with the input alphabet Σ, the transition graph Γ and the transition semigroup S. The elements of S let us consider as words over Σ.

A maximal strongly connected component of a directed graph will be denoted for brevity as **SCC**.

If there exists a path from the state $\mathbf{p}$ to $\mathbf{q}$ and the consecutive transitions of the path are labelled by $\sigma_1, ..., \sigma_k$ then for the word $s = \sigma_1 ... \sigma_k$ let us write $\mathbf{q} = \mathbf{p}s$.

The state $\mathbf{q}$ is called *sink* if for every state $\mathbf{p}$ there exists a word s such that $\mathbf{p}s = \mathbf{q}$.

The binary relation β is called *stable* if for any pair of states $\mathbf{q}, \mathbf{p}$ and any $\sigma \in \Sigma$ from $\mathbf{q}\ \beta\ \mathbf{p}$ follows $\mathbf{q}\sigma\ \beta\ \mathbf{p}\sigma$.

The graph Γ is *complete* if for every $\mathbf{p} \in \mathbf{\Gamma}$ and every $\sigma \in \Sigma$ the state $\mathbf{p}\sigma$ exists.

$|s|$ - the length of the word s in alphabet Σ.

$|P|$ - the size of the set of states of the automaton (of vertices of the graph) P.

Let Ps denote the mapping of the graph (of the automaton) P by help of $s \in \Sigma^*$.

The direct product Γ^2 of two copies of the transition graph Γ over an alphabet Σ consists of pairs $(\mathbf{p}, \mathbf{q})$ and edges $(\mathbf{p}, \mathbf{q}) \to (\mathbf{p}\sigma, \mathbf{q}\sigma)$ labelled by σ. Here $\mathbf{p}, \mathbf{q} \in \Gamma$, $\sigma \in \Sigma$.

A word $s \in \Sigma^+$ is called *synchronizing (reset)* word of an automaton with transition graph Γ if $|\Gamma s| = 1$.

A word w is called *2-reset word* of the pair $\mathbf{p}, \mathbf{q}$ if $\mathbf{p}w = \mathbf{q}w$ and let us denote by $Syn(\mathbf{p}, \mathbf{q})$ the set of all such words w.

Let ϕ be homomorphism of the DFA $\mathcal{A}$. Suppose $\mathbf{q}\ \rho\ \mathbf{p}$ iff $\mathbf{q}\phi = \mathbf{p}\phi$ for the states $\mathbf{q}$ and $\mathbf{p}$ from $\mathcal{A}$. Then the relation ρ is a *congruence* on $\mathcal{A}$. The ρ-class containing the state $\mathbf{q}$ of $\mathcal{A}$ we denote by $\mathbf{q}^\rho$. The *quotient* $\mathcal{A}/\rho$ is the automaton with the set of states $\mathbf{q}^\rho$ and the transition function defined by the rule $\mathbf{q}^\rho\sigma = \mathbf{q}\sigma^\rho$ for any $\sigma \in \Sigma$.

An *SCC* M from Γ^2 will be called *almost minimal* if for every state $(\mathbf{p}, \mathbf{q}) \in M$ and for every $\sigma \in \Sigma$ such that $\mathbf{p}\sigma \neq \mathbf{q}\sigma$ there exists a word s such that $(\mathbf{p}\sigma, \mathbf{q}\sigma)s = (\mathbf{p}, \mathbf{q})$. For every $(\mathbf{p}, \mathbf{q}) \in M$ suppose $\mathbf{p}$ and $\mathbf{q} \in \Gamma(M)$.

Let us define a relation $\succ_M$ for almost minimal *SCC* M. Suppose $\mathbf{p} \succ_M \mathbf{q}$ if $(\mathbf{p}, \mathbf{q}) \in M$ and let $\succ_M$ be the transitive closure of this relation. Let $\succeq_M$ be the reflexive closure and ρ_M be equivalent closure of the relation $\succ_M$.

Let M be almost minimal *SCC*. A non-trivial sequence of states $\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_n = \mathbf{p}_1$ such that $(\mathbf{p}_i, \mathbf{p}_{i+1})$ for $i = 1, ..., n-1$ $(n > 1)$ belong to M let as call *t-cycle*.

A t-cycle of minimal length let as call *m-cycle*.

3 The Graph Γ^2

Lemma 1. *The relation $\succeq_M$ for any almost minimal SCC $M \in \Gamma^2$ is stable. The equivalent closure ρ_M of the relation $\succ_M$ is a congruence.*

If R is a class of the relation ρ_M then for any word w the set Rw is a subset of a class of the relation ρ_M.

Proof. In the case $\mathbf{u} \succeq_M \mathbf{v}$ there exist a sequence of states $\mathbf{u} = \mathbf{p}_1, ..., \mathbf{p}_n = \mathbf{v}$ such that for every integer $i < n$ $(\mathbf{p}_i, \mathbf{p}_{i+1})$ belongs to the almost minimal *SCC* M. One has $(\mathbf{p}_i s, \mathbf{p}_{i+1}s) \in M$ or $\mathbf{p}_i s = \mathbf{p}_{i+1}s$ and therefore $\mathbf{p}_i s \succeq_M \mathbf{p}_{i+1}s$. Consequently, $\mathbf{u}s \succeq_M \mathbf{v}s$.

Suppose $\mathbf{u}\ \rho_M\ \mathbf{v}$. Then there exist a sequence of states $\mathbf{u} = \mathbf{p}_1, ..., \mathbf{p}_n = \mathbf{v}$ such that for every integer $i < n$ at least one of the states $(\mathbf{p}_{i+1}, \mathbf{p}_i)$, $(\mathbf{p}_i, \mathbf{p}_{i+1})$ belongs to the almost minimal *SCC* M. Therefore in the sequence of states $\mathbf{u}s = \mathbf{p}_1 s, ..., \mathbf{p}_n s = \mathbf{v}s$ for any two distinct neighbors $\mathbf{p}_i s, \mathbf{p}_{i+1}s$ the state $(\mathbf{p}_i s, \mathbf{p}_{i+1}s)$ or its dual belongs to M. Consequently, $\mathbf{u}s\ \rho_M\ \mathbf{v}s$. Therefore for a class R of the relation ρ_M and for any word w the set Rw is a subset of some class of the relation ρ_M.

From the definitions of the almost minimal *SCC* and the relation $\succ_M$ follows

Proposition 2. *If $\mathbf{r} \succ_M \mathbf{q}$ and for some word s one has $\mathbf{r}s \notin \Gamma(M)$ or $\mathbf{q}s \notin \Gamma(M)$, then $\mathbf{r}s = \mathbf{q}s$.*

Proposition 3. *Synchronizing word of Γ synchronizes also Γ/ρ_M for any M.*

Synchronizing word of Γ is 2-reset word for any pair of states and therefore unites every pair of states from different ρ_M-classes in one sink state.

The following lemma can be also reduced from [1]:

Lemma 4. *An automaton $\mathcal{A}$ with transition graph Γ is synchronizable if and only if Γ^2 has a sink state.*

Proof. Let s be synchronizing word of $\mathcal{A}$. Then the unique pair of the set $\Gamma^2 s$ is a sink of Γ^2.

Conversely, the components of a sink of Γ^2 obviously are equal. Let $(\mathbf{t}, \mathbf{t})$ be a sink. For any pair $(\mathbf{p}, \mathbf{q})$, there exists a word s such that $(\mathbf{p}, \mathbf{q})s = (\mathbf{t}, \mathbf{t})$, that is, $\mathbf{p}s = \mathbf{q}s = \mathbf{t}$. Some product of such words s taken for all pairs of distinct states from Γ is a synchronizing word of the graph Γ.

Lemma 5. *Let M be almost minimal SCC of Γ^2. Suppose that for some word s the state $\mathbf{q}s$ is either a maximal element of the order $\succ_M$ or $\mathbf{q}s \notin \Gamma(M)$.*

Then for any state $\mathbf{t}$ such that $\mathbf{t} \succeq_M \mathbf{q}$ holds $\mathbf{t}s = \mathbf{q}s$. The word s unites all ancestors of $\mathbf{q}$.

Proof. By Lemma 1, $\mathbf{t}s \succ_M \mathbf{q}s$ or $\mathbf{q}s = \mathbf{t}s$. The case $\mathbf{t}s \succ_M \mathbf{q}s$ is excluded because the state $\mathbf{q}s$ is a maximal state. In the case $\mathbf{q}s \notin \Gamma(M)$ also $\mathbf{t}s = \mathbf{q}s$ by Proposition 2. Thus the word s is a common synchronizing word for set of all states $\mathbf{t}$ such that $\mathbf{t} \succeq_M \mathbf{q}$.

Theorem 1. *Let M be almost minimal SCC of Γ^2 of n-state synchronizable automaton with transition graph Γ over an alphabet Σ. Let R be ρ_M-class defined by states from M and suppose that $|Rv| = 1$ for non-empty word v of length not greater [less] than $c(|R| - 1)(n - 1)$ where c is some coefficient.*

Then the automaton has synchronizing word in alphabet Σ of length not greater [less] than $c(n - 1)^2$.

Proof. Let us use the induction on n. The case $n = 2$ for non-empty v is trivial, so let us suppose $n > 2$. For $|R| = n$ the statement of the theorem is a tautology, so let assume $|R| < n$. In view of $|\Gamma(M)| > 1$ the relation ρ_M is not trivial and $|R| > 1$. So

$$n > |R| > 1$$

Let us consider the congruence ρ_M and the quotient Γ/ρ_M. Any synchronizing word of Γ synchronizes also Γ/ρ_M (Proposition 3). Therefore the graph Γ has a synchronizing word uv where u is a synchronizing word of Γ/ρ_M and v is a synchronizing word of the preimage R of Γ/ρ_M u. In view of $n > |R| > 1$, one has $|\Gamma/\rho_M| \leq n - |R| + 1 < n$ and we can use induction, assuming $|u| \leq c(|\Gamma/\rho_M| - 1)^2 \leq c(n - |R|)^2$. So $|uv| \leq c(n - |R|)^2 + c(|R| - 1)(n - 1)$ and in view of $c(n-|R|)^2 + c(|R|-1)(n-1) = c((n-1)^2 - (|R|-1)(n-|R|) < c(n-1)^2$ one has $|uv| < c(n - 1)^2$. So for $n > |R|$ the length is less than $c(n - 1)^2$.

Lemma 6. *For any word w, $Syn(\mathbf{p}, \mathbf{q}) \subseteq Syn(\mathbf{r}, \mathbf{t})$ implies $Syn(\mathbf{p}w, \mathbf{q}w) \subseteq Syn(\mathbf{r}w, \mathbf{t}w)$. The relation $Syn(\mathbf{p}, \mathbf{q})$ is a stable binary relation.*

Proof. Suppose word $u \in Syn(\mathbf{p}w, \mathbf{q}w)$. Therefore the word wu synchronizes the pair of states $\mathbf{p}, \mathbf{q}$. From $Syn(\mathbf{p}, \mathbf{q}) \subseteq Syn(\mathbf{r}, \mathbf{t})$ follows that the word wu synchronizes the pair of states $\mathbf{r}, \mathbf{t}$ and $\mathbf{r}wu = \mathbf{t}wu$. Thus the word u from $Syn(\mathbf{p}w, \mathbf{q}w)$ synchronizes also the pair $(\mathbf{r}w, \mathbf{t}w)$, whence $Syn(\mathbf{p}w, \mathbf{q}w) \subseteq Syn(\mathbf{r}w, \mathbf{t}w)$.

4 Transition Semigroup of Automaton

For definitions of D- and H-class, ideal, left ideal, idempotent and right zero see [6].

Lemma 7. *Let Γ be strongly connected graph of synchronizing automaton with transition semigroup S. Suppose $\Gamma a = \Gamma b$ for reset words a and b. Then $a = b$. Any reset word is an idempotent.*

Proof. The elements a and b from S induce equal mappings on the set of states of Γ. S can be embedded into the semigroup of all functions on the set of states under composition [6]. Therefore $a = b$ in S. $\Gamma a = \Gamma a^2$, whence $a = a^2$ for any reset word a and the element $a \in S$ is an idempotent.

Lemma 8. *Let Γ be strongly connected graph of synchronizable automaton with transition semigroup S. Suppose eSe is a group for some idempotent e.*

Then every element s from eSe is a reset word and $|\Gamma s| = 1$.

Proof. Suppose two states $\mathbf{p}$ and $\mathbf{q}$ belong to Γs. The automaton is synchronizable, whence there exists a word $t \in S$ such that $\mathbf{p}t = \mathbf{q}t$. The states $\mathbf{p}$ and $\mathbf{q}$ belong to Γs, therefore $\mathbf{p}e = \mathbf{p}$, $\mathbf{q}e = \mathbf{q}$ and $\mathbf{p}et = \mathbf{q}et$. It implies $\mathbf{p}ete = \mathbf{q}ete$. The element ete belongs to the group eSe with unit e and has an inverse in this group. Consequently, $\mathbf{p}e = \mathbf{q}e$. The states $\mathbf{p}$ and $\mathbf{q}$ belong to $\Gamma s = \Gamma se$. So $\mathbf{p}e = \mathbf{p}$, $\mathbf{q}e = \mathbf{q}$, whence $\mathbf{p} = \mathbf{q}$ in spite of our assumption.

Lemma 9. *Let Γ be strongly connected graph of synchronizable n-state automaton with transition semigroup S.*

Then S contains n distinct reset words and they form a D-class D_r of S. D_r is an ideal of S and a subsemigroup of right zeroes, subgroups of D_r are trivial.

Proof. For every state $\mathbf{p}$ from synchronizable automaton with strongly connected transition graph by Lemma 7 there exists at most one reset word s such that $\Gamma s = \mathbf{p}$. Γ is strongly connected, consequently for any state $\mathbf{q}$ there exists a word t such that $\mathbf{p}t = \mathbf{q}$. Then $\Gamma st = \mathbf{q}$ and st is also a reset word. So for any state there exists its reset word and all these reset words of distinct states are distinct. Thus there are at least n reset words. From $\Gamma s = \mathbf{p}$ follows $\Gamma us = \mathbf{p}$ for every word $u \in S$, whence by Lemma 7 $us = s$, and in particular $ts = s$. Because for every $t \in S$ st is also reset word and $ts = s$ the set of all reset words is an ideal in S.

For two states $\mathbf{p}$ and $\mathbf{q}$ the corresponding reset words are s and st. So the second word belongs to the left ideal generated by the first. The states $\mathbf{p}$ and $\mathbf{q}$ are arbitrary, whence all reset words create the same left ideal and belong to one D-class D_r. All words from D-class D_r are reset words because from $|\Gamma s| = 1$ follows $|\Gamma vsu| = 1$ for any words u and v. Distinct reset words map Γ on distinct states, so in view of Lemma 7 $|D_r| = n$.

Any H-class of D-class D_r is a group with one idempotent [6] and consists of reset words. By lemma 7 any reset word is an idempotent. Therefore any H-class is trivial and in view of Lemma 8 the set of reset words D_r is a semigroup of right zeroes.

Corollary 10. *Let Γ be transition graph of synchronizable n-state automaton with transition semigroup S. Let Γ have sink strongly connected maximal subgraph T of size k.*

Then S contains k distinct reset words and they all are idempotents, form a subsemigroup of right zeroes and an ideal of S. The difference between the minimal lengths of two distinct reset words is less than k.

Proof. Any reset word of Γ is also a reset word of T. Therefore there are only k distinct reset word and by Lemma 9 they are idempotents, form an ideal of S and a subsemigroup of right zeroes.

Any reset word u maps Γ on some state $\mathbf{p}$ of T. Any other reset word v such that $\Gamma v = \mathbf{q}$ can be obtained as a product ut for t such that $\mathbf{p}t = \mathbf{q}$. $|t| < k$, whence $|v| - |u| < k$.

5 The State Outside *t*-Cycle

Lemma 11. *Let M be almost minimal SCC from Γ^2 having some t-cycle. Then for any state $(\mathbf{q}, \mathbf{p}) \in M$ the states $\mathbf{q}$ and $\mathbf{p}$ are consecutive states of t-cycle and even of m-cycle.*

If the states $\mathbf{r}$, $\mathbf{s}$ of m-cycle are not consecutive then the state $(\mathbf{r}, \mathbf{s})$ of Γ^2 does not belong to M.

Proof. A t-cycle of minimal length exists because Γ^2 is finite. Let the states $\mathbf{q}_1, \mathbf{p}_1$ be consecutive states of m-cycle C of length m from almost minimal SCC M of Γ^2. So $(\mathbf{q}_1, \mathbf{p}_1) \in M$. For any state $(\mathbf{q}, \mathbf{p})$ from strongly connected component M there exists a word w such that $(\mathbf{q}_1, \mathbf{p}_1)w = (\mathbf{q}, \mathbf{p})$. The word w maps m-cycle C on some t-cycle of length j not greater than m. Because $\mathbf{p} \neq \mathbf{q}$ holds $j > 1$. Therefore $j = m$ and the states $\mathbf{q}, \mathbf{p}$ are consecutive states of m-cycle Cw.

If the state $(\mathbf{r}, \mathbf{s}) \in M$ but the states $\mathbf{r}$, $\mathbf{s}$ are not consecutive states of m-cycle from M then M contains a t-cycle with length less the length of m-cycle. Contradiction.

Lemma 12. *Let r be the number of ρ_M - classes of almost minimal SCC M of n-state ($n > 2$) automaton with strongly connected graph Γ. Suppose some state $\mathbf{p}$ does not belong to any t-cycle from M and let R be ρ_M - class from M.*

Then $|Rs| = 1$ for some word $s \in \Sigma^*$ of length not greater than $(n-r)(n-1)/2$.

Proof. The ρ_M-class R is defined by a state from M, therefore $|R| > 1$.

Suppose first that $\mathbf{p} \notin \Gamma(M)$. Then there exists a word w of length not greater than $n - |R|$ that maps some state from R on $\mathbf{p}$. In virtue of Lemma 1 the class Rw is out of $\Gamma(M)$ and therefore the definition of ρ_M - class implies $Rw = \mathbf{p}$. So $|Rw| = 1$. From $|R| > 1$ follows $(n - r)/2 \geq 1$. Therefore $|w| \leq n - |R| \leq (n-1)1 \leq (n-r)(n-1)/2$ in the case $\mathbf{p} \notin \Gamma(M)$.

Let us suppose now that $\mathbf{p} \in \Gamma(M)$. If there exists t-cycle in M then by Lemma 11 any state from $\Gamma(M)$ belongs to t-cycle. It contradicts to our assumption that $\mathbf{p}$ does not belong to t-cycle. Therefore we can suppose absence of t-cycles in M. The relation $\succ_M$ is a partial order in such case.

Let Max be the set of all maximal and Min be the set of all minimal states from R according to the order $\succ_M$. Both sets Max and Min are not empty in view of $|R| > 1$. $|Max| \cap |Min| = \emptyset$ because the relation $\succ_M$ is a partial order and anti-reflexive by definition. Without loss of generality, let us assume that $|Max| \geq |Min|$. Then $|Min| \leq (n-r)/2$. The relation $\succeq_M$ by Lemma 1 is stable. Therefore the number of all minimal states of Ru for any word u is also not greater than $(n-r)/2$.

For any minimal state $\mathbf{q}u \in Ru$ there exists a word w of length not greater than $n-1$ that maps the state $\mathbf{q}u$ in Max. By Lemma 5, the word w maps $\mathbf{q}u$ in Max together with all its ancestors. On this way, the number of minimal states can be reduced to zero by help of at most $|Min|$ words of length $n-1$ or less. The ρ_M-class without minimal elements is one-element, $|Min| \leq (n-r)/2$, whence for a word s of length not greater than $(n-1)(n-r)/2$ holds $|Rs| = 1$.

Theorem 2. *Let the transition graph Γ of an n-state ($n > 2$) synchronizable automaton be strongly connected and let M be almost minimal SCC of Γ^2. Suppose some state from Γ does not belong to t-cycle of M.*

Then the automaton has reset word of length not greater than $(n-1)^2/2$.

Proof. Let r be the number of ρ_M - classes. By Lemma 12, the word v of length not greater than $(n-r)(n-1)/2$ synchronizes any ρ_M-class defined by states of M. The size of any ρ_M-class R is not greater than $n-r+1$. Now by Theorem 1 for $c = 1/2$ in view of $|R|-1 \leq n-r$, there exists a synchronizing word of length not greater than $(n-1)^2/2$.

6 Aperiodic Strongly Connected DFA

Let us recall that the transition semigroup S of aperiodic automaton is finite and aperiodic [11] and the semigroup satisfies identity $x^n = x^{n+1}$ for some suitable n. So for any state $\mathbf{p} \in \Gamma$, any $s \in S$ and for some suitable k holds $\mathbf{p}s^k = \mathbf{p}s^{k+1}$.

Lemma 13. *Let $\mathcal{A}$ be an aperiodic automaton. Then the existence of sink state in $\mathcal{A}$ is equivalent to the existence of synchronizing word.*

Proof. It is clear that, for any DFA, the existence of a synchronizing word implies the existence of a sink.

Now suppose that $\mathcal{A}$ has at least one sink. For any state $\mathbf{p}$ and any sink $\mathbf{p_0}$, there exists an element s from the transition semigroup S such that $\mathbf{p}s = \mathbf{p_0}$. The semigroup S is aperiodic, whence for some positive integer m we have $s^m = s^{m+1}$. Therefore $\mathbf{p}s^m = \mathbf{p}s^{m+1} = \mathbf{p_0}s^m$, whence the element s^m brings both $\mathbf{p}$ and $\mathbf{p_0}$ to the same state $\mathbf{p_0}s^m$ which is a sink again. We repeat the process reducing the number of states on each step. Then some product of all elements of the form s^m arising on each step brings all states of the automaton to some sink. Thus, we obtain in this way a synchronizing word.

Let us go to the key lemma of the proof.

Lemma 14. *Let a DFA with the transition graph Γ be aperiodic.*

Then the graph Γ has no t-cycle, the quasi-order $\succeq_M$ for any almost minimal SCC M is a partial order and no state belongs to t-cycle.

Proof. Suppose the states $\mathbf{p}_1 \succ_M \mathbf{p}_2, .., \mathbf{p}_{m-1} \succ_M \mathbf{p}_m = \mathbf{p}_1$ form t-cycle of the minimal size m for some almost minimal SCC M.

Let us establish that $m > 2$. Indeed, $\mathbf{p}_1 \neq \mathbf{p}_2$ by the definition of the relation $\succ_M$, whence $m > 1$. If $m = 2$ then two states $(\mathbf{p}_1, \mathbf{p}_2)$ and $(\mathbf{p}_2, \mathbf{p}_1)$ belong to common SCC. For some element u from transition semigroup S, we have $(\mathbf{p}_1, \mathbf{p}_2)u = (\mathbf{p}_2, \mathbf{p}_1)$. Therefore $\mathbf{p}_1 u = \mathbf{p}_2$, $\mathbf{p}_2 u = \mathbf{p}_1$, whence $\mathbf{p}_1 u^2 = \mathbf{p}_1 \neq \mathbf{p}_1 u$. It implies $\mathbf{p}_1 u^{2k} = \mathbf{p}_1 \neq \mathbf{p}_1 u = \mathbf{p}_1 u^{2k+1}$ for any integer k. However, semigroup S is finite and aperiodic and therefore for some k holds $u^{2k} = u^{2k+1}$, whence $\mathbf{p}_1 u^{2k} = \mathbf{p}_1 u^{2k+1}$. Contradiction.

Thus we can assume that $m > 2$ and suppose that the states $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are distinct. For some element $s \in S$ and for the states $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ from considered t-cycle holds $(\mathbf{p}_1, \mathbf{p}_2)s = (\mathbf{p}_2, \mathbf{p}_3)$. We have

$$\mathbf{p}_2 = \mathbf{p}_1 s,\ \mathbf{p}_3 = \mathbf{p}_1 s^2$$

For any word $v \in S$ and any state $(\mathbf{p}_i, \mathbf{p}_{i+1})$ from M by Lemma 1 $\mathbf{p}_i v \succeq_M \mathbf{p}_{i+1} v$. Therefore for any word $v \in S$ the non one-element sequence of states $\mathbf{p}_1 v, ..., \mathbf{p}_m v$ forms t-cycle of minimal size m. It is also true for $v = s^i$ for any integer i.

The states $\mathbf{p}_1, \mathbf{p}_1 s, \mathbf{p}_1 s^2$ are distinct. Let us notice that in aperiodic finite semigroup for some l holds $s^l \neq s^{l+1} = s^{l+2}$. Therefore there exists such maximal integer $k \leq l$ such that $\mathbf{p}_1 s^k \neq \mathbf{p}_1 s^{k+1} = \mathbf{p}_1 s^{k+2}$ and in the t-cycle $\mathbf{p}_1 s^k$, $\mathbf{p}_2 s^k = \mathbf{p}_1 s^{k+1}$, $\mathbf{p}_3 s^k = \mathbf{p}_1 s^{k+2}$,..., $\mathbf{p}_m s^k$ holds $\mathbf{p}_1 s^k \neq \mathbf{p}_2 s^k = \mathbf{p}_3 s^k$. So the cardinality of the obtained t-cycle is greater than one and less than m. Contradiction.

Corollary 15. *Let M be almost minimal SCC of aperiodic DFA with transition graph Γ. Then the relation $\succ_M$ is anti-reflexive.*

Theorem 3. *Let the transition graph Γ of an n-state ($n > 2$) synchronizable automaton be strongly connected.*

Then the automaton has reset word of length not greater than $(n-1)^2/2$.

Proof follows from Theorem 2 and Lemma 14.

7 The General Case of Aperiodic DFA

Lemma 16. *Let Γ be transition graph of synchronizable n-state ($n > 2$) DFA with transition semigroup without non-trivial subgroups. Suppose that SCC Γ_1 of Γ has no ingoing edges from another SCC and $|\Gamma_1| \leq n-2$. Then the automaton has synchronizing word of length not greater than $(n-1)^2/2$.*

Proof. Let us denote $|\Gamma \setminus \Gamma_1| = r$. So $|\Gamma_1| = n - r$ and $r > 1$. By [9] (theorem 6.1), a word of length $(n-r)(n-r+1)/2$ maps Γ in $\Gamma \setminus \Gamma_1$. If $r = 2$ then $\Gamma \setminus \Gamma_1$ has reset word of length 1 and Γ has reset word of length $1 + (n-2)(n-1)/2 = (n-1)^2/2 + (3-n)/2 \leq (n-1)^2/2$ because $n > 2$. In the case $r > 2$ one can

assume by induction that the graph $\Gamma \setminus \Gamma_1$ has reset word of length $(r-1)^2/2$. Therefore Γ has reset word v of length $(n-r)(n-r+1)/2+(r-1)^2/2$. Now from $n > r > 1$ and equality
$(n-r)(n-r+1)+(r-1)^2 = (n-r)^2+n-r+(r-1)^2+2(n-r)(r-1)-2(n-r)(r-1)$
$= (n-r+r-1)^2 + n - r - 2(n-r)(r-1) = (n-1)^2 - (n-r)(2r-3)$
follows that the length of v is not greater than $(n-1)^2/2$.

Lemma 17. *Let Γ be transition graph of synchronizable n-state ($n > 2$) DFA with transition semigroup without non-trivial subgroups. Suppose that Γ is a union of SCC Γ_1 of size $n-1$ and sink $\mathbf{p}$. Then the automaton has synchronizing word of length not greater than $(n-1)^2/2$.*

Proof. Let us denote $C_i = i(i-1)/2$. Γ has only two SCC, Γ_1 and $\{\mathbf{p}\}$. For any state $\mathbf{t} \in \Gamma_1$ there exists a word $u(t)$ of minimal length such that $\mathbf{t}u(t) = \mathbf{p}$. If we form a reset word $s = s_1...s_{n-1}$ such that $s_i = u(t)$ for $\mathbf{t} \in \Gamma_1 s_1...s_{i-1}$ with minimal $u(t)$ (as in [9]) then $|s| \leq n(n-1)/2 = C_n$. Our aim is to reduce the length $|s|$ to $(n-1)^2/2$.

For any letter $\sigma \in \Sigma$ there exists a minimal integer k such that $\Gamma\sigma^k = \Gamma\sigma^{k+1}$. Then $|\Gamma\sigma^k| \geq n-k$. Let us form reset word $s = \sigma^k s_{k+1}...s_{n-1}$, the subword s_i is chosen as above, whence $|s_i| \leq i$. Then $|s| \leq k + \sum_{i=k+1}^{n-1} i = k + \sum_{i=1}^{n-1} i - \sum_{i=1}^{k} i = k + n(n-1)/2 - k(k+1)/2 = C_n - C_k$. It is not greater than $(n-1)^2/2$ if $k(k-1)/2 \geq (n-1)/2$. So let us for any letter from Σ study only the case

$$k(k-1) < n-1$$

For any state $\mathbf{q}$ there exists a letter σ such that $\mathbf{q} \neq \mathbf{q}\sigma$. Then the word σ^i of length $i \leq k$ unites any pair of states $\mathbf{q}$ and $\mathbf{q}\sigma^j$.

Let us form now a reset word $s_1...s_{n-1}$ and suppose $S_i = s_1...s_i$. For letter σ with maximal value of k suppose $s_i = \sigma$ for $i \leq k$. So $|S_k| = k$. Let us go to the values of i after k. If in $\Gamma_1 S_i$ either there exists a state $\mathbf{t}$ with $u(t) \leq i$ or two states $\mathbf{q}$ and $\mathbf{q}\sigma^j$ for some σ then let us take as s_{i+1} such $u(t)$ or σ^k. From $k \leq i$ and $u(t) \leq i$ follows $|s_{i+1}| \leq i$. Therefore $|S_{i+1}| \leq k + \sum_{j=k+1}^{i-1} j = k + \sum_{j=1}^{i-1} j - \sum_{j=1}^{k} j = k + i(i-1)/2 - k(k+1)/2 \leq C_{i+1} - C_k$.

The size of $\Gamma_1 S_i$ is at most $n-1-i$. If $|\Gamma_1 S_i| < n-1-i$ then let s_{i+1} be empty word. So $|\Gamma_1 S_{i+1}| \leq n-1-(i+1)$ and $|S_{i+1}| < C_{i+1} - C_k$. Thus anyway

$$|S_i| \leq C_i - C_k \text{ for } i > k$$

Now remains only the case of $\Gamma_1 S_i$ of size $n-1-i$ without pairs of states $\mathbf{q}$ and $\mathbf{q}\sigma^{\mathbf{j}}$ for some σ and with $u(t) > i$ for all its states $\mathbf{t}$. Γ_1 has at most $n-1-i$ states with $u(t) > i$. Consequently the states $\mathbf{t}$ with $u(t) > i$ form the set $\Gamma_1 S_i$, whence from $\mathbf{t}\sigma \neq \mathbf{t}$ follows $\mathbf{t}\sigma \notin \Gamma_1 S_i$ and $u(t)\sigma \leq i$. Therefore $u(t) \leq i+1$ for all $\mathbf{t} \in \Gamma_1 S_i$ and the maximal value of $u(t)$ in Γ_1 is $i+1$. So $n-1-i$ states of $\Gamma_1 S_i$ can be mapped in $\mathbf{p}$ by word v of length at most $(n-1-i)(i+1)$.
Therefore $S_i v$ is a reset word and $|S_i v| \leq (n-1-i)(i+1) + C_i - C_k =$
$(n-1-i)(i+1) + i(i-1)/2 - C_k = (n-1)(i+1) - i^2 - i + i^2/2 - 0.5i - C_k =$
$(n-1)(i+1) - 0.5i^2 - i - 0.5 + 0.5 - 0.5i - C_k =$
$(n-1)^2/2 - (n-1)^2/2 + (n-1)(i+1) - (i+1)^2/2 + 0.5 - 0.5i - C_k =$
$(n-1)^2/2 - (n-2-i)^2/2 - 0.5(i-1) - C_k \leq (n-1)^2/2.$

Theorem 4. *Synchronizable n-state DFA ($n > 2$) with transition semigroup having only trivial subgroups has synchronizing word of length not greater than* $(n-1)^2/2$.

Proof. Let the transition graph Γ of the automaton have SCC C of cardinality r with sink. By Theorem 3 for $\Gamma = C$ the assertion of the theorem is true. So let Γ have several SCC. In the case Γ has more than two SCC or $r > 1$, a synchronizing word of length not greater than $(n-1)^2/2$ exists by Lemma 16. In the case of two SCC and $r = 1$ the graph Γ by Lemma 17 also has synchronizing word of length not greater than $(n-1)^2/2$. Thus a word of length not greater than $(n-1)^2/2$ synchronizes the automaton.

Corollary 18. *The Černy conjecture holds true for DFA with transition semigroup having only trivial subgroups.*

References

1. J. Černy, Poznamka k homogenym eksperimentom s konechnymi automatami, Math.-Fyz. Čas., 14(1964) 208-215.
2. L.Dubuc, Sur le automates circulaires et la conjecture de Černy, RAIRO Inform. Theor. Appl., no 1-3, 32(1998) 21-34.
3. P. Frankl, An extremal problem for two families of sets, Eur. J. Comb., 3(1982) 125-127.
4. J. Kari, Synchronizing finite automata on Eulerian digraphs. Springer, Lect. Notes in Comp. Sci., 2136(2001), 432-438.
5. A.A. Kljachko, I.K. Rystsov, M.A. Spivak, An extremely combinatorial problem connected with the bound on the length of a recurrent word in an automata. Kybernetika. 2(1987) 16-25.
6. G. Lallement, Semigroups and Combinatorial Applications, Wiley, N.Y., 1979.
7. J.E. Pin, On two combinatorial problems arising from automata theory, Annals of Discrete Mathematics 17(1983), 535-548.
8. I.K. Rystsov, Almost optimal bound on recurrent word length for regular automata. Cybernetics and System An. 31, 5(1995) 669-674.
9. I.K. Rystsov, Reset words for commutative and solvable automata. Theoret. Comput. Sci. 172(1997) 273-279.
10. A. Salomaa, Generation of constants and synchronization of finite automata, J. of Univers. Comput. Sci., 8(2) (2002), 332-347.
11. M.P. Schŭtzenberger, On finite monoids having only trivial subgroups. Inf. control, 8(1965) 190-194.
12. A.N. Trahtman, An efficient algorithm finds noticeable trends and examples concerning the Černy conjecture. Lect. Notes in Comp. Sci., Springer, MFCS 2006, 4162(2006), 789-800.

On the Treewidth and Pathwidth of Biconvex Bipartite Graphs

Sheng-Lung Peng* and Yi-Chuan Yang

Department of Computer Science and Information Engineering
National Dong Hwa University
Hualien 97401, Taiwan
slpeng@mail.ndhu.edu.tw

Abstract. In this paper we explore the biclique structure of a biconvex bipartite graph G. We define two concatenation operators on bicliques of G. According to these operations, we show that G can be decomposed into two chain graphs G_L and G_R, and a bipartite permutation graph G_P. Using this representation, we propose linear-time algorithms for the treewidth and pathwidth problems on biconvex bipartite graphs.

1 Introduction

Let $G = (V, E)$ be a finite, simple, and undirected graph where V and E denote the vertex and edge sets of G, respectively. G is called *bipartite* if V can be partitioned into two sets X and Y such that every edge has its ends in different sets. Equivalently, G is bipartite if and only if it is 2-colorable and is denoted as $G = (X, Y, E)$. A *biclique* is a complete bipartite subgraph. Let $N(v) = \{w \mid (v, w) \in E\}$ and $N[v] = N(v) \cup \{v\}$.

A graph G is a *permutation graph* if there is some pair P, Q of permutations of the vertex set such that there is an edge between vertices u and v if and only if u precedes v in one of P, Q, while v precedes u in the other. *Bipartite permutation graphs* simultaneously have both of the properties of bipartite graphs and permutation graphs [20].

A bipartite graph $G = (X, Y, E)$ is *biconvex* if both X and Y can be ordered so that for every vertex v in $X \cup Y$ neighbors of v occur consecutively in the ordering [1]. We say that X and Y each possess the convexity property, and that $V = X \cup Y$ has a biconvex ordering.

A graph is *chordal* if it does not contain a chordless cycle of length greater than three. A *triangulation* of a graph $G = (V, E)$ is a graph $H = (V, E')$ such that H is a supergraph of G and H is chordal. The *treewidth* of graph G is the minimum k such that there is a triangulation of G with maximum clique size at most $k + 1$ [3]. Given a graph G and an integer k, the treewidth problem asks whether the treewidth of G is at most k. This problem remains NP-complete on cobipartite graphs (complement of bipartite graphs) [2], bipartite graphs [13], and graphs of maximum degree at most 9 [8]. For some special classes

* Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 244–255, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of graphs, it has been shown that it can be computed in polynomial time, such as cographs [7], circular-arc graphs [21], chordal bipartite graphs [15], permutation graphs [5,18], circle graphs [12], d-trapezoid graphs [6], and distance hereditary graphs [9].

A graph $G = (V, E)$ is an *interval graph* if one can put the vertices into one-to-one correspondence with a set of intervals $\{I_u\}_{u \in V}$ on the real line, such that $(u, v) \in E$ if and only if $I_u \cap I_v \neq \emptyset$. The *pathwidth* of any graph G is the minimum k such that G can be embedded into an interval graph with maximum clique size at most $k + 1$. Given a graph G and an integer k, the pathwidth problem asks whether the pathwidth of G is at most k. The pathwidth problem is NP-complete on cobipartite graphs [2], bipartite graphs [13], chordal graphs [10], cocomparability graphs [11], chordal domino graphs [16], bipartite distance-hereditary graphs [14], and planar graphs [19]. The known graph classes for which the pathwidth can be computed in polynomial time are cographs [7], permutation graphs [5,18], d-trapezoid graphs [6], split graphs [10], and graphs with bounded treewidth [4].

In this paper, we show that a bipartite permutation graph can be represented by concatenating a sequence of bicliques. Then we extend this representation to a biconvex bipartite graph. By using this representation we propose linear-time algorithms for the treewidth and pathwidth problems on biconvex bipartite graphs. The best algorithm for solving the treewidth problem on biconvex bipartite graphs runs in $O(m^3)$ time where m is the number of edges in the graph [15]. However, the complexity of the pathwidth problem on biconvex bipartite graphs is unknown. The only known result is that it is NP-complete on chordal bipartite graphs (implied from [14]).

2 Bipartite Permutation Graphs

2.1 Biclique Structure

Let $G = (X, Y, E)$ be a bipartite graph. An ordering of the vertices in X has the *adjacency property* if for each vertex y in Y, vertices of $N(y)$ are consecutive in the ordering of X. An ordering of the vertices in X has the *enclosure property* if for every pair of vertices v, w in Y such that $N(v)$ is a subset of $N(w)$, vertices in $N(w) \backslash N(v)$ occur consecutively in the ordering of X. A *strong ordering* of the vertices of $G = (X, Y, E)$ consists of an ordering of X and an ordering of Y such that for all edges $(x, y), (x', y')$ in E where x, x' are in X and y, y' are in Y, $x < x'$ and $y > y'$ imply (x, y') and (x', y) are in E. The edges (x, y) and (x', y') are called a *crossing pair*. Spinrad *et al.* give the following characterization of bipartite permutation graphs.

Theorem 1. [20] The following statements are equivalent.

1. $G = (X, Y, E)$ is a bipartite permutation graph.
2. There is a strong ordering of $X \cup Y$.
3. There exists an ordering of X which has the adjacency and enclosure properties.

For a biclique $B = (X, Y, E')$, $E' = \{(x, y) \mid x \in X \text{ and } y \in Y\}$ is completely determined by X and Y. For convenience, in the following, we use $B = (X, Y)$ to denote a biclique and assume that vertices in X (and Y) are numbered in order. Let $first(X, k)$ (respectively, $last(X, k)$) be the first (respectively, last) k vertices of X. For example, assume that $X = \{x_1, x_2, \ldots, x_r\}$. Then $first(X, k) = \{x_1, x_2, \ldots, x_k\}$ and $last(X, k) = \{x_{r-k+1}, x_{r-k+2}, \ldots, x_{r-1}, x_r\}$.

For two bicliques $B_1 = (X_1, Y_1)$ and $B_2 = (X_2, Y_2)$ and two positive integers k and l, we define a concatenation operator $\oplus$ on B_1 and B_2 as follows. $B_1 \oplus_l^k B_2$ is the bipartite graph $G = (X, Y, E)$ with $X = X_1 \cup X_2$ where $last(X_1, k) = first(X_2, k)$, $Y = Y_1 \cup Y_2$ where $last(Y_1, l) = first(Y_2, l)$, and E being the union of edges in B_1 and B_2 and k, l satisfying the following two conditions.

1. $k + l \geq 1$,
2. neither $|X_1| = |X_2| = k$ nor $|Y_1| = |Y_2| = l$.

By Condition 1, G is connected. By Condition 2, it is impossible to have $|X_1| = |X_2| = k$ (or $|Y_1| = |Y_2| = l$). Otherwise, $B_1 \oplus_l^k B_2$ becomes a biclique. Note that B_1 and B_2 are two maximal bicliques of G. For convenience, we say that $last(X_1, k) \cup last(Y_1, l) = first(X_2, k) \cup first(Y_2, l)$ is the intersection of B_1 and B_2. Figure 1 shows an example of $G = B_1 \oplus_3^2 B_2$.

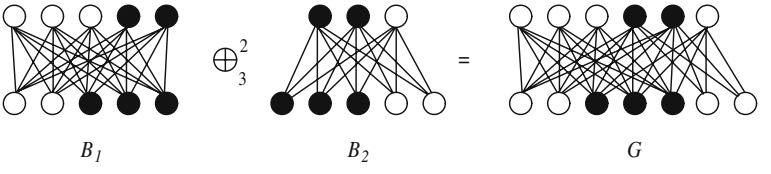


Fig. 1. $G = B_1 \oplus_3^2 B_2$

In the following, if there is no confusion, then $B_1 \oplus_l^k B_2$ is briefed as $B_1 \oplus B_2$. We now define a bipartite graph G_r by concatenating a sequence of bicliques $B_1, B_2, \ldots, B_r$ as follows.

(1) $G_1 = B_1$,
(2) let $G_i = B_1 \oplus B_2 \oplus \cdots \oplus B_i$ be the graph by concatenating $B_1, B_2, \cdots, B_i$, then $G_{i+1} = G_i \oplus B_{i+1}$ where $\oplus$ is applied on B_i and B_{i+1}.

Lemma 1. *G_r is a bipartite permutation graph.*

Proof. We prove this lemma by induction.

(i) G_1 is a biclique which is a bipartite permutation graph.
(ii) Assume that $G_i = B_1 \oplus B_2 \oplus \cdots \oplus B_i$ is a bipartite permutation graph. Now, we consider $G_{i+1} = G_i \oplus_l^k B_{i+1} = B_1 \oplus B_2 \oplus \cdots \oplus B_i \oplus_l^k B_{i+1}$ where $B_i = (X_i, Y_i)$ and $B_{i+1} = (X_{i+1}, Y_{i+1})$. It is not hard to see that the neighborhoods are changed only in the vertices of $B_i \cap B_{i+1}$. By the definition, vertices in $B_i \cap B_{i+1}$ satisfy the adjacency property. It is also not hard to see that they satisfy the enclosure property. According to Theorem 1, G_{i+1} is a bipartite permutation graph.

Thus, by induction, G_r is a bipartite permutation graph. □

Let $G = G_r = B_1 \oplus_{l_1}^{k_1} B_2 \oplus_{l_2}^{k_2} \cdots \oplus_{l_{r-1}}^{k_{r-1}} B_r$. Then we say that $B_1, B_2, \ldots, B_r$ is a *generating base* of G. On the other hand, given a bipartite permutation graph $G = (X, Y, E)$ with a strong ordering, we can obtain a generating base as follows. Assume that $X = \{x_1, x_2, \ldots, x_a\}$ and $Y = \{y_1, \ldots, y_b\}$. Let the largest index in $N(x_1)$ (respectively, $N(y_1)$) be j (respectively, i). Let $X_1 = \{x_1, \ldots, x_i\}$ and $Y_1 = \{y_1, \ldots, y_j\}$. Then $B_1 = (X_1, Y_1)$ is a maximal biclique of G. After B_1 is determined, we delete x_1 (respectively, y_1) and the other vertices whose neighborhood is $N(x_1)$ (respectively, $N(y_1)$). It is not hard to see that the remaining graph is still a bipartite permutation graph and the ordering of vertices is still strong. By repeating this process, we can obtain $B_2, B_3, \ldots, B_r$. Figure 2 shows an example for $G = B_1 \oplus_1^1 B_2 \oplus_2^1 B_3 \oplus_0^1 B_4$.

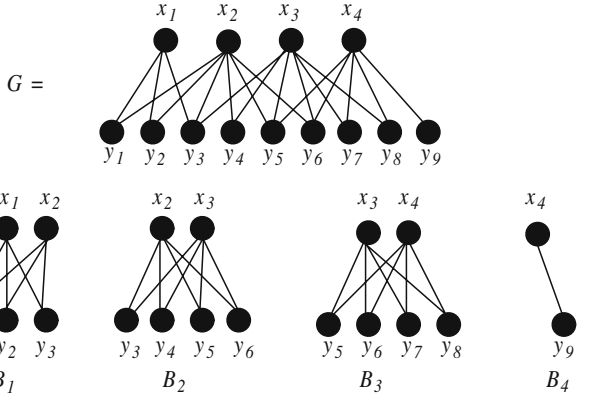


Fig. 2. A generating base of bipartite permutation graph G

Finally, we have the following theorem.

Theorem 2. *G is a connected bipartite permutation graph if and only if there exists a generating base for G.*

A bipartite graph $G = (X, Y, E)$ is a *chain graph* if the neighborhoods of vertices in X form a chain; i.e., there is an ordering $\{x_1, x_2, \ldots, x_{|X|}\}$ on X such that $N(x_1) \supseteq N(x_2) \supseteq \cdots \supseteq N(x_{|X|})$ [22]. It is easy to see that the neighborhoods of the vertices in Y form also a chain. Let $X = \{x_1, \ldots, x_{|X|}\}$ with $N(x_1) \supseteq \cdots \supseteq N(x_{|X|})$ and $Y = \{y_1, \ldots, y_{|Y|}\}$ with $N(y_1) \subseteq \cdots \subseteq N(y_{|Y|})$. Then G has a generating base, i.e., $G = B_1 \oplus_{l_1}^{k_1} B_2 \oplus_{l_2}^{k_2} \cdots \oplus_{l_{r-1}}^{k_{r-1}} B_r$ with $k_i = 0$ for all i and $l_i \geq l_j$ for all $i < j$. Therefore, we have the following corollary.

Corollary 1. *The class of chain graphs is a subclass of bipartite permutation graphs.*

2.2 Computing Treewidth in Linear Time

Given a bipartite permutation graph $G = (X, Y, E)$ with vertices of X and Y in a strong ordering, each crossing of (x, y) and (x', y') is contained in a cycle x, y, x', y', x. That is, each crossing pair represents a 4-cycle in G. By the definition, a graph is chordal if every cycle of length greater than three has a chord. Therefore, to form a chordal graph, every 4-cycle must be destroyed (by adding edges). The set of all the additional edges is called *C_4-destroying set*. A previous investigation by Spinrad *et al.* showed that the union of G and any minimal C_4-destroying set is chordal [20]. For a vertex set W, let $E_W = \{(u, v) \in E \mid u, v \in W\}$. Spinrad *et al.* showed the following lemma.

Lemma 2. [20] *Let E' be a minimal C_4-destroying set of G. If (u, w) is in E'_X (respectively, E'_Y), then for all v in X (respectively, Y) such that $u < v < w$, the edges (u, v) and (v, w) are also in E'_X (respectively, E'_Y).*

By Lemma 2 we have the following lemma.

Lemma 3. *Let $B = (X', Y')$ be a biclique in bipartite permutation graph G, then either X' or Y' must form a clique in any minimal C_4-destroying set of G.*

Intuitively, Lemma 3 tells us that we need to find all the possible maximal bicliques of G for C_4-destroying. In the following, we will see that a generating base of G is enough. Now consider that $G = (X, Y, E)$ is a bipartite permutation graph with a generating base $\{B_1, \ldots, B_r\}$. Assume that $B_i = (X_i, Y_i)$ for all $i \in \{1, \ldots, r\}$ and $G_i = B_1 \oplus \cdots \oplus B_i$. Let $Tri(G_{i-1})$ be any triangulation of G_{i-1}. With respect to $Tri(G_{i-1})$, we define six operations for triangulating G_i as follows:

1. OU_i^1: in the case that Y_{i-1} is a clique in $Tri(G_{i-1})$, vertices in X_i form a clique;
2. OU_i^2: in the case that X_{i-1} is a clique in $Tri(G_{i-1})$, vertices in $X_j \cup X_i$ form a clique where j is the smallest index such that $Y_j \cap Y_i \neq \emptyset$;
3. OU_i^3: in the case that X_{i-1} is a clique in $Tri(G_{i-1})$, vertices in X_i and $Y_{i-1} \cap Y_i$ form respective cliques;
4. OD_i^1: in the case that X_{i-1} is a clique in $Tri(G_{i-1})$, vertices in Y_i form a clique;
5. OD_i^2: in the case that Y_{i-1} is a clique in $Tri(G_{i-1})$, vertices in $Y_j \cup Y_i$ form a clique where j is the smallest index such that $X_j \cap X_i \neq \emptyset$;
6. OD_i^3: in the case that Y_{i-1} is a clique in $Tri(G_{i-1})$, vertices in Y_i and $X_{i-1} \cap X_i$ form respective cliques.

If G_i is triangulated by $Tri(G_{i-1})$ with operation OZ_i^j, we call it Type Z_i^j of triangulation where $Z \in \{U, D\}$ and $j \in \{1, 2, 3\}$. Let $B_{i \cap j}^{i \cup j}$ (respectively, $B_{i \cup j}^{i \cap j}$) be the bipartite graph $B = (X_i \cup X_j, Y_i \cap Y_j)$ (respectively, $B = (X_i \cap X_j, Y_i \cup Y_j)$). Furthermore, in the case of $Y_i \cap Y_j \neq \emptyset$ (respectively, $X_i \cap X_j \neq \emptyset$), B is a biclique.

Lemma 4. *Let $Tri(G_{i-1})$ be an optimal triangulation of G_{i-1}. Then, there is an optimal triangulation of G_i using $Tri(G_{i-1})$ with one of the six operations.*

Proof. By the definition of concatenation operator, the maximal bicliques occurred in G_i but not in G_{i-1} are B_i, $G(X(B_{p\cap i}), Y(B_{p\cup i}))$ for all p with $X_p \cap X_i \neq \emptyset$, and $G(X(B_{q\cup i}), Y(B_{q\cap i}))$ for all q with $Y_q \cap Y_i \neq \emptyset$. For each maximal biclique in a triangulation, at least one partite must become a clique by Lemma 3. Assume that in $Tri(G_{i-1})$, Y_{i-1} forms a clique. Thus we only need to consider the new bicliques occurred after adding B_i. We now show that U_i^1 is a minimal triangulation of G_i. Since $Y_q \cap Y_i \subseteq Y_{i-1}$, all the bicliques with the form $G(X(B_{q\cup i}), Y(B_{q\cap i}))$ are triangulated by the assumption. It is not hard to see that the new maximal biclique B_i is triangulated by the definition, i.e., X_i forms a clique. Similarly, all the bicliques with the form $G(X(B_{p\cup i}), Y(B_{p\cap i}))$ are triangulated since $X_p \cap X_i \subseteq X_i$. Hence type U_i^1 is a triangulation of G_i. The other operations can be showed by a similar way. This completes the proof. □

By the definition, we have the following theorem.

Theorem 3. *For $i \in \{1, \ldots, r\}$, we have the following statements.*

1. $U_i^1 = \min\{D_{i-1}^1 + OU_i^1, D_{i-1}^2 + OU_i^1, D_{i-1}^3 + OU_i^1\}$;
2. $U_i^2 = \min\{U_{i-1}^1 + OU_i^2, U_{i-1}^2 + OU_i^2, U_{i-1}^3 + OU_i^2\}$;
3. $U_i^3 = \min\{U_{i-1}^1 + OU_i^3, U_{i-1}^2 + OU_i^3, U_{i-1}^3 + OU_i^3\}$;
4. $D_i^1 = \min\{U_{i-1}^1 + OD_i^1, U_{i-1}^2 + OD_i^1, U_{i-1}^3 + OD_i^1\}$;
5. $D_i^2 = \min\{D_{i-1}^1 + OD_i^2, D_{i-1}^2 + OD_i^2, D_{i-1}^3 + OD_i^2\}$;
6. $D_i^3 = \min\{D_{i-1}^1 + OD_i^3, D_{i-1}^2 + OD_i^3, D_{i-1}^3 + OD_i^3\}$.

Let $mintri(G_i)$ be the optimal triangulation of G_i. We have the following theorem.

Theorem 4. $mintri(G_i) = \min\{U_i^1, U_i^2, U_i^3, D_i^1, D_i^2, D_i^3\}$.

It is not hard to see that $U_1^1 = U_1^2 = U_1^3$ and $D_1^1 = D_1^2 = D_1^3$. The following is our algorithm for the treewidth problem on bipartite permutation graphs using dynamic programming approach.

Algorithm. Triangulating;
Input: $G = B_1 \oplus_1 \cdots \oplus_{r-1} B_r$ where $r > 1$, $B_i = K_{q_i}^{p_i}$ and $\oplus_i = \oplus_{l_i}^{k_i}$ for all i;
Output: $Opt(G)$ $-$ the treewidth of G;

1. **initialize** $U_1^1, U_1^2, U_1^3, D_1^1, D_1^2, D_1^3$;
2. **for** $i = 2$ **to** r **do begin**
3. compute $U_i^1, U_i^2, U_i^3, D_i^1, D_i^2, D_i^3$;
4. **end**;
5. $Opt(G) = \min\{U_r^1, U_r^2, U_r^3, D_r^1, D_r^2, D_r^3\}$

For each $B_i = (X_i, Y_i)$, let $|X_i| = p_i$ and $|Y_i| = q_i$. Note that for each $\oplus_{l_i}^{k_i}$, $l_i = |Y_i \cap Y_{i+1}|$ and $k_i = |X_i \cap X_{i+1}|$. The detailed variables and formula for our algorithm are as follows.

1. $U_1^1 = U_1^2 = U_1^3 = p_1$;
2. $D_1^1 = D_1^2 = D_1^3 = q_1$;
3. $U_i^1 = \min\{\max\{D_{i-1}^1, p_i + l_{i-1} - 1, q_{i-1} + k_{i-1} - 1\},$
   $\max\{D_{i-1}^2, p_i + l_{i-1} - 1, q_{i-1} + k_{i-1} - 1\},$
   $\max\{D_{i-1}^3, p_i + l_{i-1} - 1, q_{i-1} + k_{i-1} - 1\}\};$
4. $U_i^2 = \min\{\max\{U_{i-1}^1, p_i + \Sigma_{t=j}^{i-1}(p_t - k_t) - 1\}$,
   $\max\{U_{i-1}^2, p_i + \Sigma_{t=j}^{i-1}(p_t - k_t) - 1\}$,
   $\max\{U_{i-1}^3, p_i + \Sigma_{t=j}^{i-1}(p_t - k_t) - 1\}\};$
5. $U_i^3 = \min\{\max\{U_{i-1}^1, p_i + l_{i-1} - 1, p_{i-1} + l_{i-1} - 1\},$
   $\max\{U_{i-1}^2, p_i + l_{i-1} - 1, p_{i-1} + l_{i-1} - 1\},$
   $\max\{U_{i-1}^3, p_i + l_{i-1} - 1, p_{i-1} + l_{i-1} - 1\}\};$
6. $D_i^1 = \min\{\max\{U_{i-1}^1, q_i + k_{i-1} - 1, p_{i-1} + l_{i-1} - 1\},$
   $\max\{U_{i-1}^2, q_i + k_{i-1} - 1, p_{i-1} + l_{i-1} - 1\},$
   $\max\{U_{i-1}^3, q_i + k_{i-1} - 1, p_{i-1} + l_{i-1} - 1\}\};$
7. $D_i^2 = \min\{\max\{D_{i-1}^1, q_i + \Sigma_{t=j}^{i-1}(q_t - l_t) - 1\},$
   $\max\{D_{i-1}^2, q_i + \Sigma_{t=j}^{i-1}(q_t - l_t) - 1\},$
   $\max\{D_{i-1}^3, q_i + \Sigma_{t=j}^{i-1}(q_t - l_t) - 1\}\};$
8. $D_i^3 = \min\{\max\{D_{i-1}^1, q_i + k_{i-1} - 1, q_{i-1} + k_{i-1} - 1\},$
   $\max\{D_{i-1}^2, q_i + k_{i-1} - 1, q_{i-1} + k_{i-1} - 1\},$
   $\max\{D_{i-1}^3, q_i + k_{i-1} - 1, q_{i-1} + k_{i-1} - 1\}\};$

The time complexity of the algorithm depends on the computation of U_i^2 and D_i^2. In the case of U_i^2, $p_i + \Sigma_{t=j}^{i-1}(q_t - k_t) = |X_j \cup X_{j+1} \cup \cdots \cup X_i|$. Since vertices are ordered, the value can be computed by finding the smallest vertex v_a in X_j and the largest vertex v_b in X_i. Then $p_i + \Sigma_{t=j}^{i-1}(q_t - k_t) = b - a + 1$. That is, this computation can be done in constant time. Similarly, D_i^2 can also be computed in constant time. Thus, we have the following theorem.

Theorem 5. *Algorithm* ***Triangulating*** *runs in linear time.*

3 Biconvex Bipartite Graphs

3.1 Biclique Structure

Let $G = (X, Y, E)$ be a biconvex bipartite graph with $X = \{x_1, \ldots, x_p\}$, $Y = \{y_1, \ldots, y_q\}$, and $X \cup Y$ admitting a biconvex ordering. For any two vertices $u, v \in X$ (or $u, v \in Y$), we denote $u < v$ if u is in the left side of v. Likewise, we say $u > v$ if u is right to v. For convenience, we denote $u \leq v$ if $u < v$ or $u = v$, and $u \geq v$ if $u > v$ or $u = v$. Vertices in $X \cup Y$ admit *straight ordering* (*S-ordering* for short) if $x_a < x_b$ and $y_c < y_d$, edges (x_a, y_d) and (x_b, y_c) imply at least (x_a, y_c) or (x_b, y_d) is an edge. Abbas and Stewart showed that any biconvex bipartite graph has a biconvex S-ordering [1]. Furthermore, they mentioned that it can be computed in linear time using the results in [17]. In the following, we assume that a biconvex S-ordering is given for G.

In [1], Abbas and Stweart defined y_l, y_r, G_P, G_L, and G_R as follows. Let y_l be the vertex with smallest index in $N(x_1)$ and y_r be the vertex with largest

index in $N(x_p)$. We may assume $y_1 \leq y_l \leq y_r \leq y_q$ in a biconvex ordering of G. Let $G_P = (X_P, Y_P, E_{X_P \cup Y_P})$ where $X_P = X$ and $Y_P = \{y \mid y_l \leq y \leq y_r\}$. Let $G_L = (X_L, Y_L, E_{X_L \cup Y_L})$ where $X_L = \{x \mid x \in N(y_i) \text{ for } 1 \leq i < l\}$ and $Y_L = \{y \mid y_1 \leq y < y_l\}$. Similarly, $G_R = (X_R, Y_R, E_{X_R \cup Y_R})$ where $X_R = \{x \mid x \in N(y_i) \text{ for } r < i \leq q\}$ and $Y_R = \{y \mid y_r < y \leq y_q\}$. For example, please see Figure 3. Abbas and Stweart showed the following lemmas [1] (implied by Theorems 10 and 11 of their paper).

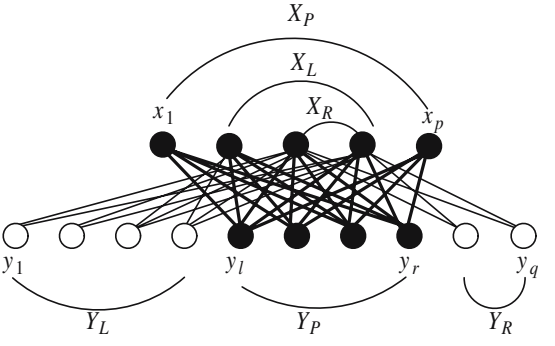


Fig. 3. G_P, G_L, and G_R

Lemma 5 ([1]). *Suppose that $G = (X, Y, E)$ is a connected biconvex bipartite graph. Then, there exists a biconvex S-ordering of $X \cup Y$ such that*

1. *G_P is a connected bipartite permutation graph.*
2. *$N(y_i) \subseteq N(y_j)$, where $y_1 \leq y_i < y_j \leq y_l$ or $y_r \leq y_j < y_i \leq y_q$ for all i, j.*

By the definition of chain graphs, we have the following lemma.

Lemma 6. *Both G_L and G_R are chain graphs.*

By Lemma 5, for a biconvex bipartite graph G with a biconvex S-ordering, there exists a bipartite permutation graph G_P such that G can be decomposed into G_L, G_P, and G_R. By Lemma 6, G_L and G_R are both chain graphs. That is, G_L, G_P, and G_R are all bipartite permutation graphs. Now we assume that $G_P = B_1 \oplus \cdots \oplus B_p$ and $B_i = (X_i, Y_i)$ is a biclique for $i \in \{1, \ldots, p\}$.

Lemma 7. *$G_L \cap G_P \subseteq G(X_1)$ and $G_P \cap G_R \subseteq G(X_p)$.*

Proof. The proof follows the property of biconvex S-ordering. □

By Lemma 7, we define the operator $\otimes$ concatenating G_L (respectively, G_R) and B_1 (respectively, B_p) as follows. Let $G_L^+ = G_L \otimes B_1$ by identifying vertices in X_L with vertices in X_1 such that $X_L (\subset X_1)$ forms consecutive vertices in X_1 and $Y_L \cap Y_1 = \phi$. For example, Figure 4 shows the biconvex bipartite graph $G_L^+ = G_L \otimes B_1$.

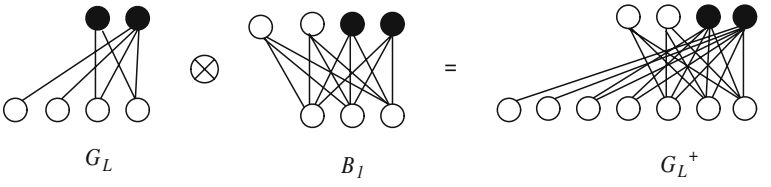


Fig. 4. $G_L^+ = G_L \otimes B_1$

We define $G_R^+ = G_R \otimes B_p$ in a similar way. Note that the strong ordering in a bipartite permutation graph is symmetric, i.e., the reverse of a strong ordering is still a strong ordering. Thus G_R^+ will be treated as G_L^+ if no confusion. By the definition of chain graphs, we have the following lemma.

Lemma 8. *Both G_L^+ and G_R^+ are chain graphs.*

Assume that $G_L = B'_1 \oplus \cdots \oplus B'_l$ and $G_R = B''_1 \oplus \ldots B''_r$. Then we reindex all the bicliques in generating bases of G_L, G_P, and G_R such that $G_L = B_1 \oplus \cdots \oplus B_l\}$, $G_P = B_{l+1} \oplus \cdots \oplus B_{l+p}\}$, and $G_R = B_{l+p+1} \oplus \cdots \oplus B_{l+p+r}$. Likewise, for each B_i, $1 \leq i \leq l+p+r$, let $B_i = (X_i, Y_i)$.

By Lemma 7, G can be represented as the following:

$$\begin{aligned} G &= G_L \otimes G_P \otimes G_R \\ &= B_1 \oplus \cdots \oplus B_l \otimes B_{l+1} \oplus \cdots \oplus B_{l+p} \otimes B_{l+p+1} \oplus \cdots \oplus B_{l+p+r} \end{aligned}$$

Finally, we have the following theorem.

Theorem 6. *G is a biconvex bipartite graph if and only if G can be represented by $B_1 \oplus \cdots \oplus B_l \otimes B_{l+1} \oplus \cdots \oplus B_{l+p} \otimes B_{l+p+1} \oplus \cdots \oplus B_{l+p+r}$ with $B_1 \oplus \cdots \oplus B_l$ and $B_{l+p+1} \oplus \cdots \oplus B_{l+p+r}$ satisfying the property of chain graphs.*

We show an example in Figure 5 that $G = B_1 \oplus_0^2 B_2 \otimes B_3 \oplus_2^2 B_4 \otimes B_5$. For convenience, bicliques $\{B_1, \ldots, B_{l+p+r}\}$ is also called a *generating base* of G.

3.2 Treewidth and Pathwidth

A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (I, F)$ with $C_i \subseteq V$ for all $i \in I$ such that

- $\bigcup_{i \in I} C_i = V$,
- for all edges $(u, v) \in E$, there exists a tree node $i \in I$ such that $u \in C_i$ and $v \in C_i$, and
- for three tree nodes $i, j, k \in I$, if j is on the path from i to k in T, then $C_i \cap C_k \subseteq C_j$.

The *width* of a tree decomposition is $max_{i \in I}|C_i| - 1$. The *treewidth* of G, denoted by $tw(G)$, is the smallest width over all possible tree decompositions. In the case

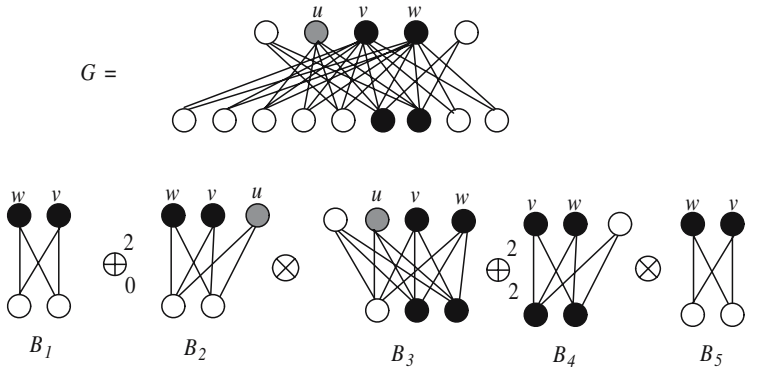


Fig. 5. A generating base of biconvex bipartite graph G

that $T = (I, F)$ is a path, then we call it a *path decomposition* of G. Similarly, the pathwidth of G is defined in a similar way.

Let $G = B_1 \oplus \cdots \oplus B_l \otimes B_{l+1} \oplus \cdots \oplus B_{l+p} \otimes B_{l+p+1} \oplus \cdots \oplus B_{l+p+r}$. Due to the space limitation, we sketch our algorithm as follows. First, using the generating base $B_1, \ldots, B_{l+1}$, we compute $U_{l+1} = \min\{U^1_{l+1}, U^2_{l+1}, U^3_{l+1}\}$ and $D_{l+1} = \min\{D^1_{l+1}, D^2_{l+1}, D^3_{l+1}\}$ using Algorithm Triangulating. Second, by using U_{l+1} and D_{l+1} as initial values, we compute $U_{l+i} = \min\{U^1_{l+i}, U^2_{l+i}, U^3_{l+i}\}$ and $D_{l+i} = \min\{D^1_{l+i}, D^2_{l+i}, D^3_{l+i}\}$ using the same algorithm for i from 2 to p. Finally, by letting U_{l+p} and D_{l+p} be the initial values, the treewidth of G can be computed from B_{l+p+1} to B_{l+p+r} using the same algorithm. It is not hard to see that the correctness thanks to Lemma 3. Thus, we have the following theorem.

Theorem 7. *The treewidth of a biconvex bipartite graph can be computed in linear time.*

It is not hard to see that a tree decomposition of a biconvex bipartite graph can also be constructed by a back tracking after its treewidth is computed. Since the triangulation of a bipartite permutation graph is an interval graph, the tree decomposition of a biconvex bipartite graph looks like the tree depicted in Figure 6.

For simplicity, in a tree decomposition of G, we use A, B, C, D, F to denote the subpaths and L, R to denote the two nodes depicted in Figure 6, respectively. In general, A, B, C, D, F, L, and R are partial path decompositions of the tree decomposition. Let $A \cap B$ denote the vertex set consisting of vertices of G in A and B. Then $ABLC$ (respectively, $ALBC$) is the path decomposition obtained by adding each node of B with $A \cap L$ (respectively, $L \cap C$). We use ABC to denote the path decomposition with minimum width of $ABLC$ and $ALBC$. The role of R is similar. For example, Figure 7 shows the path decomposition of $BACFD$ by assuming that $BALC$ is better than $BLAC$ and $CFRD$ is better than $CRFD$.

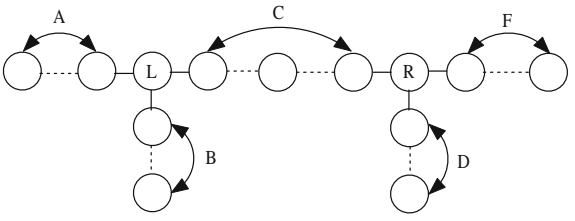


Fig. 6. A general tree decomposition of a biconvex bipartite graph

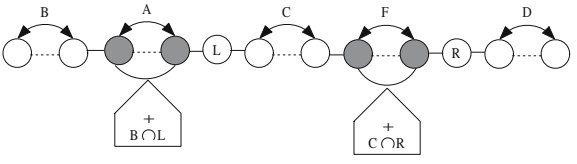


Fig. 7. The path decomposition of $BACFD$

Thus we can use a permutation of A, B, C, D, and F to represent a path decomposition of G. By the symmetry property, we only need to compute $\frac{5!}{2} = 60$ permutations. Due to the limitation of space, we omit the detail. Finally, we have the following theorem.

Theorem 8. *The pathwidth of a biconvex bipartite graph can be computed in linear time.*

References

1. N. Abbas and L. K. Stewart, Biconvex graphs: ordering and algorithms, *Discrete Applied Mathematics* **103** (2000) 1–19.
2. S. Arnborg, D. G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM Journal of Algebraic and Discrete Methods* **8** (1987) 277–284.
3. H. L. Bodlaender, A tourist guide through treewidth, *Acta Cybernetica* **11** (1993) 1–23.
4. H. L. Bodlaender and T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, *Journal of Algorithms* **21** (1996) 358–402.
5. H. L. Bodlaender, T. Kloks, and D. Kratsch, Treewidth and pathwidht of permutation graphs, *SIAM Journal on Discrete Mathematics* **8** (1995) 606–616.
6. H. L. Bodlaender, T. Kloks, D. Kratsch, and H. Müller, Treewidth and minimum fill-in on d-trapezoid graphs, *Journal of Graph Algorithms and Applications* **2** (1998) 1–23.
7. H. L. Bodlaender and R. H. Möhring, The pathwidth and treewidth of cographs, *SIAM Journal on Discrete Mathematics* **6** (1993) 181–188.
8. H. L. Bodlaender and D. M. Thilikos, Treewidth and small separators for graphs with small chordality, *Technical Report UU-CS-1995-02, Department of Computer Science, Utrecht University, Utrecht*, 1995.

9. H. Broersma, E. Dahlhaus, and T. Kloks, A linear time algorithm for minimum fill in and treewidth for distance hereditary graphs, *Scientific program 5th Twente Workshop on Graphs and Combinatorial Optimization*, 48–50, 1997.
10. J. Gustedt, On the pathwidth of chordal graphs, *Discrete Applied Mathematics* **45** (1993) 233–248.
11. M.Habib and R.H. Möhring, Treewidth of cocomparability graphs and a new order-theoretical parameter, *Order* **11** (1994) 47–60.
12. T. Kloks, Treewidth of circle graphs, *ISAAC* 1993, LNCS **762**, 108–117, 1993.
13. T. Kloks, Treewidth–computations and approximations, *Lecture Notes in Computer Science* **842**, Springer-Verlag, Berlin, 1994.
14. T. Kloks, H. Bodlaender, H. Müller, and D. Kratsch, Computing treewidth amd minimum fill-in: all you need are the minimal separators, *ESA* 1993, LNCS **726**, 260–271, 1993. Erratum: *ESA* 1994, LNCS **855**, 508–508, 1994.
15. T. Kloks and D. Kratsch, Treewidth of chordal bipartite graphs, *Journal of Algorithms* **19** (1995) 266–281.
16. T. Kloks, K. Kratsch, and H. Müller, Dominoes, *WG* 1994, LNCS **903**, 106–120, 1995.
17. W. Lipski and Jr., F.P. Preparata, Efficient Algorithms for finding maximum matchings in convex bipartite graphs and related problems, *Acta Informatica* **15** (1981) 329–346.
18. D. Meister, Computing treewidth and minimum fill-in for permutation graphs in linear time, *WG* 2005, LNCS **3787**, 91–102, 2005.
19. B. Monien and I.H. Sudborough, Min cut in NP-complete for edge weighted trees, *Theoretical Computer Science* **58** (1988) 209–229.
20. J. Spinrad, A. Brandstadt, and L. Stewart, Bipartite permutation graphs, *Discrete Applied Mathematics* **18** (1987) 279-292.
21. R. Sundaram, K. S. Singh, and C. P. Rangan, Treewidth of circular arc graphs, *SIAM Journal on Discrete Mathematics* **7** (1994) 647–655.
22. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods* **2** (1981) 77–79.

On Exact Complexity of Subgraph Homeomorphism

Andrzej Lingas and Martin Wahlen

Department of Computer Science, Lund University, 221 00 Lund, Sweden
{Andrzej.Lingas, Martin.Wahlen}@cs.lth.se

Abstract. The *subgraph homeomorphism* problem is to decide whether there is an injective mapping of the vertices of a pattern graph into vertices of a host graph so that the edges of the pattern graph can be mapped into (internally) vertex-disjoint paths in the host graph. The restriction of subgraph homeomorphism where an injective mapping of the vertices of the pattern graph into vertices of the host graph is already given is termed *fixed-vertex subgraph homeomorphism*.

We show that fixed-vertex subgraph homeomorphism for a pattern graph on p vertices and a host graph on n vertices can be solved in time $O(2^{n-p}n^{O(1)})$ or in time $O(3^{n-p}n^6)$ and polynomial space. In effect, we obtain new non-trivial upper time-bounds on the exact complexity of the problem of finding k vertex-disjoint paths and general subgraph homeomorphism.

1 Introduction

Regarded as an injective mapping, the *subgraph isomorphism* of a pattern graph P into a host graph H consists of a mapping of vertices of P into vertices of H so that edges of P map to corresponding edges of G. Generalizations of this mapping include *subgraph homeomorphism*, also termed as *topological embedding* or *topological containment*, where vertices of P map to vertices of H and edges of P map to (internally) vertex-disjoint paths in H, and *minor containment*, where vertices of P map to disjoint connected subgraphs of H and edges of P map to edges of H.

All these problems are inherently NP-complete when the pattern graph P and the guest graph G are not fixed [7]. For fixed P, all are solvable in polynomial time, which in case of subgraph homeomorphism and minor containment is highly non-trivial to show [13]. They remain to be NP-complete for several special graph classes, e.g., for graphs of bounded treewidth [8,12]. Restricting the pattern graph P to complete graphs or simple cycles or paths does not help in the case of subgraph isomorphism. The maximum clique, Hamiltonian cycle and Hamiltonian path problems are well known as basic NP-complete problems [7].

There is an extensive literature on the exact complexity of the maximum clique problem (or, equivalently the maximum independent set problem) and the Hamiltonian cycle or path problem. At present, the best known upper-time

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 256–261, 2007.
© Springer-Verlag Berlin Heidelberg 2007

bounds are respectively $O(2^{0.288n}n^{O(1)})$ [6] and $O(2^n n^{O(1)})$ [9], where n is the number of vertices of the host graph (see also [2,11] for the recent upper time-bounds for the related problem of graph coloring). For the general subgraph isomorphism problem the known upper bound is of the form $O(n^{\lceil p/3\rceil\omega+(p \mod 3)})$ where p is the number of vertices of the pattern graph and ω is the exponent of the fastest matrix multiplication algorithm (cf. [4])[1].

For general subgraph homeomorphism and general minor containment, the authors are not familiar with any non-trivial upper time-bounds on the exact complexity of these problems. For example, Downey et al. mention on page 149 in [5], that subgraph homeomorphism (termed as topological containment) can be solved in time $O(n^{O(m)})$ where n is the number of vertices and m is the number of edges of the host graph. Note that if the host graph is dense then $m = \Omega(n^2)$.

A natural simplification of subgraph homeomorphism is *fixed-vertex subgraph homeomorphism*: Given a $1-1$ map f from the vertices of the pattern graph to those of the host graph H, decide whether or not H contains a homeomorphic image of P in which each vertex of P is identified with its image under f.

Note that fixed-vertex subgraph homeomorphism includes as a sub-problem the important *k-path* problem: Given a graph G and k disjoint pairs (s_i, t_i) of its vertices, decide whether or not G contains k vertex-disjoint paths, one connecting each pair (s_i, t_i).

Similarly fixed-vertex subgraph homeomorphism and k-path problems are solvable for fixed pattern graph or fixed k respectively in polynomial time [13] and no non-trivial upper time-bounds on their exact complexity seem to be known in the literature.

We show that fixed-vertex subgraph homeomorphism for a pattern graph on p vertices and a host graph on n vertices can be solved in time $O(2^{n-p}n^{O(1)})$ or in time $O(3^{n-p}n^6)$ and polynomial space. Consequently, the k-path problem can be solved within the same asymptotic bounds. Furthermore, it follows that subgraph homeomorphism can be solved for a symmetric (i.e., with $p!$ automorphisms) pattern graph in time $O(\binom{n}{p}2^{n-p}n^{O(1)})$ or in time $O(\binom{n}{p}3^{n-p}n^6)$ and polynomial space. I follows also that in the general case subgraph homeomorphism is solvable in time $O(\binom{n}{p}p!2^{n-p}n^{O(1)})$ or in time $O(\binom{n}{p}p!3^{n-p}n^6)$ and polynomial space.

Our solution method for fixed-subgraph isomorphism is based on the use of the principle of exclusion-inclusion to count the number of feasible solutions. This method was originally used by Karp in [10] (rediscovered in [1]) in order to count the number of Hamiltonian cycles using the concept of walks avoiding a subset of the vertex set. We rely on and introduce a generalization of the latter concept to include sets of avoiding walks. We also use the very recent result on fast *subset convolution* by Björklund et al. [3] to reduce the term 3^{n-p} to 2^{n-p} in our upper time-bounds.

In the next section, we introduce the concepts of subset avoiding walks and subset avoiding sets of walks. In Section 3, we present our algorithm for

[1] This upper bound can be marginally improved by using the rectangular matrix multiplication instead of the square one.

fixed-vertex subgraph homeomorphism. In the the following section, we refine the algorithm by a reduction to the subset convolution problem. In Section 5, we derive the upper time-bound for general subgraph homeomorphism.

2 Walks and Sets of Walks

Let $G = (V, E)$ be an undirected graph on n vertices. For a subset $S \subseteq V$, $m \in \{1, ..., n-1\}$, $1 \leq i,\ j \leq n$, let $WALK^m_{i,j}(S)$ be the set of all walks that start in vertex i, end in vertex j, avoid all vertices in S and have total length m.

For a given subset $S \subseteq V$ and $m > 1$, we can compute the cardinalities $|WALK^m_{i,j}(S)|$ from the cardinalities $|WALK^{m-1}_{i',j'}(S)|$ using all the edges of G with both endpoints outside S in time $O(n^3)$ and space $O(n^2)$. Hence, we have the following lemma.

Lemma 1. *For a given subset $S \subseteq V$, all $m \in \{1, ..., n-1\}$ and all $i,\ j \in V$, we can compute the cardinalities $|WALK^m_{i,j}(S)|$ in time $O(n^4)$ and space $O(n^2)$.*

For a subset U of $V \times V$, $m = n-k+|U|$ where k is the set of different vertices in the pairs in U, and a subset $S \subseteq V$, we define $SETWALK^m_U(S)$ as the following family of sets of walks

$$\{\bigcup_{u \in U} \{WALK^{m_u}_u(S)\} | \forall_{u \in U} m_u \in \{1, ..., n-1\} \& \sum_{u \in U} m_u = m\}.$$

Given the cardinalities $|WALK^{m_u}_u(S)|$ for $u \in U$ and $m_u \in \{1, ..., n-1\}$, we can compute the cardinality $|SETWALK^m_U(S)|$ by a straightforward dynamic programming in time $O((n + |U|)^2|U|)$ and space $O((n + |U|)|U|)$. Hence, by Lemma 2, we obtain the following lemma.

Lemma 2. *For a subset U of $V \times V$, $m = n - k + |U|$ where k is the set of different vertices in the pairs in U, and a subset $S \subseteq V$, we can compute the cardinality $|SETWALK^m_U(S)|$ in time $O(n^6)$ and space $O(n^4)$.*

3 An Exact Algorithm for Fixed-Vertex Homeomorphism

To solve the fixed-vertex homeomorphism problem for a pattern graph P on p vertices and a host graph $H = (V, E)$ on n vertices, let us consider all possible choices of the set I of l internal vertices on the paths interconnecting the p vertices in H in one-to-one correspondence with the edges of P ($\binom{n-p}{l}$ ways).

For the given p vertices with their assignments to the vertices of P, and the subset I, define the graph $G = (V_G, E_G)$ as the subgraph of H induced by the union of the p vertices with I. Note that $|V_G| = l + p$. Let U be the set of pairs of the p vertices in one-to-one correspondence with the edges of the pattern graph P.

For $m_l = |V_G| - p + |U|$, i.e., $m_l = l + |U|$, consider the family of sets of walks $SETWALK^{m_l}_U(S)$ for G according to the definition from the previous section.

The next lemma follows from the inclusion exclusion principle.

Lemma 3. *The number of sets of $|U|$ (internally) vertex-disjoint paths interconnecting the p vertices in G in one-to-one correspondence with the edges of P and covering all vertices in G is $|SETWALK_U^{m_l}(\emptyset)| - |\bigcup_{i\in I} SETWALK_U^{m_l}(\{i\})| = \sum_{S\subseteq I}(-1)^{|S|} \times |SETWALK_U^{m_l}(S)|$.*

By combing Lemma 2, 3, we can compute the number of sets of (internally) vertex-disjoint paths interconnecting the p vertices in H in one-to-one correspondence with the edges of P and covering all vertices in G in time $O(2^l n^6)$ and space $O(n^4)$.

Hence, we can solve the fixed subgraph homeomorphism problem for P and H in time $O(\sum_l^{n-p} \binom{n-p}{l} 2^l n^6)$, i.e., in time $O(3^{n-p} n^6)$, and space $O(n^4)$.

Theorem 1. *The fixed-vertex homeomorphism problem for a pattern graph on p vertices and a host graph on n vertices can be solved in time $O(3^{n-p} n^6)$ and space $O(n^4)$.*

Corollary 1. *The k-path problem in a graph graph on n vertices is solvable in time $O(3^{n-2k} n^6)$, and space $O(n^4)$.*

4 A Faster Algorithm for Fixed-Vertex Homeomorphism

By using the recent upper time-bound on the subset convolution by Björklund et al. [3], we can obtain a substantially better upper time-bound for fixed-vertex homeomorphism than that of Theorem 1.

In order to reduce the fixed subgraph homeomorphism problem to a collection of subset convolution problems let us observe that the value of $SETWALK_U^{m_l}(S)$ is really only a function of $I \setminus S$ (more precisely, of the subgraph of H induced by $I \setminus S$ and the p fixed vertices) and l, since $m_l = l + |U|$, U is fixed and the avoided vertices from S in G do not affect it. Therefore, we may denote $SETWALK_U^{m_l}(S)$ by $g_l(I \setminus S)$. Next, let $f(S)$ denote $(-1)^{|S|}$, and let V' denote the set of all vertices in V different from the p fixed vertices. By Lemma 2, we may assume that the values of $f(S)$ and $g_l(S')$ are precomputed for all possible values of l, and all possible subsets S, S' of V' in time $O(2^n n^7)$. It follows by Lemma 3 that it remains to compute for each $I \subseteq V'$, the sum

$$\sum_{S\subseteq I} f(S) g_{|I|}(I \setminus S)$$

Thus, in particular, if we compute for $l = 0, ..., n-p$, the subset convolution $\{h_l(I) | I \subseteq V'\}$, given by

$$h_l(I) = \sum_{S\subseteq I} f(S) g_l(I \setminus S)$$

then we are done. By the recent result from [3], the subset convolution $\{h_l(I) | I \subseteq V'\}$ can be computed by using $O(2^{|V|} \log^{O(1)} |V| \log M)$ additions and multiplications in the integer sum-product ring, where M is an upper bound

on the absolute values of $f(S)$ and $g_l(I \setminus S)$. Hence, since $|V'| = n - p$ and $|SETWALK_U^{m_l}(S)| \leq 2^{n^{O(1)}}$, we obtain the following improved upper time-bound.

Theorem 2. *The fixed-vertex homeomorphism problem for a pattern graph on p vertices and a host graph on n vertices can be solved in time $O(2^{n-p}n^{O(1)})$.*

Corollary 2. *The k-path problem in a graph graph on n vertices is solvable in time $O(2^{n-2k}n^{O(1)})$.*

5 Exact Algorithms for Subgraph Homeomorphism

To reduce the subgraph homeomorphism problem for a pattern graph P on p vertices and a host graph H on n vertices, let us consider all possible choices of p vertices in H ($\binom{n}{p}$ ways) and all possible one-to-one assignments of these p vertices to the vertices of P ($p!$ ways). Note that if the pattern graph is symmetric, i.e., it has $p!$ automorphisms, then we can skip the assignments. Hence, we obtain the following theorem by Theorems 1, 2.

Theorem 3. *The subgraph homeomorphism for a pattern graph on p vertices and a host graph on n vertices is solvable for a symmetric pattern graph in time $O(\binom{n}{p}2^{n-p}n^{O(1)})$ or in time $O(\binom{n}{p}3^{n-p}n^6)$ and space $O(n^4)$. In general, this problem is solvable in time $O(\binom{n}{p}p!2^{n-p}n^{O(1)})$ or in time $O(\binom{n}{p}p!3^{n-p}n^6)$ and space $O(n^4)$.*

6 Final Remarks

Our upper time-bounds for fixed subgraph homeomorphism, and consequently, for subgraph homeomorphism, hold for the decision versions of these problems. To obtain the corresponding embedding in the host graph it is sufficient in the case of fixed subgraph homeomorphism to prune the host graph to a minimal subgraph satisfying the fixed subgraph homeomorphism test. The pruning can be done by repeatedly removing edges and isolated vertices and thus it adds only a multiplicative polynomial factor to the aforementioned upper time-bounds.

It is an interesting question whether or not one could use the inclusion-exclusion principle to derive similar upper time-bounds for the minor containment problem. The underlying interconnection structure in case of minor containment is a set of trees while in case of subgraph homeomorphism just a set of simple paths. This difference can make harder counting solutions to the minor containment problem.

Acknowledgments

The authors are grateful to Fedor Fomin for inspiration, and to Andreas Björklund and Thore Husfeldt for informing about their recent results.

References

1. E.T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. Information Processing Letters 47(4), pp. 203-207, 1993.
2. A. Björklund and T. Husfeldt. *Inclusion-exclusion algorithms for set partitions.* Proc. 47th IEEE Symposium on Foundations of Computer Science, Berkeley 2006, pp. 575-582.
3. A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto. *Fourier meets Möbius: Fast Subset Convolution.* arXiv, 2006, to appear in proc. STOC'07.
4. A. Czumaj and A. Lingas. *Finding a heaviest triangle is not harder than matrix multiplication.* Proc. of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA '07), 2007.
5. R.G. Downey and M.R. Fellows. Parametrized Complexity. Springer, 1999, New York.
6. F.V. Fomin, F. Grandoni and D. Kratch. Measure and conquer: A Simple $2^{0.228n}$ Independent Set Algorithm. Proceedings SODA 2006.
7. M.R. Garey and D.S. Johnson. Computers and Intractability. A Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York 2003.
8. A. Gupta and N. Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. Theoretical Computer Science, 164, pp. 287-298, 1996.
9. M. Held and R. Karp. A dynamic programming approach to sequencing problems. Journal of SIAM 10, pp. 196-210.
10. R.M. Karp. Dynamic programming meets the principle of inclusion and exclusion. Operations Research Letters 1(2), pp. 49-51, 1982.
11. M. Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion exclusion. Proc. 47th IEEE Symposium on Foundations of Computer Science, Berkeley 2006, pp. 582-590.
12. J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. Discrete Mathematics, 108, pp. 343-364, 1992.
13. N. Robertson and P.D. Seymour. Graph minors. XIII. The disjoint paths problem. J. Combinatorial Theory, Ser. B., Vol. 63 (1995) pp. 65-110.

Searching a Polygonal Region by Two Guards

Xuehou Tan

Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@wing.ncc.u-tokai.ac.jp

Abstract. We study the problem of searching for a mobile intruder in a polygonal region P by two guards. The objective is to decide whether there exists a *search schedule* for two guards to detect the intruder, no matter how fast he moves, and if so, generate a search schedule. During the search, two guards are required to walk on the boundary of P continuously and be mutually visible all the time. We present a characterization of the class of polygons searchable by two guards, and give an optimal $O(n)$ time algorithm for determining the two-guard searchability of a polygon and an algorithm for generating a search schedule in time linear in its size.

1 Introduction

Motivated by the relations to the well-known *Art Gallery* and *Watchman Route* problems, much attention has been devoted to the problem of searching for a mobile intruder in a polygonal region P of n vertices by a searcher [2, 5, 6, 7, 8, 9, 10, 11]. Both the searcher and the intruder are modeled by points that can continuously move in P, and the intruder is assumed to be able to move arbitrarily faster than the searcher. The intruder is said to be detected if he is ever within the range of the searcher's vision. The polygon is said to be *searchable* if there is a schedule for the given searcher to detect the intruder.

The visibility of a searcher is usually defined by the flashlights he holds. The *k-searcher* has k flashlights and can see only along the rays of the flashlights emanating from his position. The endpoint of a ray may not be continuous on the polygon boundary, but the 1-searcher should always move on the polygon boundary continuously [5]. If the endpoint of the ray of the 1-searcher is also required to move on the polygon boundary continuously, it introduces a slightly different type of 1-searchers, which is termed as *two guards* [3, 4].

A simple polygon with an *entrance* and an *exit* is called the *corridor*. Icking and Klein were the first to study the corridor search problem, in which two guards start at the entrance and force the intruder out of the region through the exit [4]. They gave $O(n \log n)$ time algorithm for determining whether a corridor is walkable or searchable by two guards. A linear time algorithm was later developed in [3].

Recently, Park et al. considered the problem of searching for an intruder by two guards inside a *room*, which is a polygonal region P with a point *door* d on the boundary of P [7]. In a search schedule, the intruder is kept untouched to d. They gave a characterization of the rooms searchable by two guards, and

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 262–273, 2007.
© Springer-Verlag Berlin Heidelberg 2007

sketched an $O(n \log n)$ time algorithm for the decision problem [7]. Whether the requirement of the given door can further be dropped stands open by now [7]. The main difficulty arises from the fact that the starting point (on the polygon boundary) of any successful search may be touched by the intruder for the second or more time. Therefore, the solutions of the room and/or corridor search problems cannot simply be employed, and it is a challenging work to characterize the class of polygons searchable by two guards.

In this paper, we present a characterization of the polygons searchable by two guards in terms of non-redundant components, thus solving a long-standing problem left open in [7]. Also, we present an optimal $O(n)$ time algorithm for determining the two-guard searchability of a polygon, and an $O(n \log n + m)$ time for generating a search schedule , if it exists, where m ($\leq n^2$) is the number of search instructions reported.

Our first three necessary conditions state that any polygon P is not searchable by two guards if a same type of the component configuration occurs for every boundary point. Two other necessary conditions are a combination of several pairs of disjoint components and one type of the component configuration. If none of these five conditions is true, we can specify two boundary points of P at which our search schedule starts and ends. This helps design a simple search schedule, although two endpoints may be recontaminated during the search. A close study on the structure of non-redundant components further gives critical visibility events occurred in our search schedule. All of these ideas together give a reasonably simple chracterization, although the proof consists of a rather technical case analysis.

2 Preliminary

2.1 Basic Definitions

Let P denote a simple polygon. Two points x, $y \in P$ are said to be mutually *visible* if the line segment connecting them, denoted by $\overline{xy}$, is entirely contained in P. For two regions Q_1, $Q_2 \subseteq P$, we say that Q_1 is *weakly visible* from Q_2 if every point in Q_1 is visible from some point in Q_2.

Let $g_1(t)$ and $g_2(t)$ denote the positions of the guards at a time $t > 0$. A search schedule of two guards consists of the following motion actions; Two guards move along segments of single edges such that (i) no intersections occur among all line segments $\overline{g_1 g_2}$ during the movement, or (ii) any two segments $\overline{g_1 g_2}$ intersect each other [4]. A point $x \in P$ is said to be *detected* or *illuminated* at the time t if x is contained in the segment $\overline{g_1(t)g_2(t)}$. Any region that might contain the intruder at a time (whose position is unknown to the searcher as he is capable of moving arbitrarily fast) is said to be *contaminated*; otherwise it is *clear*. A point is said to be *recontaminated* if it becomes contaminated for the second or more time. The polygon P is said to be *searchable* by two guards if there is a search schedule of two guards that finally clears the polygon P.

For a vertex x of P, let $Succ(x)$ denote the vertex immediately succeeding x clockwise, and $Pred(x)$ the vertex immediately preceding x clockwise. A vertex

of P is *reflex* if its interior angle is strictly greater than 180°. The backward ray shot from a reflex vertex r, denoted by *Backw(r)*, is defined as the first point of P hit by a "bullet" shot at r in the direction from $Succ(r)$ to r, and the forward ray shot *Forw(r)* is the first point hit by the bullet shot at r in the direction from $Pred(r)$ to r. See Fig. 1.

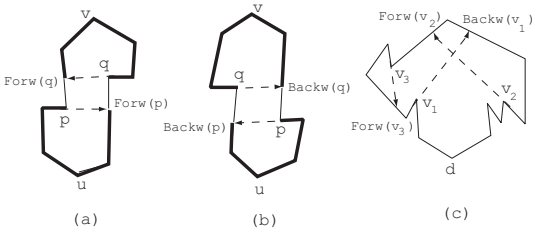


Fig. 1. Illustration of ray shots, components and deadlocks

Let x, y denote two boundary points of P, and $P[x, y]$ (resp. $P(x, y)$) the closed (resp. open) *clockwise* chain of P from x to y. We define the chain $P[r, Backw(r)]$ (resp. $P[Forw(r), r]$) as the *backward component* (resp. *forward component*) of the reflex vertex r. The vertex r is referred to as the *defining vertex* of the component. See Figs. 1(a)-(b) for some examples. A backward (resp. forward) component is said to be *non-redundant* if it does not contain any other backward (resp. forward) component. See Fig. 1(c) for an example, where three components of v_1, v_2 and v_3 are non-redundant.

A pair of reflex vertices v_1, v_2 is said to give a *d-deadlock*, where d is a boundary point of P, if both $P(v_1, Backw(v_1)]$ and $P[Forw(v_2), v_2)$ do not contain d, and the points v_1, $Forw(v_2)$, $Backw(v_1)$ and v_2 are in clockwise order. See Fig. 1(c) for an example. The point d may be identical to v_1 or v_2.

2.2 Two-Guard Searchability of Corridors

Denote by (P, u, v) the corridor with an entrance u and an exit v on the boundary of P, and by L and R the chains $P[u, v]$ and $P[v, u]$, respectively. Note that u and v can be two non-crossing, internal segments of P.

Lemma 1. *[3, 4] A corridor (P, u, v) is walkable by two guards if and only if L and R are mutually weakly visible and neither u-deadlocks nor v-deadlocks occur. It takes $\Theta(n)$ time to test the two-guard walkability of a corridor, and an optimal walk schedule can be reported in time linear in its size.*

The so-called *counter-walks* are also used in our search schedule. A counter-walk asks if there is a walk on P such that two guards move along L and R, one from u to v and the other from v to u.

Lemma 2. *[3, 4] A corridor (P, u, v) allows a counter-walk if and only if L and R are mutually weakly visible and there are no two backward components nor*

two forward components such that they are disjoint, one of them contains u and the other contains v (Figs. 1(a)-(b)). It takes $\Theta(n)$ time to test the counter-walkability of a corridor, and an optimal walk schedule can be reported in time linear in its size.

3 Necessary Conditions

A boundary point d is said to have a *BB-pair* (resp. *FF-pair*) if there are two vertices v_1, v_2 such that two components $P[v_1, Backw(v_1)]$ and $P[v_2, Backw(v_2)]$ (resp. $P[Forw(v_1), v_1]$ and $P[Forw(v_2), v_2]$) are disjoint and both of them do not contain d. See Fig. 2(b) or 2(c) for an example, where all boundary points of the polygon have BB-pairs. The point d is said to have a *BF-pair* if there are two vertices v_1, v_2 such that both components $P[v_1, Backw(v_1)]$ and $P[Forw(v_2), v_2]$, or both components $P[Forw(v_1), v_1]$ and $P[v_2, Backw(v_2)]$, do not contain d. See Fig. 2(a) for an example, where all boundary points of the polygon have BF-pairs. Note that a d-deadlock implies a BF-pair for the point d, and thus two components giving a BF-pair may not be disjoint.

What is important in clearing P is to avoid a 'circle' of recontaminations. A cycle of recontaminations occurs if every boundary point of P has a BF-pair (resp. a BB-pair or an FF-pair). It may occur in two other situations, which are a combination of several pairs of disjoint components and a boundary interval whose points all have BF-pairs.

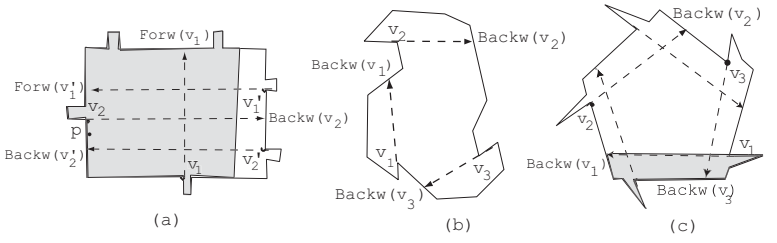


Fig. 2. The polygons satisfying the condition **C1** (a), or **C2** (b)-(c)

Theorem 1. *A simple polygon P is not searchable by two guards if (**C1**) every boundary point p has a BF-pair.*

Proof. Suppose first that for a boundary point p, there are two vertices v_1 and v_2 such that they give a BF-pair for p and the chain $P[v_1, v_2]$ contains p. See Fig. 2(a). In order to clear the vertex $Pred(v_1)$ or $Succ(v_2)$, both guards have to move to v_1 or v_2 once. But, at that moment, the area of the clear region is zero. Thus, any search schedules starting at p are *trivial*.

Assume now that there are two vertices v'_1 and v'_2 such that they give a BF-pair for p, but $P[v'_1, v'_2]$ does not contain the point p. Also, assume that a region containing p has been cleared at a time $t > 0$, but both $Pred(v'_1)$ and $Succ(v'_2)$ are still contaminated. See Fig. 2(a) for an example, where the

shaded region denotes the clear part of P at time t. In order to clear, say, $Pred(v_1')$, the segment connecting two guards has to move to $\overline{v_1' Pred(v_1')}$ by a counter-walk. However, since $P[Forw(v_1'), v_1']$ and $P[v_2', Backw(v_2')]$ are disjoint, such a counter-walk is impossible (Lemma 2). Thus, no search schedules starting at p exist. Since all boundary points of P have BF-pairs, the proof is complete. □

Theorem 2. *A simple polygon P is not searchable if (***C2***) every boundary point p has a BB-pair, or if (***C3***) every boundary point p has an FF-pair.*

Proof. Omitted in this extended abstract. □

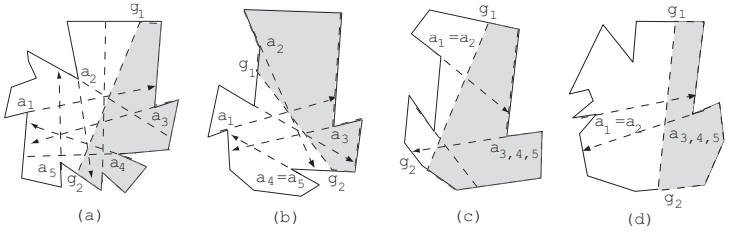


Fig. 3. The polygons satisfying the condition **C4**

Theorem 3. *A simple polygon P is not searchable by two guards if (***C4***) there are five vertices a_i ($1 \le i \le 5$) in clockwise order such that the component $P[a_1, Backw(a_1)]$ is disjoint from $P[a_3, Backw(a_3)]$ and $P[a_4, Backw(a_4)]$, the component $P[a_2, Backw(a_2)]$ is disjoint from $P[a_5, Backw(a_5)]$, and all points of $P[a_2, a_3] \cup P(a_4, a_5)$ have BF-pairs, or if (***C5***) there are five vertices b_i ($1 \le i \le 5$) in clockwise order such that the component $P[Forw(b_5), b_5]$ is disjoint from $P[Forw(b_3), b_3]$ and $P[Forw(b_2), b_2]$, the component $P[Forw(b_4), b_4]$ is disjoint from $P[Forw(b_1), b_1]$, and all points of $P[b_3, b_4] \cup P(b_1, b_2)$ have BF-pairs.*

Proof. Omitted in this extended abstract. □

4 Sufficiency

In this section, we show that the non-existence of **C1** to **C3** helps find a starting point, and the non-existence of **C4** nor **C5** ensures that there is a search schedule for P that starts at the chosen point and ends at some special point.

Theorem 4. *A simple polygon P is searchable by two guards if none of the conditions* **C1** *to* **C5** *applies.*

In the rest of this section, we prove Theorem 4. Note that if a BB-pair and an FF-pair for a boundary points d occur simultanesouly, a BF-pair for d occurs. Our search schedule is then designed to deal with the following situations.

- **Case 1:** there is a boundary point d such that none of BF-pairs, BB-pairs and FF-pairs for d occurs.
- **Case 2** (resp. **Case 3**)**:** there is a boundary point d and two vertices v_1, v_2 such that no BF-pairs for d occur, but v_1 and v_2 give the FF-pair (resp. BB-pair) for d and all points of $P[Forw(v_1), v_1] \cup P[Forw(v_2), v_2]$ (resp. $P[v_1, Backw(v_1)] \cup P[v_2, Backw(v_2)]$) have BF-pairs.

Before giving our search schedule, we need more definitions. Let d be a boundary point of P. We can then order all boundary points clockwise, starting and ending at the point d. For a complete ordering, we consider d as two points $d0$ and $d1$ such that $d0 \leq p \leq d1$ holds for all boundary points p of P. For two boundary points p, p', we say that p *precedes* p' (and p' *succeeds* p) if we encounter p before p' when traversing from $d0$ to $d1$. We write $p < p'$ if p precedes p'. A reflex vertex is said to be *critical* if its non-redundant backward or forward component does not contain d.

For a critical vertex r, we will denote by $P(\overline{rBackw(r)})$ the region bounded by $P[Backw(r), r]$ and $\overline{rBackw(r)}$), or by $P(\overline{rForw(r)})$ the region bounded by $P[r, Forw(r)]$ and $\overline{rForw(r)}$. Thus, the point d is contained in $P(\overline{rBackw(r)})$ or $P(\overline{rForw(r)})$.

4.1 The Search Schedule for Case 1

In this case, there is a boundary point d such that none of BF-pairs, BB-pairs and FF-pairs for d occurs. We will show that the absence of **C4** and **C5** is sufficient for P to be searchable by two guards.

Let $r_1, \ldots, r_j$ (resp. $l_1, \ldots, l_m$) be the clockwise sequence of critical vertices which are defined by their non-redundant backward (resp. forward) components not containing d. Assume that both j and m cannot be zero simultaneously; otherwise, the whole polygon P is visible from the point d and thus P can simply be cleared. Note that $r_j < l_1$ and either $Backw(r_j) > l_1$ or $Forw(l_1) < r_j$ hold; otherwise, r_j and l_1 give the BF-pair for d, a contradiction.

Assume below that $j \geq 1$ and $Forw(l_1) < r_j$ hold (if l_1 exists). It follows from the definition of critical vertices that $r_j < Backw(r_1) < Backw(r_2) \ldots < Backw(r_j)$ holds. From the assumption $Forw(l_1) < r_j$, we also have $Forw(l_1) < Forw(l_2) < \ldots < Forw(l_m) < r_1$; otherwise, a BF-pair for d occurs. Assume also that BF-pairs for r_j occur; otherwise, the polygon P or the corridor (P, d, r_j) can be cleared by a walk from d to r_j. Denote by e the maximum vertex of $P(d0, r_j]$ such that no BF-pairs for e occur, and f the minimum vertex of $P[r_j, d1)$ such that no BF-pairs for f occur. (In the case that $m \geq 1$ and $Backw(r_j) > l_1$ hold, two vertices e and f can be defined analogously.) Note that two vertices e and f exist; otherwise, a BF-pair for d occurs. Since all points of $P[Succ(e), Pred(f)]$ have BF-pairs, two vertices $Succ(e)$ and $Pred(f)$ are reflex. For any $r_k > e$, we have $Backw(r_k) > f$; otherwise, r_k and the vertex of $P[Succ(e), Pred(f)]$ (probably $Pred(f)$) whose backward shot contributes to the BF-pair for the point $Backw(r_k)$ satisfy the condition **C4**.

Our search schedule for Case 1 is designed according to whether there is a reflex vertex u such that $P[Forw(u), u]$ is disjoint from $P[r_h, Backw(r_h)]$ $(1 \leq$

$h \leq j$), or $P[u, Backw(u)]$ is disjoint from $P[r_1, Backw(r_1)]$. The existence of the vertex u helps defense the search against the condition **C4** or **C5**.

Case 1.1. There exists a vertex u such that $P[Forw(u), u]$ is disjoint from some component $P[r_h, Backw(r_h)]$. Assume that u is the smallest vertex such that $P[Forw(u), u]$ is disjoint from $P[r_h, Backw(r_h)]$. Clearly, u is contained in $P(d0, r_h)$, i.e., it is not critical; otherwise, the BF-pair for d occurs. Since two vertices u and r_j give the BF-pair for all points of $P[u, r_j]$ and f is the minimum vertex of $P[r_j, d1)$ such that no BF-pairs for f occur, all points of $P[u, Pred(f)]$ have BF-pairs. This property will help defense the search against **C4** or **C5**.

The basic idea for Case 1.1 is to divide the search into the following three steps and use a greedy algorithm inside each step. In Step 1, the region $P(\overline{r_1 Backw(r_1)})$ is cleared. In Step 2, we clear the region $P(\overline{r_i Backw(r_i)})$ one by one, for $i = 2, 3, \ldots, j$, and terminate the work of clearing $P(\overline{r_i Backw(r_i)})$ as soon as the component $P[r_i, Backw(r_i)]$ containing f is encountered. In Step 3, we clear the whole polygon P. The main task for a walk or a counter-walk is to check weak visibility between two chains. A reflex vertex v of a chain that blocks $Pred(v)$ or $Succ(v)$ from being visible from the other chain makes **C4** or **C5** true, or contradicts with the definitions of the points d, f, and critical vertices r_i. Particularly, v is called the *blocking* vertex.

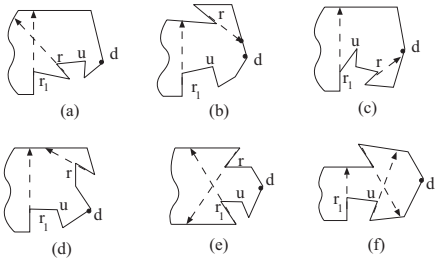


Fig. 4. Illustration for Case 1.1.1.

Case 1.1.1. Clearing the region $P(\overline{r_1 Backw(r_1)})$. Two chains $P[d0, r_1]$ and $P[Backw(r_1), d1]$ are mutually weakly visible, otherwise, there are other critical vertices before r_1 (Fig. 4(a), the FF-pair for d (Fig. 4(b)) or the BF-pair for d (Figs. 4(c)-(e)) occurs. Clearly, no d-deadlocks occur, and no deadlocks occur for r_1 and $Backw(r_1)$ simultaneously; otherwise, **C4** is true (Fig. 4(f)). Hence, $P(\overline{r_1 Backw(r_1)})$ can be cleared using a walk from d to $\overline{r_1 Backw(r_1)}$.

Case 1.1.2. Clearing the region $P(\overline{r_i Backw(r_i)})$, $2 \leq i \leq j$, provided that the vertex f is not contained in $P[r_i, Backw(r_i)]$. Suppose first that the chain $P[r_{i-1}, r_i]$ is weakly visible from $P[Backw(r_{i-1}), Backw(r_i)]$. In this case, the chain $P[Backw(r_{i-1}), Backw(r_i)]$ is also weakly visible from $P[r_{i-1}, r_i]$; otherwise, the blocking vertex and r_{i-1} give the BF-pair for d (if $r_i < Forw(v) < v$ or $v < Backw(v)$, $v \in P[Backw(r_{i-1}), Backw(r_i)]$, holds), or there is a vertex v in $P[Backw(r_{i-1}), Backw(r_i)]$ such that $Backw(v) < r_{i-1}$ holds and thus two vertices r_{i-1} and v ($< f$) satisfy the condition **C4**. See Fig. 5(a) for an example,

where we have assumed that $u < r_{i-1}$. (If the vertex u is contained in $P[r_{i-1}, r_i]$, then $P[r_{i-1}, r_i]$ is not weakly visible from $P[Backw(r_{i-1}), Backw(r_i)]$, which will be discussed later.) There are no two disjoint forward components in between two chains $P[r_{i-1}, r_i]$ and $P[Backw(r_{i-1}), Backw(r_i)]$; otherwise, a d-deadlock occurs. See Fig. 5(b) for an example, where r_{i-1} and q give the d-deadlock. Also, there are no two disjoint backward components in between $P[r_{i-1}, r_i]$ and $P[Backw(r_{i-1}), Backw(r_i)]$, because one defining vertex is r_i, but the other defining vertex cannot be contained in $P[Backw(r_{i-1}), Backw(r_i)]$. See Fig. 5(c). Hence, there is a counter-walk for two guards to move from $\overline{r_{i-1}Backw(r_{i-1})}$ to $\overline{r_i Backw(r_i)}$.

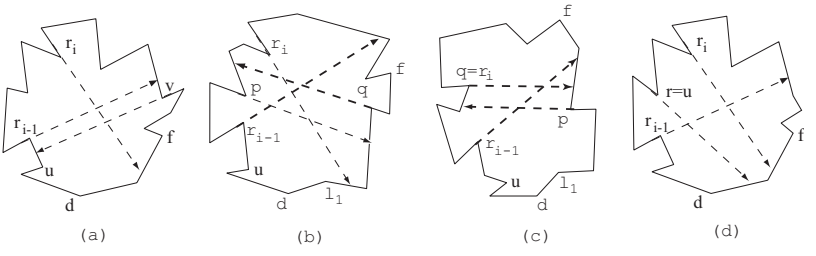


Fig. 5. Illustration for Case 1.1.2.

Consider now the situation in which $P[r_{i-1}, r_i]$ is not weakly visible from $P[Backw(r_{i-1}),\ Backw(r_i)]$. Let r denote the reflex vertex of $P[r_{i-1}, r_i]$ such that $Forw(r) > Backw(r_i)$ holds and $Forw(r)$ is the maximum among these forward shots. Note that $u \leq r$ holds, i.e., r may be identical to u. Since $\overline{Forw(r)} > \overline{Backw(r_i)}$ holds, the segment $\overline{rForw(r)}$ intersects with $\overline{r_{i-1}Backw(r_{i-1})}$, but does not intersect with $\overline{r_i Backw(r_i)}$. See Fig. 5(d) for an example. Because of the maximum of the shot $Forw(r)$, the chain $P[r_{i-1}, r]$ is weakly visible from $P[Backw(r_{i-1}), Forw(r)]$. As discussed above, the line segment connecting two guards can then be moved from $\overline{r_{i-1}Backw(r_{i-1})}$ to $\overline{rForw(r)}$. The chain $P[r, r_i]$ is now weakly visible from $P[Backw(r_i), Forw(r)]$. By a similar argument, we can also show that $P[Backw(r_i), Forw(r)]$ is weakly visible from $P[r, r_i]$. As shown in Case 1.1.1, no deadlocks occur between two chains $P[r, r_i]$ and $P[Backw(r_i), Forw(r)]$. The line segment connecting two guards can then be moved from $\overline{rForw(r)}$ to $\overline{r_i Backw(r_i)}$ using a walk.

Case 1.1.3. Clearing the whole polygon P. Assume that $P(\overline{r_k Backw(r_k)})$ has now been cleared, for some $k \leq j$. The vertex f is visible from r_k; otherwise, there is a vertex v in $P[Backw(r_k), Pred(f)]$ such that two backward components of r_k and v are disjoint and thus r_k and v satisfy **C4** (Fig. 6(a)), or there is a vertex v in $P[f, d1)$ such that r_k as well as d is contained in $P[Forw(v), v]$, and thus two vertices r_k and v give the BF-pair for d (Fig. 6(b)).

Let f' denote the third intersection point of the polygon boundary with the line through r_k and f (Fig. 6(c)). It is not difficult to show that all points of $P[Backw(r_k), f] \cup P[r_k, f']$ are visible from r_k, and thus the segment connecting

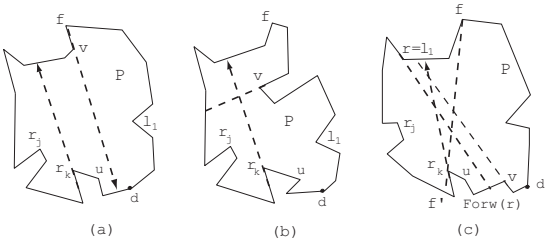


Fig. 6. Illustration for Case 1.1.3.

two guards can simply be rotated from $\overline{r_k Backw(r_k)}$ to $\overline{f'f}$ around r_k. If there are no vertices v in $P[f', f]$ such that $Forw(v) > f$ or $Forw(v) < r_1$ holds, all points of $P[f', f]$ are then visible from f; otherwise, there is a vertex v in $P[f', f]$ such that $r_k < f' < Forw(v) < v$ holds, contradicting $Forw(l_1) < r_1$, or the vertex r_{k+1} is contained in $P[f', f]$ and $Backw(r_{k+1}) < f$ (if $v < Backw(v) < f$, $v \in P[r_k, Backw(r_k)]$, ever holds), contradicting the assumption that f is contained in $P[r_{k+1}, Backw(r_{k+1})]$. Hence, P can be cleared by moving the guard at f' into the vertex f, while keeping the other stand still at f.

Suppose that there are some reflex vertices v in $P[f', f]$ such that $Forw(v) > f$ or $Forw(v) < r_1$ holds. Let r be the vertex such that $Forw(r)$ is closest to r_1 clockwise among all forward ray shots from $P[f', f]$. See Fig. 6(c) for an example. (Note that r may be some critical vertex l_i.) We show below that there is a counter-walk for two guards to move from $\overline{f'f}$ to $\overline{rForw(r)}$. Since $Backw(r_k) < f < Backw(r_{k+1}) < \ldots < Backw(r_j)$ (if $k+1 \leq j$) holds in this case, there are no vertices v in $P[f', r]$ such that $Backw(v) < f$. Thus, the chain $P[f', r]$ is weakly visible from $P[f, Forw(r)]$. Also, the chain $P[f, Forw(r)]$ is weakly visible from $P[f', r]$; otherwise, the blocking vertex v in $P[f, Forw(r)]$ and r_k (resp. r) give the BF-pair for f (resp. d) if the vertex $Succ(v)$ (resp. $Pred(v)$) is invisible from any point of $P[f', r]$, or they satisfy the condition **C5** if $v < Forw(r) < r < Forw(v)$ holds. See Fig. 6(c) for an example, where the blocking vertex v and r satisfy **C5**. There are no disjoint forward components with the defining vertices $v \in P[f, Forw(r)]$ and $v' \in P[f', r]$, which prohibit the required counter-walk; otherwise, two vertices v and r satisfy **C5** (Fig. 6(c)). Also, no disjoint backward components with the defining vertices $v \in P[f, Forw(r)]$ and $v' \in P[f', r]$ exist; otherwise, v and r give the BF-pair for f. Hence, the segment connecting two guards can be moved from $\overline{f'f}$ to $\overline{rForw(r)}$ using a counter-walk. (Note that if r is some critical vertex l_i, the point d is recontaminated during this counter-walk.)

The chain $P[r, f]$ is now weakly visible from $P[f, Forw(r)]$. Also, $P[f, Forw(r)]$ is weakly visible from $P[r, f]$; otherwise, the blocking vertex v and r give a BF-pair for f if $Succ(v)$ is invisible from any point of $P[r, f]$, or the blocking vertex v and r satisfy **C5** if $Pred(v)$ is invisible from any point of $P[r, f]$. Since no deadlocks occur between two chains $P[r, f]$ and $P[f, Forw(r)]$, the polygon P can finally be cleared by a walk from $\overline{rForw(r)}$ to f.

Case 1.2. There exists a vertex u such that $P[u, Backw(u)]$ is disjoint from $P[r_1, Backw(r_1)]$. Note that the vertex u is contained in $P[r_j, d1]$, i.e., u is not a

critical vertex. Let us consider to clear P using the search given in Case 1.1. The search is either complete, or it encounters a reflex vertex r whose backward component is disjoint from that of r_{i-1}, while clearing the region $P(\overline{r_i Backw(r_i)})$ in Case 1.1.2 (Fig. 7(a)). In the latter case, the region $P(\overline{r_i Backw(r_i)})$ cannot be cleared, but we can find a new starting point and start a new search at that point. Observe that all points of $P[r_{i-1}, r]$ cannot have BF-pairs; otherwise, two vertices r_{i-1} and r satisfy the condition **A4**. Without loss of generality, assume that $Succ(r_{i-1})$ or r_{i-1} $(i > 1)$ does not have any BF-pair. Take $Succ(r_{i-1})$ as the new starting point, and denote it by d'. No BB-pairs occur for d'; otherwise, the vertex r_{i-1} is not critical. Also, no FF-pairs occur for d'; otherwise, there is a vertex u' such that the forward component of u' is contained in $P[Backw(r_{i-1}), r_{i-1}]$, i.e., Case 1.1 occurs.

Let us now consider a search schedule that starts at the point d'. The polygon P can be cleared using the search given in Case 1.1; otherwise, one finds five vertices which satisfy the condition **C4** (like a situation shown in Fig. 3(a)). See Fig. 7(a) for an example, where our search schedule ends at the vertex f', which is defined with respect to the starting point d'. (In Fig. 7(a), all points of $P[u, v]$ do not have BF-pairs; otherwise, **C4** is true.)

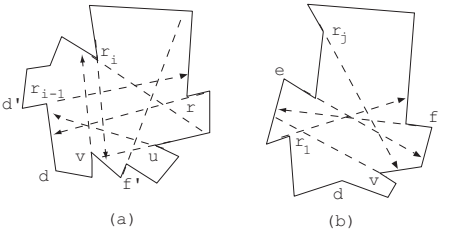


Fig. 7. Illustraions for Case 1.2 and Case 1.3.

Case 1.3. There are no vertices u such that Case 1.1 or Case 1.2 occurs. Note that there are no vertices v in $P[d0, e]$ such that d is contained in $P[Forw(v), v]$; otherwise, two vertices v and r_j give the BF-pair for all points of $P[v, r_j]$, contradicting that e is the maximum vertex of $P(d0, r_j]$ such that no BF-pairs for e occur. Let v denote the vertex such that d is contained in $P[v, Backw(v)]$ and $Backw(v)$ is the smallest among these backward shots. Then, $Backw(v) < Succ(e)$ holds, as v precedes or is identical to the vertex whose forward ray shot contributes to the BF-pair for $Succ(e)$. See Fig. 7(b) for an example. Also, we can show that there is a walk from $Backw(v)$ to f (Fig. 7(b)) or from e to f (if the vertex v does not exist).

4.2 The Search Schedule for Case 2

Let d denote a boundary point without any BF-pairs. Let l_1, ..., l_m denote the clockwise sequence of critical vertices, which are defined by non-redundant forward components not containing d. There are two vertices, say, l_k and $l_{k'}$

$(1 \leq k < k')$ such that two components $P[Forw(l_k), l_k]$ and $P[Forw(l_{k'}), l_{k'}]$ are disjoint and all points of $P[Forw(l_1), l_k]$ have BF-pairs. See Fig. 8 for an example, where $k = 2$ and $k' = 4$. Assume belowe that $Succ(d)$ has the BF-pair, and that the vertex l_k is the largest of the critical vertices satisfying the above condition. Our search schedule is first to clear the regions $P - P(\overline{l_i Forw(l_i)})$, for $i = 1, 2, \ldots, k$, then $P(\overline{l_{k'} Forw(l_{k'})})$ and finally the whole polygon P.

Case 2.1. Clearing the region $P - P(\overline{l_1 Forw(l_1)})$. Let r_1, r_2, ..., r_j be the clockwise sequence of critical vertices, which are defined by non-redundant backward components not containing d, such that $r_i < l_1$ holds, $1 \leq i \leq j$. As in Case 1.1, we first clear the region $P(\overline{r_1 Backw(r_1)})$ using a walk from d to $\overline{r_1 Backw(r_1)}$, and as in Case 1.1.2, we further clear the region $P(\overline{r_j Backw(r_j)})$. See Fig. 8(a) for an example.

Let us now show how the segment connecting two guards is moved from $\overline{r_j Backw(r_j)}$ to $\overline{l_1 Forw(l_1)}$, so as to clear the region $P - P(\overline{l_1 Forw(l_1)})$. The chain $P[r_j, l_1]$ is weakly visible from $P[Backw(r_j), Forw(l_1)]$; otherwise, there are other critical vertices between r_j and l_1. Also, $P[Backw(r_j), Forw(l_1)]$ is weakly visible from $P[r_j, l_1]$; otherwise, the blocking vertex v and r_j give the BF-pair for d if $Pred(v)$ is invisible from any point of $P[r_j, l_1]$, or the blocking vertex v, l_k and $l_{k'}$ satisfy the condition **C5** if $Succ(v)$ is invisible from any point of $P[r_j, l_1]$. Also, there are no two disjoint forward (resp. backward) components in between $P[r_j, l_1]$ and $P[Backw(r_j), Forw(l_1)]$; otherwise, a contradiction with the definition of l_1 or r_j occurs. Hence, there is a counter-walk for two guards to move from $\overline{r_j Backw(r_j)}$ to $\overline{l_1 Forw(l_1)}$. See Fig. 8(b).

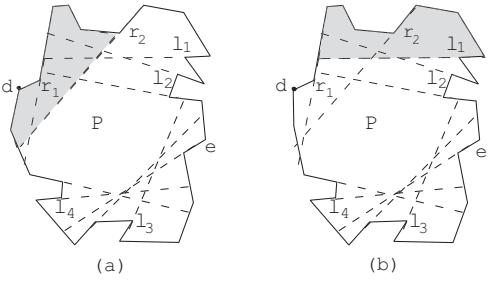


Fig. 8. Illustraion for Case 2

Case 2.2. Clearing the region $P - P(\overline{l_k Forw(l_k)})$. Observe that the inclusion of d in $P(\overline{r_i Backw(r_i)})$ is never employed in Case 1.1.2. By a procedure symmentric to Case 1.1.2, the line segment connecting two guards can be moved from $\overline{l_{i-1} Forw(l_{i-1})}$ to $\overline{l_i Forw(l_i)}$ one by one, so as to clear the region $P - P(\overline{l_i Forw(l_i)})$, $2 \leq i \leq k$.

Case 2.3. Clearing the whole polygon P. The segment connecting two guards has now been moved to $\overline{l_k Forw(l_k)}$, i.e., the region $P - P(\overline{l_k Forw(l_k)})$ is cleared. Recall that l_k is the largest vertex such that $P[Forw(l_k), l_k]$ and $P[Forw(l_{k'}), l_{k'}]$ $(k < k')$ are disjoint and all points of $P[Forw(l_k), l_{k'}]$ do not have BF-pairs.

Furthermore, let $l_{k'}$ be the largest vertex such that $P[Forw(l_k), l_k]$ and $P[Forw(l_{k'}), l_{k'}]$ are disjoint. (See also Fig. 8.) In this case, two chains $P[l_k, Forw(l_{k'})]$ and $P[l_{k'}, Forw(l_k)]$ are mutually weakly visible; otherwise, the blocking vertex contradicts with our assumptions on the vertex l_k or $l_{k'}$, or it together with l_k and $l_{k'}$ makes **C5** true. Since no BF-pair occurs for both endpoints of $\overline{l_k Forw(l_k)}$ nor for both endpoints of $\overline{l_{k'} Forw(l_{k'})}$, the segment connecting two guards can further be moved by a walk from $\overline{l_k Forw(l_k)}$ to $\overline{l_{k'} Forw(l_{k'})}$, so as to clear the region $P(\overline{l_{k'} Forw(l_{k'})})$.

Denote by e the maximum vertex such that $l_k < e$ and no BF-pairs for e occur (Fig. 8). The vertex e exists; otherwise, two vertices l_k and $l_{k'}$ satisfy **C5**. By a procedure symmetric to that for clearing the region $P - P(\overline{l_k Forw(l_k)})$, we can also show that the region $P - P(\overline{l_{k'} Forw(l_{k'})})$ can be cleared, starting at e. The reverse of this procedure then clears the whole polygon P.

The search schedule for Case 3. It can similarly be done as Case 2.

Theorem 5. *It takes $O(n)$ time and space to determine the two-guard searchability of a simple polygon, and $O(n \log n + m)$ time and $O(n)$ space to generate a search schedule of two guards, if it exists, where m ($\leq n^2$) is the number of search instructions reported.*

Proof. Omitted in this extended abstract. □

References

[1] B.K.Bhattacharya, A. Mukhopadhyay and G.Narasimhan, Optimal algorithms for two-guard walkability of simple polygons, *Lect. Notes Comput. Sci.* **2125** (2001) 438-449.

[2] A.Efrat, L.J.Guibas, S. Har-Peled, D.C.Lin, J.S.B. Mitchell and T.M.Murali, Sweeping simple polygons with a chain of guards, In *Proc., ACM-SIAM SODA* (2000) 927-936.

[3] P.J.Heffernan, An optimal algorithm for the two-guard problem, *Int. J. Comput. Geom. & Appl.* **6** (1996) 15-44.

[4] C. Icking and R. Klein, The two guards problem, *Int. J. Comput. Geom. & Appl.* **2** (1992) 257-285.

[5] S.M.LaValle, B.Simov and G.Slutzki, An algorithm for searching a polygonal region with a flashlight, *Int. J. Comput. Geom. & Appl.* **12** (2002) 87-113.

[6] J.H.Lee, S.M.Park and K.Y.Chwa, Searching a polygonal room with one door by a 1-searcher, *Int. J. Comput. Geom. & Appl.* **10** (2000) 201-220.

[7] S.M.Park, J.H.Lee and K.Y.Chwa, Characterization of rooms searchable by two guards, in *Proc. ISAAC 2000*, *Lect. Notes Comput. Sci.* **1969** (2000) 515-526.

[8] I.Suzuki and M.Yamashita, Searching for mobile intruders in a polygonal region, *SIAM J. Comp.* **21** (1992) 863-888.

[9] X.Tan, Searching a simple polygon by a k-searcher, in *Proc. ISAAC 2000*, *Lect. Notes Comput. Sci.* **1969** (2000) 503-514.

[10] X.Tan, A characterization of polygonal regions searchable from the boundary, in *Proc. IJCCGGT 2003*, *Lect. Notes Comput. Sci.* **3330** (2004) 200-215.

[11] X.Tan, Sweeping simple polygons with the minimum number of chain guards, *Inform. Process. Lett.* **102** (2007) 66-71.

On the Internal Steiner Tree Problem

Sun-Yuan Hsieh*, Huang-Ming Gao, and Shih-Cheng Yang

Department of Computer Science and Information Engineering,
National Cheng Kung University,
No.1, University Road, Tainan 70101, Taiwan
hsiehsy@mail.ncku.edu.tw

Abstract. Given a complete graph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}^+$ and a vertex subset $R \subset V$, an *internal Steiner tree* is a Steiner tree which contains all vertices in R such that each vertex in R is restricted to be an internal vertex. The *internal Steiner tree problem* is to find an internal Steiner tree T whose total costs $\sum_{(u,v)\in E(T)} c(u, v)$ is minimum. In this paper, we first show that the internal Steiner tree problem is MAX SNP-hard. We then present an approximation algorithm with approximation ratio $2\rho + 1$ for the problem, where ρ is the best known approximation ratio for the Steiner tree problem.

1 Introduction

Given a graph $G = (V, E)$ with a cost (or distance) function $c : E \to \mathbb{R}^+$, and a vertex subset $R \subseteq V$, a *Steiner tree* is a connected and acyclic subgraph of G which contains all the vertices of R. The vertices in R are *terminals*, and the vertices in $V \setminus R$ are *Steiner* (or *optional*) vertices. Note that a Steiner tree may contain optional vertices. The *cost* of a Steiner tree equals the sum of the costs of all edges in this tree. The *Steiner tree problem* (STP for short) is to find a Steiner tree with the minimum cost in G [3,5,8]. The decision version of this problem has been shown to be NP-complete [9], even in the Euclidean metric [6] or rectilinear metric [7].

There are practical applications in which the terminal vertices are required to be internal vertices in a Steiner tree [3]. For example, in a network resource allocation, the specified servers (terminals) are allowed to act only as transmitters such that in a solution tree, the terminals are restricted to be non-leaf vertices (i.e., internal vertices). Another example is in a sensor network, some nodes might be especially cheap devices that can receive but cannot transmit. In this paper, we study an interesting variant of the Steiner tree problem described as follows. Given a complete graph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}^+$ and a subset $R \subset V$, an *internal Steiner tree* is a Steiner tree which contains all vertices in R such that each vertex in R is restricted to be an internal vertex. The *internal Steiner tree problem* (ISTP for short) is to find an internal Steiner tree T whose cost $\sum_{(u,v)\in E(T)} c(u, v)$ is minimum. For convenience, we call such

* Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 274–283, 2007.
© Springer-Verlag Berlin Heidelberg 2007

an optimal tree as a *minimum internal Steiner tree*. If $|R| = 1$ and $|V| \leq 2$, there is no solution for the problem. If $|R| = 1$ and $|V| \geq 3$, there is a trivial solution by finding the first two smallest cost edges incident to the terminal in G. Therefore, throughout this paper, we assume that $|R| \geq 2$ and $|V \setminus R| \geq 2$.

The currently best-known approximation algorithm for the Steiner tree problem has the approximation ratio $\rho = 1 + \frac{\ln 3}{2} \approx 1.55$ [11]. In this paper, we first show that ISTP is MAX SNP-hard. We then present an $O(n^2 \log n + f(n, m))$-time algorithm with approximation ratio $2\rho + 1$ for ISTP, where $f(n, m)$ is the time complexity of the best-known approximation algorithm for the Steiner tree problem on an input graph with n vertices and m edges.

2 Preliminaries

This paper considers finite, simple, and loopless graphs $G = (V, E)$, where V and E are the vertex and edge sets of G, respectively. We also use the notations $V(G)$ and $E(G)$ to denote the vertex and edge sets of G, respectively. For an edge $e = (u, v)$, u and v are *end-nodes* of e. When u and v are the end-nodes of an edge, they are *adjacent* and are *neighbors*. If vertex v is an end-node of e, then v and e are *incident*. The *degree* of a vertex v in a loopless graph G, denoted by $deg_G(v)$, is the number of incident edges. A *path* of *length* l from a vertex u to a vertex u' in a graph $G = (V, E)$, denoted by $P_G[u, u']$, is a sequence $\langle v_0, v_1, v_2, \ldots, v_l \rangle$ of vertices such that $u = v_0$, $u' = v_l$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, l$. We also call the above path (u, u')-path. The cost of the path is the sum of edge-costs in the path. A path is *simple* if all vertices in the path are distinct. Throughout this paper, we sometimes regard a path as a graph for convenience. A *subgraph* of $G = (V, E)$ is a graph (V', E') such that $V' \subseteq V$ and $E' \subseteq E$. An *induced subgraph* is an edge-preserving subgraph, i.e., (V', E') is an induced subgraph of (V, E) iff $V' \subseteq V$ and $E' = \{(x, y) \in E |\ x, y \in V'\}$.

A *tree* is a connected graph and has no cycles (*acyclic*). A *leaf* of a tree is a vertex of degree 1. A non-leaf vertex is an *internal vertex*. For a tree T, a *subtree* of T is a connected subgraph (V', E') of T, where $V' \subseteq V$ and $E' \subseteq E$. A vertex v in a Steiner tree T is said to be a *leaf-terminal* if v is a terminal and also a leaf in T. Throughout this paper, we assume that the cost function associated with the given graph is metric.

Definition 1. Let Π_1 and Π_2 be two optimization problems, we say that Π_1 *L-reduces to* Π_2 if there are polynomial time algorithms, A_1 and A_2, and positive constants, δ and ω, such that for any instance I_1 of Π_1, the following conditions are satisfied:

(a) Algorithm A_1 produces an instance $A_1(I_1)$ for Π_2 such that $\mathsf{OPT}(A_1(I_1)) \leq \delta \cdot \mathsf{OPT}(I_1)$, where $\mathsf{OPT}(I_1)$ and $\mathsf{OPT}(A_1(I_1))$ represent the costs of optimal solutions of I_1 and $A_1(I_1)$, respectively.
(b) Given any solution of $A_1(I_1)$ with cost $cost_2$, Algorithm A_2 produces a solution of I_1 with cost $cost_1$ in polynomial time such that $|cost_1 - \mathsf{OPT}(I_1)| \leq \omega \cdot |cost_2 - \mathsf{OPT}(A_1(I_1))|$. □

A problem is said to be *MAX SNP-hard* if every MAX SNP problem L-reduces to this problem, but the problem itself may not be MAX SNP. On the other hand, we can show a problem to be MAX SNP-hard by providing an L-reduction from some MAX SNP-hard problem to it.

3 MAX SNP-Hardness Result

For the Steiner tree problem, if we restrict the co-domain of the cost function from $\mathbb{R}^+$ to $\{1, 2\}$, then the restricted Steiner tree problem is called the *(1,2)-Steiner Tree Problem* (STP(1,2) for short).

Lemma 1. [2] *STP(1,2) is MAX SNP-hard.*

We will prove ISTP is MAX SNP-hard by providing an L-reduction from STP (1,2) to ISTP. We present two polynomial-time algorithms, namely A_1 and A_2, shown in Figures 1 and 2, respectively, for the L-reduction. Given a graph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}^+$, define $c(G) = \sum_{e \in E} c(e)$. We can also define $c(G')$ for a subgraph G' of G similarly. For two vertices u and v in G with a cost function c, let $c^*_G(u, v)$ be the minimum cost of a path between u and v in G, i.e., $c^*_G(u, v) = \min_{P_G[u,v]} \{\sum_{e \in P_G[u,v]} c(e)\}$.

Algorithm. $A_1(I_1)$

Input: An instance $I_1 = (G_1, R_1, c_1)$ of STP(1,2), where $G_1 = (V_1, E_1)$ is a complete graph, a vertex subset $R_1 = \{r_1, r_2, \ldots, r_k\}$ with $|R_1| = k \geq 2$, and the cost function $c_1 : E_1 \to \{1, 2\}$.

Output: An instance $I_2 = (G_2, R_2, c_2)$ of ISTP.

1: For each $r_i \in R_1$, create one auxiliary vertex a_i corresponding to r_i. Let $AV = \{a_1, a_2, \ldots, a_k\}$ be the *auxiliary vertex set*.
2: Construct a complete graph $G_2 = (V_2, E_2)$ as follows:
(a) Set $V_2 = V_1 \cup AV$.
(b) Let $AE = \{(a_i, a_j) |\ 1 \leq i, j \leq k \text{ and } i \neq j\} \cup \{(a_i, v) |\ a_i \in AV \text{ and } v \in V_1\}$ be the *auxiliary edge set*, and each edge in AE be an *auxiliary edge*. Set $E_2 = E_1 \cup AE$.
3: Set $R_2 = R_1$.
4: Define the cost function $c_2 : E_2 \to \mathbb{R}^+$ such that for each $e \in E_2$,

$$c_2(e) = \begin{cases} c_1(e) & \text{if } e \in E_1, \\ \frac{1}{2k} & \text{if } e = (a_i, r_i), \text{ where } 1 \leq i \leq k, \\ 2c(G_1) & \text{if } e = (a_j, r_i), \text{ where } i \neq j, \\ c^*_{G_1}(r_i, r_j) + 1 & \text{if } e = (a_i, a_j), \text{ where } i \neq j, \\ c^*_{G_1}(r_i, v) + 1 & \text{if } e = (a_i, v), \text{ where } 1 \leq i \leq k \text{ and } v \in V_1 \setminus R_1. \end{cases}$$

Fig. 1. An algorithm for the input instance transformation from STP(1,2) to ISTP

Lemma 2. $\mathsf{OPT}(A_1(I_1)) \leq 2\mathsf{OPT}(I_1)$.

Proof. Let T_S^* be a minimum Steiner tree of I_1, and let $r_1, r_2, \ldots, r_q$, where $q \leq k$, be the leaf-terminals of T_S^*. By adding q auxiliary edges to T_S^*, we obtain an internal Steiner tree of $A_1(I_1)(= I_2)$ because each terminal is now an internal vertex of the resulting tree. Therefore, $\mathsf{OPT}(A_1(I_1)) \leq \mathsf{OPT}(I_1) + \frac{q}{2k} \leq \mathsf{OPT}(I_1) + \frac{k}{2k} = \mathsf{OPT}(I_1) + \frac{1}{2} < \mathsf{OPT}(I_1) + 1 \leq 2\mathsf{OPT}(I_1)$. □

For a acyclic subgraph T in $A_1(I_1)(= I_2)$, let $AV(T) = AV \cap V(T)$, and let $av(T) = |AV(T)|$.

Suppose that T_1 and T_2 are two connected acyclic subgraphs of a connected graph G such that $V(T_1) \cap V(T_2) = \emptyset$. Let u and u' be two vertices of T_1 and T_2, respectively, and let $S_G[u, u'] = \langle v_0(= u), v_1, \ldots, v_l(= u') \rangle$ be a shortest path[1] between u and u' in G. A *connection of u and u' using $S_G[u, u']$* is to merge T_1 and T_2 into one connected acyclic subgraph, which is obtained by adding the edges of $S_G[u, u']$ to the above two trees and then for each cycle in the resulting graph, deleting an arbitrary tree-edge if it is in the cycle. Due to the space limitation, the proof of the following lemma is omitted.

Lemma 3. *Suppose that T_I^* is a minimum internal Steiner tree of I_2. If we delete each auxiliary vertex from T_I^*, the resulting graph is a minimum Steiner tree of I_1.*

We now present our Algorithm A_2 shown in Figure 2. The following result can be obtained from the algorithm.

Lemma 4. *The following two statements hold:*

1. *Given any internal Steiner tree T_I (feasible solution) of I_2, Algorithm A_2 can generate a Steiner tree (feasible solution) T_S of I_1.*
2. *If $c(T_I) = cost_2$ and $c(T_S) = cost_1$, then $cost_1 \leq cost_2 - \frac{1}{2k} av(T_I)$.*

Proof. It is not difficult to verify that the algorithm can generate a Steiner tree of I_2. Thus we only show that (2) holds. There are the following two possible scenarios.

Case 1. $AV(T_I) = \emptyset$. According to lines 1–2 of Algorithm A_2, T_I will be returned as a feasible solution of I_1. Therefore, $cost_2 = c(T_I) = cost_1$ and the result holds.

Case 2. $AV(T_I) \neq \emptyset$. We further consider the following two subcases.

Case 2.1. T_I contains an auxiliary edge with the cost $2c(G_1)$. According to lines 3–6 of Algorithm A_2, a Steiner tree T' of I_1 is obtained. Note that T' is obtained from T_I by deleting at least one auxiliary edge with cost $2c(G_1)$ together with $av(T_I)$ auxiliary edges in which each has cost $\frac{1}{2k}$. By the above observation and the total costs of those $l - 1$ edges used to merge $T_1, T_2, \ldots, T_l$ into T' is obviously smaller than $2c(G_1)$, we conclude that $cost_1 < cost_2 - \frac{1}{2k} av(T_I)$.

[1] In this paper, a shortest path between two distinct vertices is a minimum cost path between them.

Algorithm. $A_2(T, A_1(I_1))$

Input: An internal Steiner tree T of $A_1(I_1)$, where $I_1 = (G_1, R_1, c_1)$.
Output: A Steiner tree of I_1.

```
 1: if T contains no auxiliary vertex, i.e., AV(T) = ∅ then
 2:     return T
 3: else if there exists any auxiliary edge in T with cost 2c(G_1) then
 4:     Delete all auxiliary vertices in AV(T). After deleting those vertices, let
        T_1, T_2, ..., T_l be the resulting subtrees such that each T_i contains at least one
        terminal, i.e.,V(T_i) ∩ R_1 ≠ ∅ for all 1 ≤ i ≤ l.
 5:     Merge T_1, T_2, ..., T_l into one tree T' using l − 1 edges in E(G_1) \ (⋃_{i=1}^{l} E(T_i))
        in which the total cost of these l − 1 edges is as smallest as possible.
 6:     return T'
 7: else
 8:     for every auxiliary vertex a_i in T do
 9:         for every auxiliary edge e in T incident to a_i do
10:             if e = (a_i, r_i) then        ▷ a_i is the auxiliary vertex corresponding to r_i
11:                 Delete e from T
12:             else if e = (a_i, a_j) then                ▷ a_j is another auxiliary vertex
13:                 Delete e from T
14:                 if r_i and r_j are disconnected after deleting e then
15:                     Connect r_i and r_j using a shortest path S_{G_1}[r_i, r_j]
16:             else                                  ▷ e = (a_i, v), where v ∈ V(G_1) \ R_1
17:                 Delete e from T
18:                 if r_i and v are disconnected after deleting e then
19:                     Connect r_i and v using a shortest path S_{G_1}[r_i, v]
20:     Delete all auxiliary vertices in AV(T)
21:     return the resulting tree
```

Fig. 2. An algorithm that generates a feasible solution of I_1 from a feasible solution of I_2

Case 2.2. T_I contains no auxiliary edge with cost $2c(G_1)$. We prove this case by induction on the number of auxiliary vertices in $AV(T_I)$. If T_I contains only one auxiliary vertex, a_i, then Algorithm A_2 will delete all auxiliary edges incident to a_i and then delete a_i. In deleting each auxiliary edge, if the resulting graph is composed by two subtrees, then a shortest path between some two terminals, one in each subtree, is used to merge the two subtrees into one tree. According to lines 7–21 of Algorithm A_2, it is not difficult to check that after deleting a_i, the cost of the resulting tree is reduced from the original one by at least $\frac{1}{2k}$. Thus, $cost_1 \leq c(T_I) - \frac{1}{2k} = cost_2 - \frac{1}{2k}$ and the base case holds. Now consider $av(T_I) = t > 1$. Without loss of generality, assume that a_1 is the first auxiliary vertex selected by Algorithm A_2. Again, Algorithm A_2 will delete all auxiliary edges incident to a_1 and then delete it. Let T' be the resulting tree after deleting a_1. Using an argument similar

to show the base case, we have $c(T') + \frac{1}{2k} \leq c(T_I) = cost_2$. Since the number of auxiliary vertices in $AV(T')$ is now $t-1$, by the induction hypothesis, we have a Steiner tree T'' of I_1 such that $cost_1 = c(T'') \leq c(T') - \frac{1}{2k}av(T')$. By the above equations, we have that $cost_1 = c(T'') \leq c(T') - \frac{1}{2k}av(T') \leq c(T_I) - \frac{1}{2k} - \frac{1}{2k}av(T') = c(T_I) - (\frac{1}{2k} + \frac{1}{2k}av(T')) = cost_2 - \frac{1}{2k}(av(T')+1) = cost_2 - \frac{1}{2k}av(T_I)$. Thus, $cost_1 \leq cost_2 - \frac{1}{2k}av(T_I)$.

Combining Case 1 and Case 2, the result holds. □

Theorem 1. *ISTP is MAX SNP-hard.*

Proof. We prove the theorem by providing an L-reduction from a known SNP-hard problem STP(1,2) to ISTP. We have presented two polynomial-time algorithms A_1 and A_2. Next, we need find out two positive constants δ and ω to satisfies the two conditions of Definition 1. By Lemma 2, we have that $\mathsf{OPT}(A_1(I_1)) \leq 2\mathsf{OPT}(I_1)$. By Lemma 4, given any feasible solution, T_I, of $A_1(I_1)$ with $c(T_I) = cost_2$, we can get a feasible solution, T_S, of I_1 with $c(T_S) = cost_1$ such that $cost_1 \leq cost_2 - \frac{1}{2k}av(T_I)$.

If $cost_2 = \mathsf{OPT}(A_1(I_1))$, i.e., the given feasible solution is a minimum internal Steiner tree, T_I^*, of $A_1(I_1)$, then Algorithm A_2 will delete all auxiliary vertices of T_I^* and return a Steiner tree of I_1. According to Lemma 3, this Steiner tree is also a minimum Steiner tree, T_S^*, of I_1, i.e., $cost_1 = \mathsf{OPT}(I_1)$. In this special case, we have $|cost_1 - \mathsf{OPT}(I_1)| = 0 = |cost_2 - \mathsf{OPT}(A_1(I_1))|$ and

$$\mathsf{OPT}(I_1) = \mathsf{OPT}(A_1(I_1)) - \frac{1}{2k}av(T_I^*) \tag{1}$$

Therefore, $|cost_1 - \mathsf{OPT}(I_1)| \leq |cost_2 - \frac{1}{2k}av(T_I) - \mathsf{OPT}(I_1)|$ (by Lemma 4) $\leq |cost_2 - \frac{1}{2k}av(T_I) - (\mathsf{OPT}(A_1(I_1)) - \frac{1}{2k}av(T_I^*))|$(by Eq.1) $= |cost_2 - \mathsf{OPT}(A_1(I_1)) + \frac{1}{2k}av(T_I^*) - \frac{1}{2k}av(T_I)| \leq |cost_2 - \mathsf{OPT}(A_1(I_1))| + |\frac{1}{2k}av(T_I^*) - \frac{1}{2k}av(T_I)| = |cost_2 - \mathsf{OPT}(A_1(I_1))| + |\frac{1}{2k}av(T_I) - \frac{1}{2k}av(T_I^*)| \leq |cost_2 - \mathsf{OPT}(A_1(I_1))| + |cost_2 - cost_1 - \frac{1}{2k}av(T_I^*)|$ (by Lemma 4) $= |cost_2 - \mathsf{OPT}(A_1(I_1))| + |cost_2 - cost_1 - (\mathsf{OPT}(A_1(I_1)) - \mathsf{OPT}(I_1))|$ (by Eq.1) $= |cost_2 - \mathsf{OPT}(A_1(I_1))| + |cost_2 - \mathsf{OPT}(A_1(I_1)) + (\mathsf{OPT}(I_1) - cost_1)| \leq |cost_2 - \mathsf{OPT}(A_1(I_1))| + |cost_2 - \mathsf{OPT}(A_1(I_1))|$ (by $\mathsf{OPT}(I_1) \leq cost_1$) $= 2|cost_2 - \mathsf{OPT}(A_1(I_1))|$. Then, both δ and ω can be set to 2 and thus ISTP is MAX SNP-hard. □

4 An Approximation Algorithm

In this section, we present an approximation algorithm, namely A_{ISTP}, for ISTP. Let A_{STP} denote the best-known approximation algorithm for STP with approximation ratio $\rho = 1 + \frac{\ln 3}{2} \approx 1.55$ [11], and also let $S_A = (V_A, E_A)$ be the Steiner tree returned by A_{STP}. To make sure that the solution of ISTP exists, in what follows, we assume that $|V(G) \setminus R| \geq 2$. The concept of our algorithm is first to apply A_{STP} to obtain a Steiner tree $S_A = (V_A, E_A)$, and then transform it to an internal Steiner tree.

Let T be a Steiner tree of the instance $I = (G, R, c)$ of the problem ISTP. The following property is useful to our algorithm for transforming T to an internal Steiner tree. It is not difficult to show the following result.

Lemma 5. *Let T be a Steiner tree of the instance $I = (G, R, c)$ of the problem ISTP such that $|V(T) \setminus R| \geq 2$. If v is a leaf-terminal of T, then there is an internal vertex $\alpha_v \in V(T)$ such that one of the following two conditions hold: (1) $\deg_T(\alpha_v) = 2$ and $\alpha_v \notin R$; (2) $\deg_T(\alpha_v) \geq 3$.*

We next present our approximation algorithm. We call the two vertices α_v and β_v selected by Algorithm A_{ISTP} for each leaf-terminal v as the *critical vertex* and the *target vertex* of v, respectively.

ALGORITHM A_{ISTP} (approximation algorithm for the internal Steiner tree problem)
Input: A complete graph $G = (V, E)$ with a metric cost function $c : E \to \mathbb{R}^+$ and a proper subset $R \subset V$ of terminals such that $|V \setminus R| \geq 2$.
Output: An internal Steiner tree T_G.

Step 1. ▷ Find a Steiner tree in $G = (V, E)$
Use the currently best-known approximation algorithm to find a Steiner tree S_A in G.
Step 2. $S'_A \leftarrow S_A$
Step 3. ▷ Transform S'_A into an internal Steiner tree
if S'_A is **not** an internal Steiner tree **then**
if $|V(S'_A) \setminus R| = 0$ **then**
Find the first two smallest cost edges, (p_1, q_1) and (p_2, q_2), between $(V \setminus V(S'_A))$ and $V(S'_A)$, where $p_1 \neq p_2$, $p_1, p_2 \in V \setminus V(S'_A)$, and $q_1, q_2 \in V(S'_A)$.
Add the two vertices, p_1 and p_2, to $V(S'_A)$ and add the two edges, (p_1, q_1) and (p_2, q_2), to $E(S'_A)$.
else if $|V(S'_A) \setminus R| = 1$ **then**
Find the smallest cost edge, (p, q), between $(V \setminus V(S'_A))$ and $V(S'_A)$, where $p \in V \setminus V(S'_A)$ and $q \in V(S'_A)$. Add the vertex p to $V(S'_A)$ and add the edge (p, q) to $E(S'_A)$.
for each leaf-terminal v in S'_A **do**
▷ Determine the critical vertex α_v and the target vertex β_v
(a) Select the nearest vertex $\alpha_v \in V(S'_A)$ satisfying one of the following condition, based on the depth-first search:
(1) $\deg_{S'_A}(\alpha_v) = 2$ and $\alpha_v \notin R$.
(2) $\deg_{S'_A}(\alpha_v) \geq 3$. ▷ The existence of α_v is ensured by Lemma 5
(b) Choose a vertex $\beta_v \in V(S'_A)$ which is adjacent to α_v, but does not belong to the path $P_{S'_A}[v, \alpha_v]$.
(c) $E(S'_A) \leftarrow E(S'_A) \cup \{(v, \beta_v)\}$
(d) $E(S'_A) \leftarrow E(S'_A) \setminus \{(\alpha_v, \beta_v)\}$
end for
Return $T_I \leftarrow S'_A$

Lemma 6. *Assume that v is a leaf-terminal of the current tree T handled by Algorithm A_{ISTP} in an iteration of the for-loop within Step 3. Then, we have the following observations:*

1. *$deg_T(\alpha_v)$ will be decreased by 1 in the next iteration.*
2. *$deg_T(\beta_v)$ will be unchanged in the next iteration.*
3. *$deg_T(v)$ will be increased by 1, and fixed as 2 until the algorithm terminates.*

Two paths are said to be *edge-disjoint* if both paths have no common edge. A set of paths is said to be *pairwise edge-disjoint* if any two of them are edge-disjoint.

Lemma 7. *Suppose that v_0 is a leaf-terminal of S_A and v_k is the critical vertex of v_0 selected by Algorithm A_{ISTP}, i.e., $\alpha_{v_0} = v_k$. Let $P_{S_A}[v_0, v_k] = \langle v_0, v_1, \ldots, v_k\rangle$ be a path of S_A. Then, during the executing of Algorithm A_{ISTP} before v_0 being handled, there always exists a path $P_T[v_0, v_k]$ in the current tree T, which is extended from $P_{S_A}[v_0, v_k]$ such that the following three properties hold:*

1. *The length of $P_T[v_0, v_k]$ is at least k.*
2. *The path $P_T[v_0, v_k]$ contains all the vertices of $P_{S_A}[v_0, v_k]$, and also retains the relative order of the vertices of $P_{S_A}[v_0, v_k]$ as $v_0 \rightsquigarrow v_1 \rightsquigarrow v_2 \rightsquigarrow \cdots \rightsquigarrow v_k$ by traversing $P_T[v_0, v_k]$ from v_0 to v_k.*
3. *If $V(P_T[v_0, v_k]) \setminus V(P_{S_A}[v_0, v_k]) \neq \emptyset$, then the vertices in $V(P_T[v_0, v_k]) \setminus V(P_{S_A}[v_0, v_k])$ are all in R, and each vertex in $V(P_T[v_0, v_k]) \setminus V(P_{S_A}[v_0, v_k])$ has a fixed degree of 2 until the algorithm terminates.*

Proof. We show the lemma by induction on the number h of leaf-terminals handled before v_0. The basis case of $h = 0$ holds clearly. Assume that the result holds after $h = l$ leaf-terminals were handled. Now consider the current tree T' and the $(l + 1)$-th leaf-terminal, say $w_0(\neq v_0)$, is handled by the algorithm. By the induction hypothesis, there is a path connecting v_0 and v_k in T', denoted by $P_{T'}[v_0, v_k] = \langle u_0, u_1, u_2, \ldots, u_{q-1}, u_q\rangle$, where $u_0 = v_0$ and $u_q = v_k$, such that the desired three properties hold. By the execution of the algorithm, the condition for changing $P_{T'}[v_0, v_k]$ is u_j for some j in $\{1, 2, \ldots, q\}$, is the critical vertex of w_0 and one of its neighbor on $P_{T'}[v_0, v_k]$ is the target vertex of w_0. Therefore, if no u_j is selected as the critical vertex of w_0, then the result holds trivially. Otherwise, $P_{T'}[v_0, v_k]$ contains the critical vertex of w_0. Assume that u_i is the critical vertex of w_0. According to the algorithm, there must be a path $P_{T'}[w_0, u_i] = \langle w_0, w_1, \ldots, w_t, u_i\rangle$ such that $deg_{T'}(w_1) = deg_{T'}(w_2) = \cdots = deg_{T'}(w_t) = 2$ and $w_0, w_1, \ldots, w_t \in R$. Moreover, it is not difficult to verify that $P_{T'}[w_0, u_i]$ and $P_{T'}[v_0, v_k]$ are edge-disjoint sharing only one common vertex u_i. There are the following two cases:

Case 1. u_i is the critical vertex and u_{i-1} is the target vertex of w_0. Then, a new path $\langle u_0, u_1, \ldots, u_{i-1}, w_0, w_1, w_2, \ldots, w_t, u_i, u_{i+1}, \ldots, u_q\rangle$ is obtained after handling w_0.

Case 2. u_i is the critical vertex and u_{i+1} is the target vertex of w_0. Then, a new path $\langle u_0, u_1, \ldots, u_i, w_0, w_1, w_2, \ldots, w_t, u_{i+1}, u_{i+2}, \ldots, u_q\rangle$ is obtained after handling w_0.

According to the above cases, $P_{T'}[v_0, v_k]$ is now extended as the new path. By Lemma 6, the added vertices $w_0, w_1, w_2, \ldots, w_t$ are all in R, and each has a fixed degree of 2 until the algorithm terminates. Clearly, the relative order $v_0 \rightsquigarrow v_1 \rightsquigarrow v_2 \rightsquigarrow \cdots \rightsquigarrow v_k$ is also retained in the new path. Therefore, the three properties hold. □

The proofs of the following two lemmas are omitted.

Lemma 8. *Suppose that u and v are two different leaf-terminals of S_A such that u is handled before v by the algorithm. Then, the two paths $P_{S_A}[u, \alpha_u]$ and $P_{S_A}[v, \alpha_v]$ are edge-disjoint.*

Lemma 9. *Let $v_1, v_2, \ldots, v_l$ be an order of the leaf-terminals of S_A handled by Algorithm A_{ISTP}. Then, the paths $P_{S_A}[v_1, \alpha_{v_1}], P_{S_A}[v_2, \alpha_{v_2}], \ldots, P_{S_A}[v_l, \alpha_{v_l}]$ are pairwise edge-disjoint.*

Lemma 10. *Suppose that v is a leaf-terminal of S_A. Then, the target vertex β_v does not belong to $P_{S_A}[v, \alpha_v]$.*

Lemma 11. *Let $P = \langle v_1, v_2, \cdots, v_{k-1}, v_k \rangle$ be a path of a graph $G = (V, E)$ with a metric cost function $c : E \to \mathbb{R}^+$, and let $P' = \langle v_1, v_2, \cdots, v_{k-2}, v_{k-1} \rangle$ be a subpath of P. Then, $c(v_1, v_k) - c(v_{k-1}, v_k) \leq c(P')$, where $c(P') = \sum_{j=1}^{k-2} c(v_j, v_{j+1})$.*

Theorem 2. *Algorithm A_{ISTP} is a $(2\rho + 1)$-approximation algorithm for the internal Steiner tree problem, where ρ is the approximation ratio of the best-known algorithm for the Steiner tree problem.*

Proof. It is easy to see that the algorithm can transform a Steiner tree to an internal Steiner tree. We next analyze its approximation ratio. Let T_I^* and T_S^* be a minimum internal Steiner tree and a minimum Steiner tree of G, respectively. Since we use a ρ-approximation algorithm to find a Steiner tree S_A and then transform it to an internal Steiner tree, we have $c(S_A) \leq \rho c(T_S^*)$. Since T_I^* is also a Steiner tree for R, $c(T_S^*) \leq c(T_I^*)$. Therefore,

$$c(S_A) \leq \rho c(T_I^*). \tag{2}$$

Recall that in Step 3 of Algorithm A_{ISTP}, we add the edges into $E(S'_A)$ when $|V(S'_A) \setminus R| \leq 1$. Let μ denote the total cost of the added edges. If no edge is added in this case, then $\mu = 0$. Let T_I be an internal Steiner tree returned by Algorithm A_{ISTP}, and let $c(P_{S_A}[v, \alpha_v])$ be the sum of costs of the edges in $P_{S_A}[v, \alpha_v]$. By the construction of T_I, we have the following inequalities: $c(T_I) \leq c(S_A) + \sum_{v \in R}(c(v, \beta_v) - c(\alpha_v, \beta_v)) + \mu$ by the algorithm $\leq c(S_A) + \sum_{v \in R}(c(P_{S_A}[v, \alpha_v])) + \mu$ (by Lemmas 10 and 11) $\leq c(S_A) + c(S_A) + \mu$ (by Lemma 9) $= 2c(S_A) + \mu \leq 2\rho c(T_I^*) + \mu$ (by Equation 2). Consequently, $\frac{c(T_I)}{c(T_I^*)} = 2\rho + \frac{\mu}{c(T_I^*)}$.

We now bound the value $\frac{\mu}{c(T_I^*)}$. If $|V(S'_A) \setminus R| \geq 2$, then $\mu = 0 = \frac{\mu}{c(T_I^*)}$. Otherwise, if $|V(S'_A) \setminus R| \leq 1$, then $\mu > 0$. Obviously, a minimum internal

Steiner tree T_I^* must contain at least one edge (respectively, two edges) between $V(S'_A) \setminus R$ and $V(S'_A)$ when $|V(S'_A) \setminus R| = 1$ (respectively, $|V(S'_A) \setminus R| = 0$); otherwise, an internal Steiner tree cannot be constructed. Since μ is the smallest cost for the added edges, $\mu \leq c(T_I^*)$. Therefore, $\frac{\mu}{c(T_I^*)} \leq 1$ and the approximation ratio of the algorithm equals $2\rho + 1$. □

Algorithm A_{ISTP} can be implemented to run in $O(n^2 \log n) + f(m, n)$ time, where $f(n, m)$ is the time complexity of the best known approximation algorithm for the Steiner tree problem on an input graph with n vertices and m edges.

5 Concluding Remarks

In this paper, we prove that the internal Steiner tree problem is MAX SNP-hard. We also present a $(2\rho + 1)$-approximation algorithm for the problem under the metric space. It would be interesting to find a better (approximation) algorithm for the internal Steiner tree problem.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *Journal of the Association for Computing Machinery*, vol. 45, pp. 501–555, 1998.
2. M. Bern and P. Plassmann, "The Steiner problem with edge lengths 1 and 2," *Information Processing Letters*, vol. 32(4), pp. 171–176, 1989.
3. X. Cheng and D. Z. Du, *Steiner Trees in Industry*, Kluwer Academic Publishers, Dordrecht, Netherlands, 2001.
4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms* (Second Edition), MIT Press, Cambridge, 2001.
5. D. Z. Du, J. M. Smith, and J. H. Rubinstein, *Advance in Steiner Tree*, Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.
6. M. Garey, R. Graham, and D. Johnson, "The complexity of computing Steiner minimal trees," *SIAM Journal on Applied Mathematics*, vol. 32, pp. 835–859, 1977.
7. M. Garey and D. Johnson, "The rectilinear Steiner problem is NP-complete," *SIAM Journal on Applied Mathematics*, vol. 32, pp. 826–834, 1977.
8. F. K. Hwang, D. S. Richards, and P. Winter, "The Steiner Tree Problem," *Annuals of Discrete Mathematices 53*, Elsevier Science Publishers, Amsterdam, 1992.
9. R. Karp, "Reducibility among combinatorial problems," in R. E. Miller, J. W. Thatcher eds.: *Complexity of Computer Computations*, Plenum Press, New York, pp. 85–103, 1972.
10. C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," in *Proceedings of the 20th ACM Symposium on Theory of Computing*, pp. 229–234, 1988.
11. G. Robins and A. Zelikovsky, "Improved Steiner tree approximation in graphs," in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), pp. 770–779, 2000.

Approximately Optimal Trees for Group Key Management with Batch Updates*

Minming Li[1], Ze Feng[1], Ronald L. Graham[2], and Frances F. Yao[1]

[1] Department of Computer Science
City University of Hong Kong
{minmli,fengze}@cs.cityu.edu.hk, csfyao@cityu.edu.hk
[2] Department of Computer Science and Engineering
University of California at San Diego
graham@ucsd.edu

Abstract. We investigate the group key management problem for broadcasting applications. Previous work showed that, in handling key updates, batch rekeying can be more cost-effective than individual rekeying. One model for batch rekeying is to assume that every user has probability p of being replaced by a new user during a batch period with the total number of users unchanged. Under this model, it was recently shown that an optimal key tree can be constructed in linear time when p is a constant and in $O(n^4)$ time when $p \to 0$. In this paper, we investigate more efficient algorithms for the case $p \to 0$, i.e., when membership changes are sparse. We design an $O(n)$ heuristic algorithm for the sparse case and show that it produces a nearly 2-approximation to the optimal key tree. Simulation results show that its performance is even better in practice. We further design a refined heuristic algorithm and show that it achieves an approximation ratio of $1 + \epsilon$ as $p \to 0$.

1 Introduction

With the increase of subscription-based network services, strategies for achieving secure multicast in networks are becoming more important. For example, to limit the service access to the authorized subscribers only, mechanisms such as content encryption and selective distribution of decryption keys have been found useful. One can regard the secure multicast problem as a group broadcast problem, where we have n subscribers and a group controller (GC) that periodically broadcasts messages (e.g., a video clip) to all subscribers over an insecure channel. To guarantee that only the authorized users can decode the contents of the messages, the GC will dynamically maintain a key structure for the whole group. Whenever a user leaves or joins, the GC will generate some new keys

* This work was supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, the U.S. National Science Foundation Grant CCR-0310991, a grant from the Research Grants Council of Hong Kong under Project Number CityU 1165/04E and a grant from City University of Hong Kong (Project No. 7200072).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 284–295, 2007.
© Springer-Verlag Berlin Heidelberg 2007

as necessary and notify the remaining users of the group in some secure way. Surveys on the key management for secure group communications can be found in [1][2].

In this paper, we consider the key tree model [3] for the key management problem. We describe this model briefly as follows (a precise formulation is given in Section 2). Every leaf node of the key tree represents a user and stores his individual key. Every internal node stores a key shared by all leaf descendants of that internal node. Every user possesses all the keys along the path from the leaf node (representing the user) to the root. To prevent revoked users from knowing future message contents and also to prevent new users from knowing past message contents, the GC updates a subset of keys, whenever a new user joins or a current user leaves, as follows. As long as there is a user change among the leaf descendants of an internal node v, the GC will 1) replace the old key stored at v with a new key, and 2) broadcast (to all users) the new key encrypted with the key stored at each child node of v. Note that only users corresponding to the leaf descendants of v can decipher useful information from the broadcast. Furthermore, this procedure must be done in a bottom-up fashion (i.e., starting with the lowest v whose key must be updated—see Section 2 for details) to guarantee that a revoked user will not know the new keys. The cost of the above procedure counts the number of encryptions used in step 2) above (or equivalently, the number of broadcasts made by the GC).

When users change frequently, the method for updating the group keys whenever a user leaves or joins may be too costly. Thus, a batch rekeying strategy was proposed by Li et al. in [4] whereby rekeying is done only periodically instead of immediately after each membership change. It was shown by simulation that among the totally balanced key trees (where all internal nodes of the tree have branching degree 2^i), degree 4 is the best when the number of requests (leave/join) within a batch is not large. For a large number of requests, a star (a tree of depth 1) outperforms all such balanced key trees. Further work on the batch rekeying model was done by Zhu et al. in [5]. They introduced a new model where the number of joins is assumed to be equal to the number of leaves during a batch updating period and every user has probability p of being replaced by a new user for some p. They studied the optimal tree structure subject to two restrictions: A) the tree is totally balanced, and B) every node on level i has 2^{k_i} children for some parameter k_i depending on i. Under these restrictions, characterizations of the optimal key tree were given together with a construction algorithm.

Recently, Graham, Li and Yao [6] studied the structure of the true optimal key tree when restrictions A) and B) are both removed. They proved that when $p > 1 - 3^{-1/3} \approx 0.307$, the optimal tree is an n-star. When $p \leq 1 - 3^{-1/3}$, they proved a constant upper bound 4 for the branching degree of any internal node v other than the root, and also an upper bound of $-4/\log q$, where $q = 1 - p$, for the size of the subtree rooted at v. By using these characterizations, they designed an $O(n^4)$ algorithm for computing the optimal key tree for n users. The running time of their algorithm is in fact linear when p is a fixed

constant, but becomes $O(n^4)$ when p approaches 0. Although polynomial, the $O(n^4)$ complexity is still too costly for large scale applications. Indeed, the case $p \to 0$ (when user changes are sparse) is a realistic scenario in many applications. In this paper, we investigate more efficient heuristics for the sparse case. As shown in [6], degree 3 is quite favored in the optimal tree as $p \to 0$. In fact, their results implied that for $n = 3^t$, the optimal key tree is a balanced ternary tree, and for many other values of n, the optimal tree is as close to a balanced ternary tree with n leaves as possible, subject to some number-theoretical properties of n.

In this paper, we investigate how closely a "simple" ternary tree can approximate the optimal tree for arbitrary n. We propose a heuristic *LR* which constructs a ternary tree in a left to right manner and prove that it gives a nearly 2-approximation to the optimal tree. Simulation results show that the heuristic performs much better than the theoretical bound we obtain. We then design a refined heuristic *LB* whose approximation ratio is shown to be $1 + \epsilon$ as $p \to 0$ where ϵ can be made arbitrarily small.

The rest of the paper is organized as follows. In Section 2, we describe the batch update model in detail. In Section 3, we establish a lower bound for the optimal tree cost as $p \to 0$. The lower bound is useful for obtaining performance ratios for our approximation algorithms. In Section 4, We describe the heuristic LR and analyze its performance against the optimal key tree; some simulation results are also given. Then we design a refined heuristic LB and analyze its performance in Section 5. Finally, we summarize our results and mention some open problems in Section 6.

2 Preliminaries

Before giving a precise formulation of the key tree optimization problem to be considered, we briefly discuss its motivation and review the basic key tree model for group key management. This model is referred to in the literature either as key tree [3] or LKH (logical key hierarchy) [7].

In the key tree model, there are a Group Controller (GC), represented by the root, and n subscribers (or users) represented by the n leaves of the tree. The tree structure is used by the GC for key management purposes. Associated with every node of the tree (whether internal node or leaf) is an encryption key. The key associated with the root is called the Traffic Encryption Key (TEK), which is used by the subscribers for accessing encrypted service contents. The key k_v associated with each non-root node v is called a Key Encryption Key (KEK) which is used for updating the TEK when necessary. Each subscriber possesses all the keys along the path from the leaf representing the subscriber to the root.

In the batch update model to be considered, only simultaneous join/leave is allowed, that is, whenever there is a revoked user, a new user will be assigned to that vacant position. This assumption is justified since, in a steady state, the number of joins and departures would be roughly equal during a batch processing period. To guarantee forward and backward security, a new user assigned to

a leaf position will be given a new key by the GC and, furthermore, all the keys associated with the ancestors of the leaf must be updated by the GC. The updates are performed from the lowest ancestor upward for security reasons. We then explain the updating procedure together with the updating cost in the following.

The GC first communicates with each new subscriber separately to assign a new key to the corresponding leaf. After that, the GC will broadcast certain encrypted messages to all subscribers in such a way that each valid subscriber will know all the new keys associated with its leaf-to-root path while the revoked subscribers will not know any of the new keys. The GC accomplishes this task by broadcasting the new keys, in encrypted form, from the lowest level upward as follows. Let v be an internal node at the lowest level whose key needs to be (but has not yet been) updated. For each child u of v, the GC broadcasts a message containing $E_{k_u^{new}}(k_v^{new})$, which means the encryption of k_v^{new} with the key k_u^{new}. Thus the GC sends out d_v broadcast messages for updating k_v if v has d_v children. Updating this way ensures that the revoked subscribers will not know any information about the new keys (as long as they do not know the new key k_u^{new}in a lower level, they can not get the information of the new key k_v^{new} in a higher level) while current subscribers can use one of their KEKs to decrypt the useful $E_{k_u^{new}}(k_v^{new})$ sequentially until they get the new TEK.

We adopt the probabilistic model introduced in [5] that each of the n positions has the same probability p to independently experience subscriber change during a batch rekeying period. Under this model, an internal node v with N_v leaf descendants will have probability $1 - q^{N_v}$ that its associated key k_v requires updating, where $q = 1 - p$. The updating incurs $d_v \cdot (1 - q^{N_v})$ expected broadcast messages by the procedure described above. We thus define the expected updating cost $C(T)$ of a key tree T by $C(T) = \sum_v d_v \cdot (1 - q^{N_v})$ where the sum is taken over all the internal nodes v of T. It is more convenient to remove the factor d_v from the formula by associating the weight $1 - q^{N_v}$ with each of v's children. This way we express $C(T)$ as a node weight summation: for each non-root tree node u, its node weight is defined to be $1 - q^{N_v}$ where v is u's parent. The optimization problem we are interested in can now be formulated as follows.

Optimal Key Tree for Batch Updates: We are given two parameters $0 \leq p \leq 1$ and $n > 0$. Let $q = 1 - p$. For a rooted tree T with n leaves and node set V (including internal nodes and leaves), define a weight function $w(u)$ on V as follows. Let $w(r) = 0$ for root r. For every non-root node u, let $w(u) = 1 - q^{N_v}$ where v is u's parent. Define the cost of T as $C(T) = \sum_{u \in V} w(u)$. Find a T for which $C(T)$ is minimized. We say that such a tree is (p, n)-*optimal*, and denote its cost by $\mathrm{OPT}(p, n)$.

3 Lower Bound for Optimal Tree Cost as $p \to 0$

In a tree T with n leaves, denote the set of leaf nodes as $L(T)$ and for each leaf u, let the set of ancestor nodes of u (including u itself) be denoted by $Anc(u)$.

To obtain a lower bound for the optimal tree cost, we first rewrite $C(T)$ as

$$C(T) = \sum_{u \in V} w(u) = \sum_{u \in V} N_u \cdot \frac{w(u)}{N_u} = \sum_{u \in L(T)} \sum_{x \in Anc(u)} \frac{w(x)}{N_x} = \sum_{u \in L(T)} c(u),$$

where we define $c(u) = \sum_{x \in Anc(u)} \frac{w(x)}{N_x}$. In other words, we distribute the weight $w(u)$associated with every node $u \in V$ evenly among its leaf descendants, and then sum the cost over all the leaves of T.

Let the path from a leaf u to the root r be $p_0 p_1 \ldots p_{k-1} p_k$, where $p_0 = u$ and $p_k = r$. Note that $c(u) = \sum_{i=0}^{k-1} \frac{1-q^{N_{p_i+1}}}{N_{p_i}}$ is uniquely determined by the sequence of numbers $\{N_{p_0}, N_{p_1}, \ldots, N_{p_k}\}$, where $N_{p_0} = 1$ and $N_{p_k} = n$. We will thus extend the definition of c to all such sequences $\{a_0, a_1, \ldots, a_k\}$ and analyze the minimum value of c.

Definition 1. *Let S_n denote any sequence of integers $\{a_0, a_1, \ldots, a_k\}$ satisfying $1 = a_1 < a_2 < \ldots < a_k = n$. We call S_n an n-progression. Define $c(p, S_n)$ to be $c(p, S_n) = \sum_{i=1}^{k} \frac{1-q^{a_i}}{a_{i-1}}$ and let $F(p, n)$ to be the minimum of $c(p, S_n)$ over all n-progressions S_n. For $n = 3^t$, the special n-progression $\{1, 3, 9, \ldots, 3^{t-1}, 3^t\}$ will be denoted by S_n^*.*

Thus, we have $C(T) \geq n \cdot F(p, n)$ for any tree T with n leaves, and hence

$$\mathrm{OPT}(p, n) \geq n \cdot F(p, n). \tag{1}$$

Next we focus on properties of $c(p, S_n)$ and $F(p, n)$. First, we derive the following monotone property for $F(p, n)$.

Lemma 1. $F(p, n) < F(p, n+1)$.

Proof. Suppose $S_{n+1} = \{1, a_1, a_2, \ldots, a_{k-1}, n+1\}$ is the optimal $(n+1)$-progression that achieves the value $F(p, n+1)$. Let $S_n = \{1, a_1, a_2, \ldots, a_{k-1}, n\}$. Because $\frac{1-q^{n+1}}{a_{k-1}} > \frac{1-q^n}{a_{k-1}}$, we know that $c(p, S_{n+1}) > c(p, S_n)$. By definition, we have $F(p, n) \leq c(p, S_n)$. Combining these two facts, we have $F(p, n) < F(p, n+1)$. □

For a given n-progression $S_n = \{1, a_1, a_2, \ldots, a_{k-1}, n\}$, the slope of $c(p, S_n)$ at $p = 0$ is denoted by λ_{S_n} and can be expressed as $\lambda_{S_n} = \sum_{i=0}^{k-1} \frac{a_{i+1}}{a_i}$ where a_0=1 and a_k=n. The minimum $c(p, S_n)$ as $p \to 0$ will be achieved by those S_n with $c(p, S_n)$ having the smallest slope at $p = 0$. We next prove the following lemma.

Lemma 2. *When $n = 3^t$, the n-progression $S_n^* = \{1, 3, 9, \ldots, 3^{t-1}, 3^t\}$ satisfies the following relation: $\lambda_{S_n} \geq 0.995 \cdot \lambda_{S_n^*}$ for any S_n.*

Proof. For positive numbers $b_1, b_2, \ldots, b_k$, we have $\sum_{i=1}^{k} b_i \geq k(\prod_{i=1}^{k} b_i)^{\frac{1}{k}}$. Therefore, $\lambda_{S_n} \geq k n^{\frac{1}{k}}$ if S_n consists of k numbers. We now estimate a lower bound for $f(k) = k n^{\frac{1}{k}}$ when $n = 3^t$. Consider $g(k) = \log_3 f(k) = \frac{\ln k}{\ln 3} + \frac{t}{k}$. Notice that $g'(k) = \frac{1}{k \ln 3} - \frac{t}{k^2}$. Therefore, we have $g'(k) < 0$ when $k < t \ln 3$ and

$g'(k) > 0$ when $k > t \ln 3$. This implies that $g(k)$, and hence $f(k)$, is minimized when $k = t \ln 3$. Therefore, we have $f(k) \geq f(t \ln 3) = t 3^{\frac{1}{\ln 3}} \ln 3$, which implies $\lambda_{S_n} \geq t 3^{\frac{1}{\ln 3}} \ln 3$. On the other hand, we know that $\lambda_{S_n^*} = f(t) = 3t$. Hence we have

$$\frac{\lambda_{S_n}}{\lambda_{S_n^*}} \geq \frac{f(t \ln 3)}{f(t)} = 3^{\frac{1+\ln \ln 3}{\ln 3} - 1} \approx 0.9950250.995. \geq \qquad \square$$

We obtain the following theorem from the above analysis.

Theorem 1. *For* $3^t \leq n < 3^{t+1}$*, we have* $\mathrm{OPT}(p, n) \geq 0.995 \cdot n \cdot c(p, S_{3^t}^*)$ *when* $p \to 0$*.*

Proof. This is a direct consequence of Lemma 1, Lemma 2 and inequality (1). $\square$

4 Heuristic LR and Its Approximation Ratio

We design the heuristic LR as follows. LR maintains an almost balanced ternary tree (i.e., the depth of any two leaves differ by at most 1) in which at most one internal node has degree less than 3. Moreover, LR adds new leaves incrementally in a left to right order. Figure 1 shows the tree we get by using LR for $n = 2, \ldots, 9$. We can also recursively build a key tree using LR in the following way. For a tree with $n \geq 3$ leaves, the number of leaves in the root's three subtrees is decided by the table below; while for a tree with 2 leaves, the tree structure is a root with two children (a star).

	No. of leaves (Left)	No. of leaves (Middle)	No. of leaves (Right)
$3^t \leq n < 5 \cdot 3^{t-1}$	$n - 2 \cdot 3^{t-1}$	3^{t-1}	3^{t-1}
$5 \cdot 3^{t-1} \leq n < 7 \cdot 3^{t-1}$	3^t	$n - 4 \cdot 3^{t-1}$	3^{t-1}
$7 \cdot 3^{t-1} \leq n < 3^{t+1}$	3^t	3^t	$n - 2 \cdot 3^t$

We denote the tree with n leaves constructed by LR as T_n. Note that this heuristic only needs linear time to construct a ternary tree. Furthermore, the structure of the ternary tree can be decided in $\log n$ time because every time we go down the tree, there is at most one subtree whose number of leaves is not a power of 3 and needs further calculation.

Let $\mathrm{LR}(p, n)$ denote the cost of the ternary tree constructed by LR for given n and p. To obtain an upper bound for $\mathrm{LR}(p, n)$, we first prove the following lemmas.

Lemma 3. *The inequality* $\mathrm{LR}(p, n) < \mathrm{LR}(p, n+1)$ *holds for all* $n > 0$ *and* $0 < p < 1$*.*

Proof. We view $C(T_n)$ as the node weight summation given in Section 2 and compare the cost of the corresponding nodes $w(u)$ and $w(u')$ in T_n and T_{n+1} respectively. Due to the addition of one leaf node, if $w(u) = 1 - q^k$, then $w(u') = w(u)$ or $w(u') = 1 - q^{k+1}$. Therefore we have $w(u) \leq w(u')$. There are also additional weights associated with nodes that appear in T_{n+1} but not in T_n. This proves the lemma. $\square$

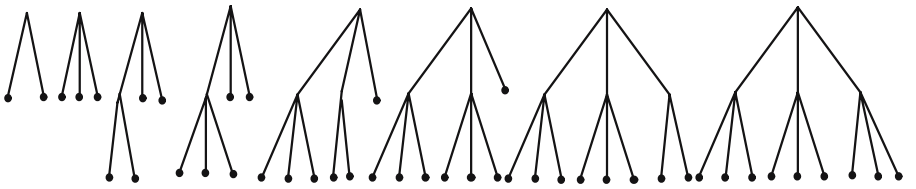


Fig. 1. Trees generated by LR for $n = 2$ to 9

Lemma 4. *For any integer $t > 0$ and $0 < q < 1$, we have* $\frac{1-q^{3^t}}{3^{t-1}} > \frac{1-q^{3^{t+1}}}{3^t}$.

Proof. Note that $3\sum_{i=1}^{3^t} q^{i-1} > (1 + q^{3^t} + q^{2\cdot 3^t})\sum_{i=1}^{3^t} q^{i-1} = \sum_{i=1}^{3^{t+1}} q^{i-1}$. The lemma is proved by multiplying $\frac{1-q}{3^t}$ on both sides. □

Lemma 5. *For any integer $t > 0$ and $0 < q < 1$, we have*

$$\mathrm{LR}(p, 3^{t+1}) < 3(1 + \frac{1}{t})\mathrm{LR}(p, 3^t).$$

Proof. By Lemma 4 and the definitions, we have $c(p, S^*_{3^t})/t > c(p, S^*_{3^{t+1}})/(t+1)$. Therefore, we have

$$\mathrm{LR}(p, 3^{t+1}) = 3^{t+1}\cdot c(p, S^*_{3^{t+1}}) < 3(1+\frac{1}{t})\cdot 3^t \cdot c(p, S^*_{3^t}) = 3(1+\frac{1}{t})\mathrm{LR}(p, 3^t). \qquad \square$$

Now we are ready to prove the first approximation ratio.

Theorem 2. *When $p \to 0$, we have* $\mathrm{LR}(p, n) < 3.015(1 + \frac{1}{\lfloor \log_3 n \rfloor})\mathrm{OPT}(p, n)$.

Proof. Suppose $3^t \le n < 3^{t+1}$. We claim the following

$$\begin{aligned}
\mathrm{LR}(p, n) &< \mathrm{LR}(p, 3^{t+1}) \\
&< 3(1 + \frac{1}{t})\mathrm{LR}(p, 3^t) \\
&= 3(1 + \frac{1}{t})3^t \cdot c(p, S^*_{3^t}) \\
&\le 3.015(1 + \frac{1}{t})\mathrm{OPT}(p, n).
\end{aligned}$$

The first inequality is implied by Lemma 3 and the second one by Lemma 5. The last inequality holds due to Theorem 1. □

In the above discussion, we use the smallest balanced ternary tree with no less than n leaves as an upper bound for $\mathrm{LR}(p, n)$. By adding a small number of leaves instead of filling the whole level, we can obtain a better approximation ratio which is shown below.

We divide the integers in the range $(3^t, 3^{t+1}]$ into three consecutive subsets of equal size $H = \frac{3^{t+1}-3^t}{3}$ as follows:

$$P_1 = (3^t, 3^t + H], P_2 = (3^t + H, 3^t + 2H], P_3 = (3^t + 2H, 3^{t+1}].$$

For any $n \in P_i$, we can use $\mathrm{LR}(p, n')$ where $n' = \max P_i$ to upper bound the value of $\mathrm{LR}(p, n)$ by Lemma 3. Let $\Delta_t = \mathrm{LR}(p, 3^t) - \mathrm{LR}(p, 3^{t-1})$ and define $a = 1 - q^{3^{t+1}}$. Notice that

$$\mathrm{LR}(p, 3^{t+1}) = 3a + 3 \cdot \mathrm{LR}(p, 3^t) = 3a + 3\Delta_t + 3 \cdot \mathrm{LR}(p, 3^{t-1}).$$

It's not hard to verify the following inequalities based on the definition of the tree cost:

$$\mathrm{LR}(p, 7 \cdot 3^{t-1}) < \mathrm{LR}(p, 3^{t+1}) - \Delta_t,$$
$$\mathrm{LR}(p, 5 \cdot 3^{t-1}) < \mathrm{LR}(p, 3^{t+1}) - 2\Delta_t.$$

We now derive a lower bound for the value of Δ_t.

Lemma 6. *For $0 < p < 1$, we have $\Delta_t \geq \frac{1}{6} \cdot \mathrm{LR}(p, 3^{t+1})$.*

Proof. We only need to prove $\Delta_t > a + \mathrm{LR}(p, 3^{t-1})$. By the definition of Δ_t, we know that $\Delta_t = 2 \cdot \mathrm{LR}(p, 3^{t-1}) + 3(1 - q^{3^t})$. Then by using Lemma 4, we have $3(1 - q^{3^t}) \geq (1 - q^{3^{t+1}})$, which implies $\mathrm{LR}(p, 3^{t-1}) + 3(1 - q^{3^t}) \geq (1 - q^{3^{t+1}})$. Therefore, we have $\Delta = 2 \cdot \mathrm{LR}(p, 3^{t-1}) + 3(1 - q^{3^t}) > \mathrm{LR}(p, 3^{t-1}) + a$. □

By making use of Lemma 6, we can obtain the following theorem on the performance of LR (proof given in the Appendix).

Theorem 3. *When $p \to 0$, we have $\mathrm{LR}(p, n) < 2.01(1 + \frac{1}{\lfloor \log_3 n \rfloor})\mathrm{OPT}(p, n)$.*

Proof. We prove the theorem using Lemma 6 and similar arguments used in Theorem 2. The discussion below is divided into three cases according to the value of n.

Case A) $3^t < n \leq 5 \cdot 3^{t-1}$.

$$\begin{aligned}
\mathrm{LR}(p, n) &< \mathrm{LR}(p, 5 \cdot 3^{t-1}) \\
&< \frac{2}{3}\mathrm{LR}(p, 3^{t+1}) \\
&< \frac{2}{3} \cdot 3.015(1 + \frac{1}{t}) \cdot \frac{3^t}{n} \cdot \mathrm{OPT}(p, n) \\
&\leq 2.01(1 + \frac{1}{\lfloor \log_3 n \rfloor}) \cdot \mathrm{OPT}(p, n).
\end{aligned}$$

Case B) $5 \cdot 3^{t-1} < n \leq 7 \cdot 3^{t-1}$.

$$\begin{aligned}
\mathrm{LR}(p, n) &< \mathrm{LR}(p, 7 \cdot 3^{t-1}) \\
&< \frac{5}{6}\mathrm{LR}(p, 3^{t+1}) \\
&< \frac{5}{6} \cdot 3.015(1 + \frac{1}{t}) \cdot \frac{3^t}{n} \cdot \mathrm{OPT}(p, n) \\
&< \frac{5}{6} \cdot 3.015(1 + \frac{1}{t}) \cdot \frac{3}{5} \cdot \mathrm{OPT}(p, n) \\
&< 2.01(1 + \frac{1}{\lfloor \log_3 n \rfloor}) \cdot \mathrm{OPT}(p, n).
\end{aligned}$$

Case C) $7 \cdot 3^{t-1} < n \leq 3^{t+1}$.

$$\begin{aligned}\mathrm{LR}(p,n) &< \mathrm{LR}(p,3^{t+1}) \\ &< 3.015(1+\frac{1}{t})\cdot\frac{3^t}{n}\cdot\mathrm{OPT}(p,n) \\ &< 3.015(1+\frac{1}{t})\cdot\frac{3}{7}\cdot\mathrm{OPT}(p,n) \\ &< 2.01(1+\frac{1}{\lfloor\log_3 n\rfloor})\cdot\mathrm{OPT}(p,n). \end{aligned}$$ □

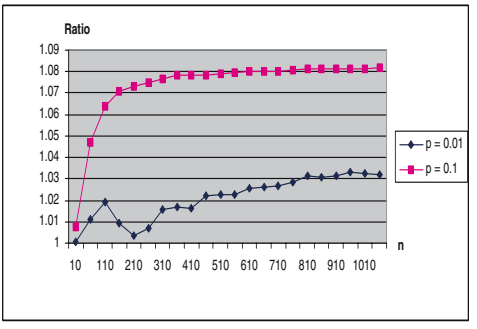
We run simulations on the performance of LR for various values of p and n. Define $Ratio(p,n) = \mathrm{LR}(p,n)/\mathrm{OPT}(p,n)$. Figure 2(a) shows $Ratio(p,n)$ as a function of n for $p = 0.1$ and $p = 0.01$ respectively. Within the simulation range, we see that for the same n, $Ratio(p,n)$ is larger when p is larger. For $p = 0.1$ the maximum ratio within the range is below 1.09. We then simulated the performance of LR when $p \to 0$ as a function of n. Figure 2(b) shows $Ratio(p,n)$ when we set $p = 0.01/n^3$. We found the maximum ratio reached within the simulation range to be less than 1.018. Figure 3 plots $Ratio(p,n)$ as a function of p while n is chosen to be $n = 100$ and $n = 500$ respectively. Notice that each curve has some dips and is not monotonically increasing with p.

5 Heuristic LB and Its Approximation Ratio

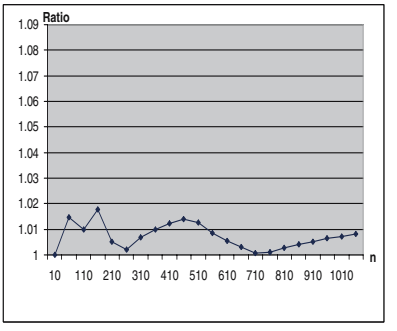
In this section, we consider a more refined heuristic LB (Level Balance) and show that it has an approximation ratio of $1+\epsilon$ as $p \to 0$.

The heuristic LB adopts two different strategies for adding a new leaf to a tree when n grows from 3^t to 3^{t+1}. Let T'_n denote the ternary tree with n leaves constructed by LB. When $3^t < n \le 2\cdot 3^t$, the heuristic LB changes the leftmost leaf on level t in T'_{n-1} into a 2-star (i.e., an internal node with 2 leaf children); when $2\cdot 3^t < n \le 3^{t+1}$, the heuristic LB changes the leftmost 2-star on level t in T'_{n-1} into a 3-star (an internal node with three leaf children).

In the following discussion, we refer to the initial formulation of $C(T)$ as a node weight summation over all nodes: $C(T) = \sum_{u\in V} w(u)$. Define the slope of $C(T)$ at $p = 0$ as β_T and let $t = \lfloor log_3 n\rfloor$. When T'_{3^t} changes incrementally to T'_n where $3^t < n \le 2\cdot 3^t$, every time the size is increased by 1, we change a single leaf node to a 2-star. Therefore, in all intermediate levels (except for the root and the bottom level), exact three nodes will undergo a weight change: it changes from $1-q^{N_v}$ to $1-q^{N_v+1}$. For the two newly added nodes, the weight of each node is $1-q^2$. Altogether, these changes contribute an increase of $3t+2\cdot 2$ to the slope $\beta_{T'_n}$. When $T'_{2\cdot 3^t}$ is changes incrementally to T'_n where $2\cdot 3^t < n \le 3^{t+1}$, by using a similar argument, we conclude that every time the size is increased by 1, all weight changes together contribute an increase of $3t+9-4 = 3t+5$ to the slope $\beta_{T'_n}$, where $9-4$ means the slope increase due to $3(1-q^3)-2(1-q^2)$



(a) $p = 0.1$ and $p = 0.01$



(b) $p = 0.01/n^3$

Fig. 2. Simulation results on $Ratio(p, n)$ for $p = 0.1$, $p = 0.01$ and $p = 0.01/n^3$

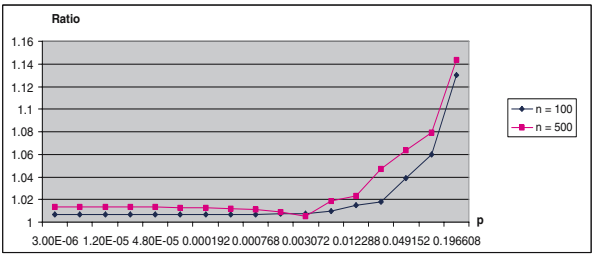


Fig. 3. Simulation results for fixed n

(because we change a 2-star to a 3-star). To summarize, our new heuristic has the following property.

$$\beta_{T'_n} = \begin{cases} \beta_{T'_{n-1}} + 3t + 4 \text{ if } \ 3^t < n \leq 2 \cdot 3^t, \\ \beta_{T'_{n-1}} + 3t + 5 \text{ if } 2 \cdot 3^t < n \leq 3^{t+1}. \end{cases} \tag{2}$$

Using the base value of $\beta_{T'_n}$ and the recurrence relation (2), we have the following lemma.

Lemma 7

$$\beta_{T'_n} = \begin{cases} (3t+4)n - 4 \cdot 3^t \ \textit{if} \ \ 3^t < n \leq 2 \cdot 3^t, \\ (3t+5)n - 6 \cdot 3^t \ \textit{if} \ 2 \cdot 3^t < n \leq 3^{t+1}. \end{cases}$$

By Lemma 1 and Theorem 1, we can prove the following Theorem. Please refer to the Appendix for details.

Theorem 4. *When $p \to 0$, we have $C(T'_n) < 1.005(1 + \frac{1}{\lfloor \log_3 n \rfloor})\mathrm{OPT}(p, n)$.*

Proof. To compare the cost of two trees as $p \to 0$ is in fact comparing slopes of the corresponding $C(T)$ at $p = 0$. Note that for a full ternary tree T with height

t, the slope of $C(T)$ at $p = 0$ equals $3t \cdot 3^t$. Always let $t = \lfloor log_3 n \rfloor$ in the following. Using Lemma 1, we can prove the theorem by proving $\beta_{T'_n} < (1 + \frac{1}{t}) \cdot 3t \cdot n$ as follows.

Case A) $n = 3^t + r$ where $0 < r \leq 3^t$. In this case, we have

$$\begin{aligned}\beta_{T'_n} &= 3t \cdot 3^t + r(3t + 4) \\ &< 3t \cdot 3^t + r(3t + 3) + 3 \cdot 3^t \\ &= (3t + 3)(3^t + r) \\ &= (1 + \frac{1}{t}) \cdot 3t \cdot n.\end{aligned}$$

Case B) $n = 2 \cdot 3^t + r$ where $0 < r \leq 3^t$. In this case, we have

$$\begin{aligned}\beta_{T'_n} &= 3t \cdot 3^t + 3^t \cdot (3t + 4) + r(3t + 5) \\ &= 2 \cdot 3t \cdot 3^t + 4 \cdot 3^t + 3t \cdot r + 5r \\ &\leq 2 \cdot 3t \cdot 3^t + 6 \cdot 3^t + 3t \cdot r + 3r \\ &= (3t + 3)(2 \cdot 3^t + r) \\ &= (1 + \frac{1}{t}) \cdot 3t \cdot n.\end{aligned}$$

□

The upper bound in Theorem 4 can be further improved to $1 + \epsilon$ where ϵ can be made arbitrarily small when $p \to 0$. To accomplish this, we make use of the following technical lemma which was originally proved in [6] for a different purpose.

Lemma 8. *For any tree T with n leaves, we have*

$$\beta_T \geq \begin{cases} (3t + 4)n - 4 \cdot 3^t \text{ if } \ 3^t < n \leq 2 \cdot 3^t, \\ (3t + 5)n - 6 \cdot 3^t \text{ if } 2 \cdot 3^t < n \leq 3^{t+1}. \end{cases}$$

By comparing Lemma 7 and 8 we can deduce that, for the tree T'_n constructed by the heuristic LB, the slope of $C(T'_n)$ is equal to the lower bound of the slope of the optimal key tree with n leaves. Therefore, as $p \to 0$, the approximation ratio of LB can be arbitrarily close to 1. This leads to the following theorem.

Theorem 5. *For any fixed n, we have $C(T'_n) < (1 + \epsilon)\mathrm{OPT}(p, n)$ when $p \to 0$.*

6 Conclusions

In this paper, we consider the group key management problem for broadcasting applications. In particular, we focus on the case when membership changes are sparse. Under the assumption that every user has probability p of being replaced by a new user during a batch rekeying period, previously available algorithm requires $O(n^4)$ time to build the optimal key tree as $p \to 0$. We design a linear-time heuristic LR to construct an approximately good key tree and analyze its performance as $p \to 0$. We prove that LR produces a nearly 2-approximation

to the optimal key tree. Simulation results show that LR performs much better than the theoretical bound we obtain. We also design a refined heuristic LB whose approximation ratio is shown to be $1 + \epsilon$ as $p \to 0$.

Some interesting problems remain open. We believe that the heuristics LB and LR also perform well for p in a more general range, not just when p approaches 0. In [6], it was shown that a star is the optimal tree when $p \in (0.307, 1)$. Do LR and LB perform well for all $p \leq 0.307$? We are so far not able to prove a constant approximation ratio for this range. To prove a constant bound, one needs to understand better the mathematical structure of the optimal n-progression S_n that can achieve the value $F(p, n)$ for arbitrary p and n.

References

1. S. Rafaeli and D. Hutchison, *A Survey of Key Management for Secure Group Communication*, ACM Computing Surveys, v.35 n.3, p.309-329, 2003.
2. M. T. Goodrich, J. Z. Sun, and R. Tamassia, *Efficient Tree-Based Revocation in Groups of Low-State Devices*, Proceedings of CRYPTO, 2004.
3. C. K. Wong and M. G. Gouda, and S. S. Lam, *Secure Group Communications Using Key Graphs*, IEEE/ACM Transactions on Networking, v.8 n.1, p.16-30, 2003.
4. X. S. Li, Y. R. Yang, M. G. Gouda, and S .S. Lam, *Batch Re-keying for Secure Group Communications*, WWW10, May2-5, 2001, Hong Kong.
5. F. Zhu, A. Chan, and G. Noubir, *Optimal Tree Structure for Key Management of Simultaneous Join/Leave in Secure Multicast*, Proceedings of MILCOM, 2003.
6. R. L. Graham, M. Li and F. F. Yao, *Optimal Tree Structures for Group Key Management with Batch Updates*, to appear in SIAM Journal on Discrete Mathematics.
7. D. Wallner, E. Harder, and R. C. Agee, *Key Management for Multicast: Issues and Architectures*, RFC 2627, June 1999.

On Deciding Deep Holes of Reed-Solomon Codes

Qi Cheng and Elizabeth Murray*

School of Computer Science
The University of Oklahoma
Norman, OK 73019, USA
{qcheng, wumpus}@ou.edu

Abstract. For generalized Reed-Solomon codes, it has been proved [7] that the problem of determining if a received word is a deep hole is co-NP-complete. The reduction relies on the fact that the evaluation set of the code can be exponential in the length of the code – a property that practical codes do not usually possess. In this paper, we first present a much simpler proof of the same result. We then consider the problem for standard Reed-Solomon codes, i.e. the evaluation set consists of all the nonzero elements in the field. We reduce the problem of identifying deep holes to deciding whether an absolutely irreducible hypersurface over a finite field contains a rational point whose coordinates are pairwise distinct and nonzero. By applying Cafure-Matera estimation of rational points on algebraic varieties, we prove that the received vector $(f(\alpha))_{\alpha\in\mathbf{F}_p}$ for the Reed-Solomon $[p-1,k]_p$, $k < p^{1/4-\epsilon}$, cannot be a deep hole, whenever $f(x)$ is a polynomial of degree $k+d$ for $1 \le d < p^{3/13-\epsilon}$.

Keywords: Reed-Solomon codes, deep hole, NP-complete, algebraic surface.

1 Introduction

A signal, when transfered over a long distance, always has a possibility of being corrupted. Error-detecting and error-correcting codes alleviate the problem and make the modern communication possible. The Reed-Solomon codes are very popular in engineering a reliable channel due to their simplicity, burst error-correction capabilities, and the powerful decoding algorithms they admit.

Let $\mathbf{F}_q$ be the finite field with q elements, where q is a prime power. The encoding process of generalized Reed-Solomon codes can be thought of as a map from $\mathbf{F}_q^k \to \mathbf{F}_q^n$, in which a message $(a_1, a_2, \cdots, a_k)$ is mapped to a vector

$$(f(x_1), f(x_2), \cdots, f(x_n)),$$

where $f(x) = a_k x^{k-1} + a_{k-1}x^{k-2} + \cdots + a_1 \in \mathbf{F}_q[x]$ and $\{x_1, x_2, \cdots, x_n\} \subseteq \mathbf{F}_q$ is called the evaluation set. (Note that different encoding schemes are possible.)

It is not difficult to see that the set of codewords formed in this manner is a linear subspace of $\mathbf{F}_q^n$ which has dimension k. Generalized Reed-Solomon codes

* This research is partially supported by NSF Career Award CCR-0237845.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 296–305, 2007.
© Springer-Verlag Berlin Heidelberg 2007

are therefore *linear codes*, because they are linear subspaces of $\mathbf{F}_q^n$, where n is the length of a codeword.

The *Hamming distance* between two codewords is the number of coordinates in which they differ – or one can think of this as the number of modifications required to transform one vector into another. A *Hamming ball of radius* m is a set of vectors within Hamming distance m to some vector in $\mathbf{F}_q^n$. The minimum distance of a code is the smallest distance between any two distinct codewords, and is a measure of how many errors the code can correct or detect. The *covering radius of a code* is the maximum possible distance from any vector in $\mathbf{F}_q^n$ to the closest codeword. A *deep hole* is a vector which achieves this maximum. The minimum distance of generalized Reed-Solomon codes is $n-k+1$. The covering radius of generalized Reed-Solomon codes is $n-k$.

A code is useless without a *decoding algorithm*, which takes some *received word* (a vector in $\mathbf{F}_q^n$) and outputs a message. The message should correspond, ideally, to the codeword which is closest, with respect to Hamming distance, to the received word. If we assume that each coordinate in a received word is equally likely to be in error, then the closest codeword is the most likely to be the intended transmission.

Standard Reed-Solomon codes use $\mathbf{F}_q^*$ as their evaluation set. If the evaluation set is $\mathbf{F}_q$, then the code is called an *extended Reed-Solomon code*. If the evaluation set is the set of rational points in a projective line over $\mathbf{F}_q$, then the code is known as a *doubly extended Reed-Solomon code*. The difference between standard Reed-Solomon codes, extended Reed-Solomon codes and doubly extended Reed-Solomon codes is not significant, but generalized Reed-Solomon codes are quite unique, as the evaluation set can be exponentially larger than the length of a codeword.

1.1 Related Work

The pursuit of efficient decoding algorithms for Reed-Solomon codes has yielded intriguing results. If the radius of a Hamming ball centered at some received word is less than half the minimum distance, there can be at most one codeword in the Hamming ball. Finding this codeword is called *unambiguous decoding*. It can be efficiently solved, see [1] for a simple algorithm.

If the radius is less than $n-\sqrt{n(k-1)}$, the problem can be solved by the Guruswami-Sudan algorithm [6], which outputs all the codewords inside a Hamming ball. If the radius is stretched further, the number of codewords in a Hamming ball may be exponential. We then study *the bounded distance decoding problem*, which outputs just one codeword in any Hamming ball of a certain radius. More importantly, we can remove the restriction on radius and investigate the *maximum likelihood decoding problem*, which is the problem of computing the closest codeword to any given vector in $\mathbf{F}_q^n$.

The question on decodability of Reed-Solomon codes has attracted attention recently, due to recent discoveries on the relationship between decoding Reed-Solomon codes and some number theoretical problems. Allowing exponential alphabets, Guruswami and Vardy proved that the maximum likelihood decoding

is NP-complete. They essentially showed that deciding deep holes is co-NP-complete. When the evaluation set is precisely the whole field or $\mathbf{F}_q^*$, an NP-completeness result is hard to obtain, Cheng and Wan [3] managed to prove that the decoding problem of Reed-Solomon codes at certain radius is at least as hard as the discrete logarithm problem over finite fields. In this paper, we wish to establish an additional connection between decoding of standard Reed-Solomon codes and a classical number-theoretic problem – that of determining the number of rational points on an algebraic hypersurface.

1.2 Our Results

The decoding problem of Reed-Solomon codes can be reformulated into the problem of *curve fitting* or *noisy polynomial reconstruction*. In this problem, we are given n points

$$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$$

in $\mathbf{F}_q^2$. The goal is to find polynomials of degree $k-1$ that pass as many of the n points as possible. Note that all the x-coordinates are distinct.

Given the received word $w = (y_1, y_2, \cdots, y_n)$, we are particularly interested in the polynomial obtained by interpolating the n points.

$$\begin{aligned} w(x) = y_1 &\frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)} \\ &+\cdots+ y_i \frac{(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_1-x_n)} \\ &+\cdots+ y_n \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\cdots(x_n-x_{n-1})}. \end{aligned}$$

In this paper, we say that a polynomial $w(x)$ *generates* a vector $w \in \mathbf{F}_q^n$ if

$$w = (w(x_1), w(x_2), \cdots, w(x_n)).$$

If the polynomial $w(x)$ has degree $k-1$ or less, w must be a codeword, and vice versa (since codewords consist of the encodings of all messages of length k). If $w(x)$ has degree k, w must be a deep hole (as we will later show). What if it has degree larger than k? Can it be a deep hole?

In this paper, we try to answer this question. If a received word is a deep hole, there is no codeword which is at distance $n-k-1$ or closer to the received word. Hence if the distance bound is $n-k-1$, a decoding algorithm can tell a received word is deep hole or not by checking whether there is a codeword in the Hamming ball of radius $n-k-1$. This shows that maximum likelihood decoding of Reed-Solomon codes, as well as the bounded distance decoding at radius $n-k-1$, is at least as hard as deciding deep holes. Observe that the bounded distance decoding at a distance of $n-k$ or more can be done efficiently. It is hoped that we can decrease the radius until we reach the domain of hard problems.

We are mainly concerned with the case when the evaluation set consists of nonzero elements of the field. Notice that for generalized Reed-Solomon code, the bounded distance decoding at distance $n-k-1$ is NP-hard. We reduce the problem to deciding whether an absolutely irreducible hypersurface contains a rational point whose coordinates are pairwise distinct and nonzero. From the reduction, we show that if k and the degree of $w(x)$ are small, $w(x)$ cannot generate deep holes. More precisely

Theorem 1. *Let p be a prime and $1 < k < p^{1/4-\epsilon}$ be a positive integer. The vector $(w(\alpha))_{\alpha\in\mathbf{F}_p}$ is not a deep hole in the Reed-Solomon code $[p,k]_p$ if the degree of $w(x)$ is greater than k but less than $k+p^{3/13-\epsilon}$.*

Roughly speaking, the theorem indicates that a vector generated by a low degree polynomial can not be a deep hole, even though it is very far away from any codeword.

To prove the theorem, we need to estimate the number of rational points on an algebraic hypersurface over a finite field. This problem is one of the central problems in algebraic geometry and finite field theory. Weil, through his proof of the Riemann Hypothesis for function fields, provided a bound for the number of points on algebraic curves. This bound was later generalized by Weil and Lang to algebraic varieties. Schmidt [8] obtained some better bounds for absolutely irreducible hypersurfaces by elementary means. In this paper, we will use an improved bound, obtained by Cafure and Matera [2] very recently. But first we give a new proof that deciding whether or not a received word is a deep hole is co-NP-complete. Our reduction is straight-forward and much simpler that the one constructed by Guruswami and Vardy.

2 A Simple Proof That the Maximum Likelihood Decoding Is NP-Complete

We reduce the following finite field subset sum problem [5, Page 233] to deep hole problem of generalized Reed-Solomon codes.

Instance: A set of n elements $A = \{x_1, x_2, x_3, \cdots, x_n\} \subseteq \mathbf{F}_{2^m}$, an element $b \in \mathbf{F}_{2^m}$ and a positive integer $k < n$.

Question: Is there a nonempty subset $\{x_{i_1}, x_{i_2}, \cdots, x_{i_{k+1}}\} \subseteq A$ of cardinality $k+1$ such that

$$x_{i_1} + x_{i_2} + \cdots + x_{i_{k+1}} = b.$$

Now consider the generalized Reed-Solomon code $[n,k]_{2^m}$ with evaluation set A. Suppose we have a received word

$$w = (f(x_1), f(x_2), \cdots, f(x_n))$$

where $f(x) = x^{k+1} - bx^k$. If the word is not a deep hole, it is at most $n-k-1$ away from a codeword. Suppose that the codeword is generated by $t(x)$. Then

$t(x)$ has degree at most $k-1$ and $f(x)-t(x)$ has at least $k+1$ distinct roots in A. Since $f(x)-t(x)$ is a monic polynomial with degree $k+1$, we have

$$f(x)-t(x) = x^{k+1} - bx^k - t(x) = (x-x_{i_1})(x-x_{i_2})\cdots(x-x_{i_{k+1}}),$$

for some $x_{i_1}, x_{i_2}, \cdots, x_{i_{k+1}}$ in A. Therefore $x_{i_1} + x_{i_2} + \cdots + x_{i_{k+1}} = b$.

On the other hand, if $x_{i_1}+x_{i_2}+\cdots+x_{i_{k+1}} = b$, $f(x)-(x-x_{i_1})(x-x_{i_2})\cdots(x-x_{i_{k+1}})$ generates a codeword. It shares $k+1$ values with w, thus has distance less than $n-k-1$ away from a codeword, so it cannot be a deep hole.

In summary, w is not a deep hole if and only if the answer to the instance of the finite field subset sum problem is "Yes". Hence the deep hole problem is co-NP-complete.

By a similar argument, we know that the polynomials of degree k must generate deep holes. Hence

Corollary 1. *For a generalized Reed-Solomon code $[n,k]_q$, there are at least $(q-1)q^k$ many deep holes.*

We remark that the argument cannot work for small evaluation sets, because the subset sum is easy in that case. In fact, if the evaluation set is the whole field and $k>1$, then a polynomial of degree $k+1$ cannot generate a deep hole.

3 A Hypersurface Related to Deep Holes

The above argument motivates us to consider vectors generated by polynomials of larger degree. We are given some received word w and we want to know whether or not it is a deep hole. The received word is generated by $w(x)$ of the form $f(x)+t(x)$ where $f(x)$ is some polynomial containing no terms of degree smaller than k, and $t(x)$ is some polynomial containing only terms of degree $k-1$ or smaller. For purposes of determining whether or not w is a deep hole, we fix a monic $f(x)$

$$f(x) = x^{k+d} + f_{d-1}x^{k+d-1} + \cdots + f_0 x^k \in \mathbf{F}_q[x]$$

and let $t(x)$ vary and ask whether $f(x)+t(x)$ has $k+1$ roots, or perhaps more.

In its essence, the question is one of finding a polynomial whose leading terms are $f(x)$, and which has as many zeroes as possible over a certain field. (As stated earlier, for $k>1$, if $f(x)$ has degree k, then w is a deep hole. If $f(x)$ has degree $k+1$, then it is not a deep hole.)

The most obvious way to approach this problem is to symbolically divide $f(x)$ by a polynomial that is the product of $k+1$ distinct (but unknown) linear factors, and determine whether or not it is possible to set the the leading term of the remainder, i.e., the coefficient of x^k, equal to zero. If the leading coefficient is 0, the remainder has degree $k-1$ or less in x, which generates a codeword. The distance between the codeword and w will be at most $n-k-1$.

A polynomial that is the product of $k+1$ distinct linear factors will have the elementary symmetric polynomials as coefficients,

$$\Pi = (x - x_1)(x - x_2)...(x - x_{k+1}) = x^{k+1} + \pi_1 x^k + \pi_2 x^{k-1} + ... + \pi_{k+1},$$

where π_i is the i-th symmetric polynomial in $x_1, x_2, \cdots, x_{k+1}$.

Since Π is monic, the remainder of the division of $f(x)$ by Π will be a polynomial in $\mathbf{F}_q[x_1, x_2, \cdots, x_{k+1}][x]$ of degree less than $k+1$. Denote the leading coefficient of the remainder by $L_{f_0,f_1,\cdots,f_{d-1}}(x_1, x_2, \cdots, x_{k+1})$. This is a multivariate polynomial of degree d.

As an example, imagine the division of the polynomial x^{k+1} by Π. We can easily verify that the leading term of the remainder is $-\pi_1 x^k$. Since we can always find $k+1$ distinct values that will satisfy $\pi_1 = 0$, we know that x^{k+1} cannot be a deep hole. But, in most cases $w(x)$ will have larger degree and contain many terms, and the remainder will be a more complex multivariate polynomial in $k+1$ variables, rather than a linear polynomial in $k+1$ variables. If the leading coefficient itself has a solution where all roots are distinct and invertible, then $f(x) + t(x)$ cannot generate a deep hole.

We now argue that the leading coefficient of the remainder is absolutely irreducible. We write

$$L_{f_0,f_1,\cdots,f_{d-1}}(x_1, x_2, \cdots, x_{k+1}) = F_d + F_{d-1} + \cdots + F_0,$$

where F_i is a form containing all the terms of degree i in L. The polynomial $L_{f_0,f_1,\cdots,f_{d-1}}(x_1, x_2, \cdots, x_{k+1})$ is absolutely irreducible if F_d is absolutely irreducible. To see this, suppose that L can be factored as $L'L''$. Let F'_{d_1} be the form of highest degree in L' and F''_{d_2} be the form of highest degree in L''. Then we have $F_d = F'_{d_1} F''_{d_2}$, a contradiction to the condition that F_d is absolutely irreducible.

Fortunately F_d does not depend on f_i's.

Lemma 1. *The form of the highest degree in* $L_{f_0,f_1,\cdots,f_{c-1}}(x_1, x_2, \cdots, x_{k+1})$ *is exactly* $L_{0,0,\cdots,0}(x_1, x_2, \cdots, x_{k+1})$.

Proof. It can be proved by mathematical induction on c.

In the next section, we argue that the term of highest degree in the leading coefficient, which we will call $\chi_d(x_1, x_2, ...x_{k+1})$, is absolutely irreducible. We will actually show that $\chi_d(x_1, x_2, 1, 0, 0...0)$ is absolutely irreducible. This is because that $\chi_d(x_1, x_2, 1, 0, 0...0)$ has the same degree as $\chi_d(x_1, x_2, ...x_{k+1})$, if the former is irreducible, so is the latter.

Lemma 2. $\chi_d(x_1, x_2, 1, 0, 0...0) = \Sigma_{i+j \le d} {x_1}^i {x_2}^j$.

Proof. We need to compute the leading coefficient of the remainder after dividing x^{k+d} by $(x - x_1)(x - x_2)(x - 1)x^{k-2}$. It is as same as the leading coefficient of the remainder after dividing x^{d+2} by $(x - x_1)(x - x_2)(x - 1)$. The remainder is a quadratic polynomial in x. When we evaluate it at x_1, it takes the value x_1^{d+2}. When we evaluate it at x_2, it takes the value x_2^{d+2}. When we evaluate it at 1, it takes value 1. By interpolating, we obtain the unique polynomial satisfying these conditions. It is

$$x_1^{d+2}\frac{(x-x_2)(x-1)}{(x_1-x_2)(x_1-1)}+x_2^{d+2}\frac{(x-x_1)(x-1)}{(x_2-x_1)(x_2-1)}+\frac{(x-x_1)(x-x_2)}{(1-x_1)(1-x_2)}$$

The leading coefficient is

$$\frac{x_1^{d+2}}{(x_1-x_2)(x_1-1)}+\frac{x_2^{d+2}}{(x_2-x_1)(x_2-1)}+\frac{1}{(1-x_1)(1-x_2)},$$

which is equal to $\Sigma_{i+j\le d}{x_1}^i{x_2}^j$.

4 A Smooth Curve

The section is devoted to the proof of the irreducibility of the bivariate polynomial $\Sigma_{i+j\le d}x^iy^j$ over $\mathbf{F}_p$.

Lemma 3. *Let p be a prime and $d<p-2$ be a positive integer. The curve $f(x,y)=\Sigma_{i+j\le d}x^iy^j=0$ is absolutely irreducible over $\mathbf{F}_p$.*

Proof. To show that $f(x,y)=\Sigma_{i+j\le d}x^iy^j$ is absolutely irreducible, we actually prove a stronger statement that $f(x,y)=0$ is a smooth algebraic curve. It can be verified by simple calculation that places on the curve at infinity are nonsingular. Hence it is sufficient to show that all the affine places on the curve are nonsingular, i.e. that the system of equations

$$\begin{cases} f(x,y)=0\\ \frac{\partial f}{\partial x}=0\\ \frac{\partial f}{\partial y}=0\end{cases}$$

has no solution.

First, it is convenient to write $f(x,y)$ as

$$x^d+(y+1)x^{d-1}+(y^2+y+1)x^{d-2}+\cdots+(y^d+...+1).$$

We write $\frac{\partial f}{\partial x}$ as:

$$dx^{d-1}+(d-1)(y+1)x^{d-2}+...+(y^{d-1}+\cdots+1)$$

and $\frac{\partial f}{\partial y}$ as

$$dy^{d-1}+(d-1)(x+1)y^{d-2}+...+(x^{d-1}+\cdots+1)$$

Assume that there is a solution (X,Y) to the system of equations. Compute $(x-y)\frac{\partial f}{\partial x}$:

$$\begin{aligned} x\frac{\partial f}{\partial x}&=dx^d+(d-1)(y+1)x^{d-1}+...+x(y^{d-1}+...1)\\ &=(d+1)x^d+d(y+1)x^{d-1}+\cdots+(y^d+y^{d-1}+\cdots+1)-f(x,y)\\ y\frac{\partial f}{\partial x}&=dyx^{d-1}+(d-1)(y^2+y)x^{d-2}+...+(y^d+...1)\end{aligned}$$

Their difference is:

$$\begin{aligned}(x-y)\frac{\partial f}{\partial x} &= dx^d + [d-(y+1)]x^{d-1} + ... - (y^d + ... + 1) \\ &= (d+1)x^d + dx^{d-1} + (d-1)x^{d-2} + \cdots + 1 - f(x,y).\end{aligned}$$

Since $f(X,Y)$ must be zero, we know that:

$$(d+1)X^d + dX^{d-1} + (d-1)X^{d-2} + \cdots + 1 = 0$$

we multiply by the above X, and then subtract the original expression to get:

$$(d+1)X^{d+1} = X^d + X^{d-1} + \cdots + 1. \tag{1}$$

Repeat the process on $\frac{\partial f}{\partial y}$, we get

$$(d+1)Y^{d+1} = Y^d + Y^{d-1} + \cdots + 1. \tag{2}$$

This shows that neither X nor Y can be zero. Now, observe that

$$(x-y)f(x,y) = x^{d+1} + x^d + \cdots + 1 - y^{d+1} - y^d - \cdots - 1 = 0.$$

This means that $(d+2)X^{d+1} = (d+2)Y^{d+1}$. We also know that the right hand sides of (1) and (2) are actually $\frac{X^{d+1}-1}{X-1}$, and $\frac{Y^{d+1}-1}{Y-1}$. So multiplying both sides by $X-1$ for 1 and by $Y-1$ for 2, we obtain

$$(d+1)X^{d+2} - (d+2)X^{d+1} = -1$$

$$(d+1)Y^{d+2} - (d+2)Y^{d+1} = -1.$$

Hence we have $(d+1)X^{d+2} = (d+1)Y^{d+2}$.

Since $d+2 < p$, this means that the characteristic of the field does not divide either $d+1$ or $d+2$. We have $X^{d+2} = Y^{d+2}$ and $X^{d+1} = Y^{d+1}$. Therefore $X = Y$.

At the point (X,Y), $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ have the same form

$$(d + (d-1) + \cdots + 1)X^{d-1} + ((d-1) + (d-2) + \cdots)X^{d-2} + \cdots + 1 = 0.$$

And $f(X,Y)$ can be written (since $X = Y$) as

$$(d+1)X^d + dX^{d-1} + (d-1)X^{d-2} + \cdots + 1 = 0.$$

If we subtract $f(X,Y)$ from $\frac{\partial f}{\partial x}(X,Y)$, we get

$$(d+1)X^d - ((d-1) + (d-2) + \cdots + 1)X^{d-1} - \cdots - X = 0.$$

Divide the result by X, we have

$$\begin{aligned}&(d+1)X^{d-1} - ((d-1) + (d-2) + \cdots + 1)X^{d-2} - \cdots - 1 \\ &= (d+1)X^{d-1} + (d + (d-1) + \cdots + 1)X^{d-1} \\ &= 0.\end{aligned}$$

This means that $\frac{(d+1)(d+2)}{2}X^{d-1} = 0$, hence $X = 0$, and this is a contradiction.

5 Estimation of Number of Rational Points

Cafure and Matera [2, Theorem 5.2] have obtained the following estimation of number of rational points on an absolutely irreducible hypersurface:

Proposition 1. *An absolutely irreducible* $\mathbf{F}_q$*-hypersurface of degree* d *in* $\mathbf{F}_q^n$ *contains at least*

$$q^{n-1} - (d-1)(d-2)q^{n-3/2} - 5d^{13/3}q^{n-2}$$

many $\mathbf{F}_q$*-rational points.*

We also use the following proposition [2, Lemma 2.2].

Proposition 2. *Suppose* $f_1(x_1, x_2, \cdots, x_n)$ *and* $f_2(x_1, x_2, \cdots, x_n)$ *are polynomials of degree not greater than* d*, and they donot have a common factor. Then the number of* $\mathbf{F}_q$*-rational solutions of*

$$f_1 = f_2 = 0$$

is at most d^2q^{n-2}*.*

We seek solutions of $L(x_1, x_2, \cdots, x_{k+1})$ but not of

$$\Pi_{1\le i\le k+1}x_i\Pi_{1\le i<j\le k+1}(x_i - x_j) = 0.$$

We count the number of rational solutions of $L(x_1, x_2, \cdots, x_{k+1})$, minus the number of rational solutions of

$$\begin{cases} L(x_1, x_2, \cdots, x_{k+1})=0 \\ \Pi_{1\le i\le k+1}x_i\Pi_{1\le i<j\le k+1}(x_i - x_j)=0 \end{cases}$$

The number is greater than

$$p^k - (d-1)(d-2)p^{k-1/2} - 5d^{13/3}p^{k-1} - [\max(d, \frac{k^2+k+2}{2})]^2p^{k-1},$$

which is greater than 0 if $d < p^{3/13-\epsilon}$ and $k < p^{1/4-\epsilon}$. This concludes the proof of the main theorem.

6 Concluding Remarks

It has been proved that for generalized Reed-Solomon codes, the bounded distance decoding at radius $n-k-1$ is NP-hard. In this paper, we try to determine the complexity of this problem for standard Reed-Solomon codes. We reduce the problem to a problem of determining whether a hypersurface contains a rational point of distinct coordinates. While we didnot solve the problem completely, we show that for prime fields and small k, this problem is easy if a vector is generated by a small degree polynomial. In essence, we ask whether there exists

a polynomial with many distinct rational roots under the restriction that some coefficients are prefixed. This problem bears an interesting comparison with the active research [4] on the construction of irreducible polynomial with some prefixed coefficients.

To solve the problem for every k and every vector, there are two apparent approaches: 1) Find a better estimation of number of rational points on the hypersurfaces. 2) Explore the specialty of the hypersurfaces. From an average argument, it is tempting to conjecture that the vectors generated by polynomials of degree k are the only possible deep holes. If so, we can completely classify deep holes. We leave it as a open problem.

References

1. E. Berlekamp and L. Welch. Error correction of algebraic block codes. U.S. Patent Number 4633470, 1986.
2. A. Cafure and G. Matera. Improved explicit estimates on the number of solutions of equations over a finite field. http://www.arxiv.org/abs/math.NT/0405302, 2004.
3. Qi Cheng and Daqing Wan. On the list and bounded distance decodibility of the Reed-Solomon codes (extended abstract) (FOCS). In *Proc. 45th IEEE Symp. on Foundations of Comp. Science*, pages 335–341, 2004.
4. Stephen D. Cohen. Explicit theorems on generator polynomials. *Finite Fields and Their Applications*, 2005. To appear.
5. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
6. Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6): 1757–1767, 1999.
7. Venkatesan Guruswami and Alexander Vardy. Maximum-likelihood decoding of Reed-Solomon codes is NP-hard. In *Proceeding of SODA*, 2005.
8. W. Schmidt. *Equations over Finite Fields. An Elementary Approach*, volume 536 of *Lecture Notes in Mathematics*. Springer-Verlag, 1976.

Quantum Multiparty Communication Complexity and Circuit Lower Bounds

Iordanis Kerenidis[1,2,⋆]

[1] CNRS
[2] LRI-Univ. de Paris-Sud

Abstract. We define a quantum model for multiparty communication complexity and prove a simulation theorem between the classical and quantum models. As a result, we show that if the quantum k-party communication complexity of a function f is $\Omega(\frac{n}{2^k})$, then its classical k-party communication is $\Omega(\frac{n}{2^{k/2}})$. Finding such an f would allow us to prove strong classical lower bounds for $k \geq \log n$ players and make progress towards solving a main open question about symmetric circuits.

1 Introduction

Communication complexity is a central model of computation with numerous applications. It has been used for proving lower bounds in many areas including Boolean circuits, time-space tradeoffs, data structures, automata, formulae size, etc. Examples of these applications can be found in the textbook of Kushilevitz and Nisan [12].

The "Number on the Forehead" (NoF) model of multiparty communication complexity was introduced by Chandra, Furst and Lipton [7]. In this model, there are k parties that wish to compute a function $f : X_1 \times \cdots \times X_k \to \{0,1\}$ on the input $(x_1, \ldots, x_k) \in (X_1 \times \cdots \times X_k)$. We can assume w.l.o.g. that $X_1 = \ldots = X_k = \{0,1\}^n$. Each player sees only $(k-1)$ of the inputs (the other one is on his forehead). The players communicate by writing messages on a common blackboard. In the general model, in every round, the players take turns writing one bit on the blackboard that might depend on the previous messages. In the Simultaneous Messages variant (SMNoF), all players write simultaneously a single message on the blackboard. At the end of the protocol, the blackboard must contain enough information to compute the value of $f(x_1, \ldots, x_k)$. The communication cost of the protocol is the number of bits written on the blackboard. The deterministic k-party communication complexity of f, $C(f)$, is the communication cost of the optimal deterministic protocol for f. In the randomized setting, we allow the players to be probabilistic, share public coins and the output of the protocol to be correct with probability at least $1/2 + \delta$. We define $C_\delta(f)$ to be the probabilistic k-party communication complexity of f with correctness $1/2 + \delta$.

⋆ Supported by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as Contract Number 015848.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 306–317, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In the above definition the number of players was equal to the number of arguments of f. However, we can easily generalize the model for the case of $\ell \leq k$ players. The model of communication remains the same and each of the ℓ players still receives $(k-1)$ arguments of f. We denote with $C_\delta^\ell(f)$ the ℓ-party communication complexity of $f(X_1, \ldots, X_k)$.

Multiparty communication complexity has been studied extensively and has proved relevant to important questions in circuit lower bounds. For example, one of the major open problems in circuit complexity is to prove that an explicit function f is not in the circuit complexity class ACC^0, which is defined in the next subsection (see [12],Open problem 6.21). By the results of [10,18], this question reduces to proving a superlogarithmic communication lower bound for the k-party communication complexity of some explicit function f, where the number of players is superlogarithmic. However, all known techniques for proving multiparty communication lower bounds fail when the number of players becomes $k = \log n$.

In this paper, we propose a new technique for proving multiparty communication complexity lower bounds and hence, circuit lower bounds. We define a quantum model for multiparty communication complexity, where both the players' inputs and messages are quantum, and prove a simulation theorem between the classical and quantum models. More specifically, we show how to simulate k classical players with only $k/2$ quantum ones. This enables us to reduce questions about classical communication to potentially easier questions about quantum communication complexity and shows that quantum information theory could be a powerful tool for proving classical circuit lower bounds. Similar connections between classical and quantum computation have been proved to be very fruitful in the last few years ([11,17,1,2]).

1.1 Multiparty Communication Complexity and Circuit Lower Bounds

Multiparty communication complexity was introduced as a tool for the study of boolean circuits, however the known techniques for proving lower bounds are very limited. Babai et al. [3] have proved a lower bound of $\Omega(\frac{n}{2^{2k}} + \log \delta)$ for the k-party communication complexity of the Generalized Inner Product function. Raz [14] simplified their proof technique and showed a similar lower bound for another function, namely Matrix Multiplication, which seems to be hard even for $\log n$ players. Unfortunately, the above techniques are limited and cannot prove lower bounds better than $\Omega(\frac{n}{2^k} + \log \delta)$ for any function. Despite the importance of the question and its serious consequences on circuit lower bounds, it has not been possible to find any new lower bound techniques. For the Generalized Inner Product function, Grolmusz [9] showed an upper bound of $O(k\frac{n}{2^k} + \log \delta)$.

The Number on the Forehead model is related to the circuit complexity class ACC^0. ACC^0 consists of languages recognized by a family of constant-depth polynomial size, unbounded fan-in circuits with NOT, AND, OR and MOD_m gates, where m is fixed for the family. It is a major open question to find an explicit function outside the class ACC^0. Yao [18] and Beigel and Tarui [4] have

shown that ACC^0 circuits can be simulated by symmetric circuits. The circuit class $SYM(d, s)$ is the class of circuits of depth 2, whose top gate is a symmetric gate of fan-in s and each of the bottom level gates is an AND gate of fan-in at most d. Specifically, they showed that $ACC^0 \subseteq SYM(\text{polylog } n, 2^{\text{polylog } n})$. The connection to multiparty communication was made by Hastad and Goldmann [10], who noticed that when a function f belongs to $SYM(d, s)$, then there exists a $(d+1)$-party simultaneous protocol with complexity $O(d \log s)$. Hence, if we want to show that a function f is outside $SYM(d, s)$, then we need to prove a $(d + 1)$-party communication lower bound of $\omega(d \log s)$ in the simultaneous model. However, as we said, no techniques are known to give communication lower bounds for $k = \log n$ players or more. In the next sections we describe a technique that can potentially give strong lower bounds for $k \geq \log n$ players nad hopefully help towards proving that a function is outside ACC^0.

2 Quantum Multiparty Communication Complexity

We assume basic familiarity with the formalism of quantum computing and refer to [13] for further details. One natural way of defining the quantum analog of simultaneous multiparty communication would be the following: there are k parties that wish to compute a function $f : X_1 \times \cdots \times X_k \to \{0, 1\}$ on the input $(x_1, \ldots, x_k) \in X_1 \times \cdots \times X_k$. We assume w.l.o.g. that $X_1 = \ldots = X_k = \{0, 1\}^n$. Each player sees only $(k - 1)$ of the inputs (the other one is on his forehead). The players communicate by writing simultaneously a *quantum* message each on a common blackboard that they can all see. After that, the value of f can be computed with high probability by performing some measurement on these quantum messages. The quantum communication cost is the sum of the number of qubits of each message. In this model, we have kept the inputs to the players classical but made the communication quantum. Unfortunately, not very much is known about the power of this model of quantum multiparty communication. It is an open question to see if this model can be exponentially more powerful than the classical one and also what is its relation to our model.

Here, we define a different variant of quantum multiparty communication where, in addition, we allow the inputs to the quantum players to be quantum as well. Our primary goal is to define a natural model that has consequences to the study of circuit lower bounds. In order to make the definition of the quantum model more intuitive, we are going to describe the classical model of the Number on the Forehead in a different but equivalent way.

In high level, a simultaneous multiparty protocol consists of three rounds: first, the players receive their inputs; second they each output some answer that depends on their input and third, the value of f is computed as a function of the players' answers. For the rest of the paper, we assume the players are equivalent and their answers have the same size. We can achieve that by having each player play the role of each one of the ℓ players and hence increase the communication by just a factor of ℓ, which won't be of significance.

Conceptually, one can think of such a protocol as a circuit for the function f that consists of three subcircuit blocks, each corresponding to one round of the communication protocol and satisfies certain properties that are described below (see Fig. 1).

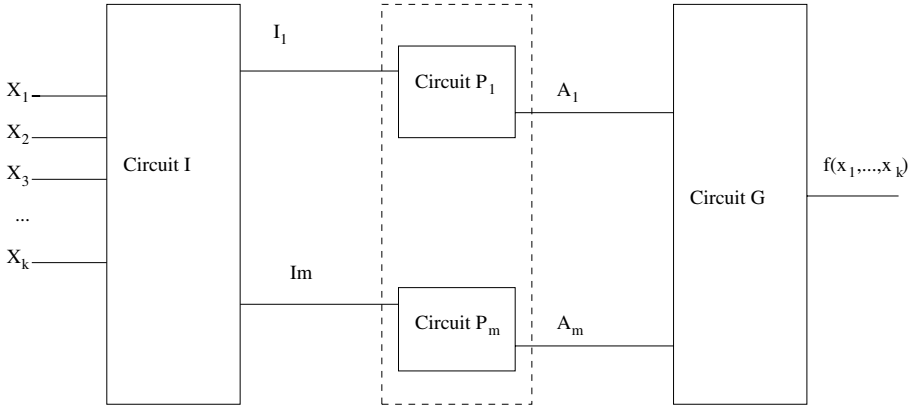


Fig. 1. Classical Number on the Forehead

Classical Simultaneous Number on the Forehead (SNoF)

- The first block I takes as input a string $(x_1, \ldots, x_k) \in (X_1 \times \cdots \times X_k)$ and produces the inputs $I_j = (x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k)$ for every subcircuit-player P_j.
- The second block consists of ℓ probabilistic subcircuits-players P_j, such that each one takes as input I_j and outputs an answer A_j.
- The third block consists of a subcircuit G that takes as input all the answers $(A_1, \ldots, A_\ell)$ and outputs $g(A_1, \ldots, A_\ell)$ as a guess for $f(x_1, \ldots, x_k)$. The function g is fixed in advance and is independent of the input $(x_1, \ldots, x_k)$.

The correctness of the protocol guarantees that for every input $(x_1, \ldots, x_k) \in \{0,1\}^{kn}$, $Pr[g(A_1, \ldots, A_\ell) = f(x_1, \ldots, x_k)] \geq 1/2 + \delta$, where the probability is over the random coins of the subcircuit-players P_j. The "communication cost" of the circuit-protocol is the sum of the lengths of the outputs of the players or equivalently the sum of the lengths of the inputs to the final subcircuit G, i.e. $\sum_{i=1}^{\ell} |A_i|$. The communication complexity of f is the cost of the optimal circuit-protocol. It's easy to see that the formulation described above is equivalent to the usual simultaneous Number on the Forehead model.

Intuitively, we define the quantum analog by allowing the circuit to be quantum (see Fig. 2). One has to be careful though with the constraints one needs to impose on the circuit and the definition of the "cost" of the circuit.

In high level, the first block of the quantum circuit-protocol takes as input the classical string $(x_1, \ldots, x_k) \in \{0,1\}^{kn}$ and creates quantum inputs ρ_j for the subcircuits-players. We put restrictions on the legal quantum inputs to ensure

that each quantum player obtains information for at most $(k-1)$ of the inputs x_i, exactly like the classical players.

The second block consists of ℓ quantum subcircuits-players P_j that take these inputs and output some quantum answers.

The third block consists of a general measurement that produces a guess for the value of the function f. In order to ensure that the measurement is independent of the specific input $(x_1, \ldots, x_k)$ we first quantumly "erase" the registers that contain the inputs I_j and then perform a measurement on the remaining state. More formally, the quantum model is defined as follows:

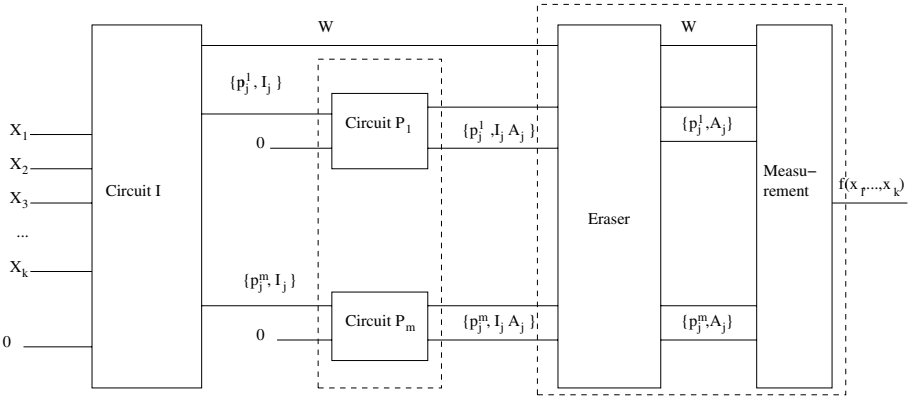


Fig. 2. Quantum Simultaneous Number on the Forehead

Quantum Simultaneous Number on the Forehead

- The subcircuit I takes as input a string $(x_1, \ldots, x_k) \in (X_1 \times \cdots \times X_k)$ and some ancilla and performs the following operation for $i = 1, \ldots, \ell$:
  It constructs a state $\sum_{j=1}^{k} \sqrt{p_j^i}|j\rangle$ and performs a mapping T

$$T: \ |j\rangle|0\rangle \mapsto |j\rangle|j, I_j\rangle$$

  resulting in the state

$$|\phi_i\rangle = \sum_{j=1}^{k} \sqrt{p_j^i}|j\rangle|j, I_j\rangle.$$

  The second register of this state contains the input state for the subcircuit-player P_i ($\rho_i = \{p_j^i, (j, I_j)\}$) and the first register contains the purification of this mixed state. The distribution is fixed by the protocol and is independent of the input $(x_1, \ldots, x_k)$.
- The second block consists of ℓ subcircuits-players that perform the following mapping:

$$|j, I_j\rangle|0\rangle \mapsto |j, I_j\rangle|A_j^i\rangle$$

  where A_j^i is the quantum answer of player i to input I_j.

– The third round takes as input the quantum states

$$|\psi_i\rangle = \sum_{j=1}^{k} \sqrt{p_j^i}|j\rangle|j, I_j\rangle|A_j^i\rangle.$$

In order to ensure that the measurement doesn't take advantage of the fact that the second registers contain the input of the function , the circuit first performs the inverse mapping T^{-1}, i.e.

$$T^{-1}: \ |j\rangle|j, I_j\rangle \mapsto |j\rangle|0\rangle,$$

resulting in the states

$$|\psi_i\rangle = \sum_{j=1}^{k} \sqrt{p_j^i}|j\rangle|A_j^i\rangle.$$

Then, a general measurement M is performed on these states, whose outcome is the guess for $f(x_1, \ldots, x_k)$. The measurement M is fixed by the protocol and is independent of the input x.

The correctness of the protocol implies that for all $(x_1, \ldots, x_k) \in \{0,1\}^{kn}$,

$$Pr[\text{outcome of } M = f(x_1, \ldots, x_k)] \geq 1/2 + \delta.$$

The communication cost of the protocol is the sum of the lengths of the inputs to the final measurement M, i.e. $\sum_{i=1}^{\ell} |A_i|$, where $|A_i|$ is the size of the answer register of player i.[1] The communication complexity of f is the cost of the optimal protocol.

The above model is a natural generalization of the classical model, in the case where we want to allow the players to receive quantum inputs. The inputs $\{p_j^i, (j, I_j)\}$ ensure that each player gains information for $(k-1)$ of the inputs x_i. In the special case where the distributions $\{p^i\}$ are delta functions, then the inputs become equal to the classical inputs. On the other hand, a more "'quantum"' input of the form $\sum_{j=1}^{k} |I_j\rangle$ would enable the quantum player to learn $(k-1)$ arbitrary bits of information about $(x_1, \ldots, x_k)$.

In the case where the inputs were classical, one could immediately ignore (trace out) the registers that contain the inputs to the players and perform the final operation just on the players' answers. On the other hand, when the inputs are quantum, we need to be more careful. For example, if we just ignore the input registers, then we just reduce our model to the one with classical inputs. The quantum "erasure" of the inputs in the third stage of the circuit ensures that the inputs can be quantum and that the final measurement only depends on the players' answers. The "cost" of the circuit is defined as the size of the state on which we perform the final measurement that yields the value of f.

[1] More precisely, the communication should be defined as $\sum_{i=1}^{\ell}(|W_i|+|A_i|)$, where W_i is the Hilbert space that contains the purification of the input of player i. However the communication according to this definition is in the worst case an additive factor of $\ell \log k$ greater than our definition which will not be of any significance.

3 Simulating Classical Players

In this section, we prove that we can simulate a k-party classical protocol by a $k/2$-party quantum protocol with the same communication, albeit with larger error probability. In what follows we mostly use the notation of communication complexity (protocols, players) than the equivalent one of circuits.

Theorem 1. *Let P be a SNoF protocol for the function $f : X_1, \ldots, X_k \to \{0,1\}$ with k players, communication C and correctness $1/2 + \delta$. Then, there exists a quantum SNoF protocol Q for the same function f with $k/2$ quantum players, communication $C/2$ and correctness $1/2 + \delta/2^C$ on an average input.*

Proof. First, we prove a lemma similar to Lemma 2 in [11], which shows that we can assume the players compute the parity of a subset of the answer bits as their guess for f. We switch from the $\{0,1\}$-notation to the $\{-1,1\}$-notation for f, we view the answers of the players A_i as $\frac{C}{k}$-bit strings and $A_i[j]$ the j-th bit of the string A_i. We denote by $S_i \subseteq [\frac{C}{k}]$ the possible subsets of bits of A_i and $A_{S_i} = \prod_{j \in S_1} A_{S_i}[j]$ is the parity of the subset S_i of the bits of A_i.

Lemma 1. *Let P be a classical protocol with communication C and correctness probability $1/2+\delta$ and assume that the players compute a function $g(A_1, \ldots, A_k)$ as their guess for $f(x)$, where A_i is the answer of player i. Then, there exists a classical protocol P' with communication C that works on average input with correctness $1/2 + \delta/2^{C/2}$ and where the players compute a parity of a subset of bits of the answers A_i, i.e. $g(A_1, \ldots, A_k) = \oplus_{i=1}^k A_{S_i}$.*

Proof. Let $f(x) = b$. From the correctness of the protocol P we know that $E_x[g(A_1, \ldots, A_k) \cdot b] \geq 2\delta$. We can represent g by its Fourier representation as

$$g(A_1, \ldots, A_k) = \sum_{S_1, \ldots, S_k} \hat{g}_{S_1, \ldots, S_k} A_{S_1} \cdots A_{S_k}$$

and have

$$2\delta \leq E_x[g(A_1, \ldots, A_k) \cdot b] = \sum_{S_1, \ldots, S_k} \hat{g}_{S_1, \ldots, S_k} E_x[A_{S_1} \cdots A_{S_k} \cdot b]$$

By the fact that $\sum_{S_1, \ldots, S_k} (\hat{g}_{S_1, \ldots, S_k})^2 = 1$ we have $\sum_{S_1, \ldots, S_k} \hat{g}_{S_1, \ldots, S_k} \leq 2^{C/2}$ and hence there exist some subsets $S_1, \ldots, S_k$ for which

$$E_x[A_{S_1} \cdots A_{S_k} \cdot b] \geq 2\delta/2^{C/2}.$$

This means that the protocol P' which would output the XOR of these subsets is correct on an average input with probability $\geq 1/2 + \delta/2^{C/2}$.

Hence, in the classical protocol P', in the first round each player j receives input I_j, in the second round they output the answers A_j and in the third round, the guess for f is computed by taking the XOR of a subset of the bits of the A_j's. Now we will describe the quantum circuit-protocol with only $k/2$ players that simulates the classical k-party one. We denote the $k/2$ quantum players with $i = 1, 3, \ldots, k-1$.

– In the first round the circuit I creates the following states:

$$|\phi_i\rangle = |i\rangle|i, I_i\rangle + |i+1\rangle|i+1, I_{i+1}\rangle,$$

where the second register is the input of quantum player i and the first one is the purification of the state in the workspace W. Note that the reduced density matrix of quantum player i is the same as if he was classical player i with probability $1/2$ and classical player $i+1$ with probability $1/2$. Hence, this is a legal input.

– In the second round, each subcircuit-player performs the following mapping:

$$T : |j, I_j\rangle|0\rangle \mapsto |j, I_j\rangle|A_j\rangle,$$

i.e. on input $|j, I_j\rangle$ computes the same function A_j as the classical player j in P'. Note that the answer of the classical player j can depend on his private randomness and we assume that the quantum player uses for each input (j, I_j) the same randomness used by the classical player j. The total communication is $\frac{k}{2}\frac{C}{k} = \frac{C}{2}$ qubits.

– In the third round, the states are

$$|\phi_i\rangle = |i\rangle|i, I_i\rangle|A_i\rangle + |i+1\rangle|i+1, I_{i+1}\rangle|A_{i+1}\rangle$$

First, the "erasure" circuit erases the input registers resulting in the states

$$|\psi_i\rangle = |i\rangle|A_i\rangle + |i+1\rangle|A_{i+1}\rangle.$$

Last, a measurement is performed on the states (described by Lemma 2) that computes f with high probability.

We need to show that there exists a quantum procedure M on the states $|\psi_i\rangle$ that is able to compute the function $\oplus_{i=1}^{k} S_i$. A key observation is that we can rewrite the function as

$$\oplus_{i=1}^{k} S_i = \oplus_{i=1,3,\ldots,k-1}(S_i \oplus S_{i+1}).$$

It's a simple calculation to show that if we can independently predict $S_i \oplus S_{i+1}$ with probability $1/2 + \epsilon$ then we can predict the entire $\oplus_i S_i$ with probability $1/2 + 2^{k/2-1}\epsilon^{k/2}$. The following lemma from [16] describes a quantum procedure M to compute $S_i \oplus S_{i+1}$ with the optimal ϵ.

Lemma 2. *(Theorem 2,[16]) Suppose $f : \{0,1\}^{2t} \to \{0,1\}$ is a boolean function. There exists a quantum procedure M to compute $f(a_0, a_1)$ with success probability $1/2 + 1/2^{t+1}$ using only one copy of $|0\rangle|a_0\rangle + |1\rangle|a_1\rangle$, with $a_0, a_1 \in \{0,1\}^t$.*

We use this lemma with $t = C/k$ and get $\epsilon = 1/2^{C/k+1}$. We also note that the success probability is independent of the a_0, a_1. Hence, there exists a quantum procedure that will output the correct $\oplus_i S_i$ with probability

$$Pr[M \text{ outputs } \oplus_i S_i] = \frac{1}{2} + 2^{k/2-1} \cdot \frac{1}{2^{(C+k)/2}} = \frac{1}{2} + \frac{1}{2^{C/2+1}}.$$

Finally, the quantum protocol is correct with probability

$$\begin{aligned} p &= Pr[M \text{ outputs } \oplus S_i] \cdot Pr[\oplus S_i = b] \\ &+ Pr[M \text{ doesn't output } \oplus S_i] \cdot Pr[\oplus S_i \neq b] \\ &= (\frac{1}{2} + \frac{1}{2^{C/2+1}})(\frac{1}{2} + \frac{\delta}{2^{C/2}}) + (\frac{1}{2} - \frac{1}{2^{C/2+1}})(\frac{1}{2} - \frac{\delta}{2^{C/2}}) = \frac{1}{2} + \frac{\delta}{2^C}. \end{aligned}$$

Note that the success probability of the quantum protocol is not guaranteed for every input but only on average input. In fact, it is easy to see that it works for any distribution on inputs, since Lemma 1 does not depend on the distribution of the input. Though proving lower bounds for such protocols can be potentially harder than proving lower bounds for worst-case protocols, most known lower bounds work equally for both cases.

4 A Quantum Reduction for Circuit Lower Bounds

The theorem in the previous section shows how to simulate a classical protocol with k players with a quantum protocol with $k/2$ players. We are going to use this theorem in order to get a reduction from a classical circuit lower bound question to one about quantum communication complexity.

Theorem 2. *Suppose $f : X_1 \times \cdots \times X_k \to \{0,1\}$ is a function for which the $(\frac{k}{2})$-party quantum average communication complexity is $QC_\delta^{k/2} = \Omega(k\frac{n}{2^{k/2}} + \log\delta)$. Then this function does not belong to the class $SYM(k-1, 2^{o(n/2^{k/2})})$.*

Proof. Let the function f have $(\frac{k}{2})$-party quantum communication complexity $QC_\delta^{k/2} \geq \gamma(k\frac{n}{2^{k/2}} + \log\delta)$, for a positive constant γ. Assume that the classical k-party communication complexity is $C_\delta \leq \frac{\gamma}{1+\gamma}(k\frac{n}{2^{k/2}} + \log\delta)$, then by Theorem 1 there exists an $(\frac{k}{2})$-party quantum protocol with correctness $1/2 + \delta/2^{C_\delta}$ and quantum communication $QC_{\delta/2^{C_\delta}}^{k/2} = \frac{C_\delta}{2}$. This contradicts the lower bound on QC since

$$QC_{\delta/2^{C_\delta}}^{k/2} \geq \gamma(k\frac{n}{2^{k/2}} + \log\frac{\delta}{2^{C_\delta}}) \geq \gamma(k\frac{n}{2^{k/2}} + \log\delta) - \gamma C_\delta > \frac{C_\delta}{2}$$

By [10] the function f does not belong to the class $SYM(k-1, 2^{o(n/2^{k/2})})$.

Taking $k = \log n + 1$, the function f is not in the class $SYM(\log n, 2^{o(\sqrt{n})})$. In other words, we reduced the question of finding a function outside the class $SYM(\log n, 2^{\omega(polylogn)})$ to that of finding an explicit function $f : X_1 \times \cdots \times X_k \to \{0,1\}$ with $(\frac{k}{2})$-party quantum complexity equal to $\Omega(\frac{n}{2^{k/2}} + \log\delta)$. Note that we do know explicit functions for which the classical communication is exactly of this form, e.g. the function Matrix Multiplication ([3,14]). In fact, the proofs given in these papers consider only k-party communication, but as we will see in section 5 they can easily be modified for the case of $\ell \leq k$ parties.

5 The Quantum Communication Complexity of GIP

In this section, we further study the power of our quantum communication model by looking at the function of Generalized Inner Product (GIP). We look at general multiparty protocols, where the player's answers can depend on each other. It should be clear how one can define the quantum model for general multiparty computation, where now each subcircuit-player P_j takes as input I_j and also all previous answers $A_1, \ldots, A_{j-1}$ and performs a controlled unitary operation. We refrain from giving a formal definition for the general model, since for the circuit lower bounds we need only look at the simultaneous version and moreover, for our separation we only use a very simple non-simultaneous protocol.

The Generalized Inner Product Function $GIP(X_1, \ldots, X_k)$

Let $X_i \in \{0,1\}^n$. We can think of the k inputs as the rows of a $k \times n$ matrix. Then $GIP(X_1, \ldots, X_k)$ is equal to the number (mod 2) of the columns of the matrix that have all elements equal to 1. More formally, denote with X_i^j the (i,j) element of this matrix (which is equal to the j-th bit of X_i), then

$$GIP(X_1, \ldots, X_k) = \sum_{j=1}^{n} \prod_{i=1}^{k} X_i^j (\text{mod } 2)$$

The function GIP has been studied extensively in the multiparty communication model. Babai *et al.* [3] showed a $\Omega(\frac{n}{2^{2^k}})$ lower bound in the general multiparty model where the answers of the players may depend on previous answers. Chung [8] claimed to improve it to $\Omega(\frac{n}{2^k})$, however the proof is flawed.

It is easy to see that the ℓ-party randomized communication complexity of $GIP(X_1, \ldots, X_k)$ is at least the ℓ-party randomized communication complexity of $GIP(X_1, \ldots, X_\ell)$. If there exists an ℓ-party communication protocol P for the function $GIP(X_1, \ldots, X_k)$, then we can construct an ℓ-party protocol for $GIP(X_1, \ldots, X_\ell)$ by fixing $X_{\ell+1}, \ldots, X_k$ to be the $\mathbf{1}$ vectors.

On the other hand, Grolmusz [9] described a k-party communication protocol for $GIP(X_1, \ldots, X_k)$ with communication $(2k-1)\lceil \frac{n}{2^{k-1}-1} \rceil$. This is a slightly non-simultaneous protocol, since first, player 1 outputs a message and then, depending on that message, the other players output their answers simultaneously.

Using our simulation from Theorem 1, we can show that there exists a quantum $\lceil \frac{k-1}{2} \rceil + 1$-party communication protocol for GIP with the same communication and the same correctness probability (assume without loss of generality that k is odd.) For $k = \log(n+1) + 1$ the quantum communication is only $O(\log n)$. The best known classical protocol for $k = \log(n+1)+1$ has communication $O(\sqrt{n})$. Showing that this bound is optimal, or in other words improving the lower bound for GIP to $\Omega(\frac{n}{2^k})$ would establish an exponential separation between randomized and quantum multiparty communication complexity.

Theorem 3. *Let* $k = \log(n+1)+1$, $\ell = \lceil \frac{k-1}{2} \rceil + 1$ *and a constant* δ. *Then, the* ℓ*-party quantum communication complexity of* $GIP(X_1, \ldots, X_k)$ *is* $QC_\delta^\ell(GIP) = O(\log n)$.

Proof. Grolmusz [9] showed a k-party protocol for $GIP(X_1, \ldots, X_k)$ with communication $(2k-1)\lceil \frac{n}{2^{k-1}-1} \rceil$. Taking $k = \log(n+1)+1$ the communication cost is $(2k-1)$ bits. In fact, the first player communicates a $(k-1)$-bit string and a single bit and the other $(k-1)$ players simultaneously communicate a single bit each. The final answer is the parity of the single bits. The single bits of the $(k-1)$ players depend on the message of the first player and hence this is not a simultaneous messages protocol. We are going to simulate exactly the protocol of Grolmusz by using only $\lceil \frac{k-1}{2} \rceil + 1$ quantum players.

Quantum Protocol

Let $I_1, \ldots, I_k$ be the inputs to the k players in Grolmusz's protocol and $A_1, \ldots, A_k$ the messages they output. As we said, $A_1 \in \{0,1\}^{k-1} \times \{0,1\}$ and for $i = 2, \ldots, k$ A_i is a bit that depends on (I_i, A_1). The idea is to use the first quantum player to simulate exactly the first classical player and for the other players use our simulation technique from section 3. Our protocol is non-simultaneous since the answers of the quantum players $2, \ldots, k$ depend on the classical answer of player 1. More specifically,

- In the first round, we create the following states:

$$|\phi_1\rangle = |1, I_1\rangle, \quad |\phi_i\rangle = |i\rangle|i, I_i\rangle + |i+1\rangle|i+1, I_{i+1}\rangle, \quad i = 2, 4, \ldots, k-1$$

- In the second round, first, quantum player 1 outputs the classical string A_1. The other players read the classical string A_1 and proceed to perform the mapping

$$T : |j, I_j\rangle|0\rangle \mapsto |j, I_j\rangle(-1)^{A_j}|0\rangle$$

- In the third round, we have the classical string A_1 and the states

$$|\chi_i\rangle = |i\rangle|i, I_i\rangle(-1)^{A_i}|0\rangle + |i+1\rangle|i+1, I_{i+1}\rangle(-1)^{A_{i+1}}|0\rangle, \; i = 2, \ldots, k-1.$$

  The protocol quantumly "erases" the inputs resulting in the states

$$|\psi_i\rangle = (-1)^{A_i}|i\rangle + (-1)^{A_{i+1}}|i+1\rangle.$$

  By measuring in the basis $\{|i\rangle \pm |i+1\rangle\}$ it is possible to compute $A_i \oplus A_{i+1}$ exactly and hence compute the parity of all the bits like in the classical protocol.

The correctness of the protocol is $\frac{1}{2} + \delta$, same as in the classical case.

References

1. S Aaronson, Quantum Computing, Postselection, and Probabilistic Polynomial-Time In *Proceedings of the Royal Society A, 461(2063):3473-3482, 2005.*
2. D Aharonov, O Regev, Lattice problems in NP ∩ coNP In *Proc. 45th IEEE FOCS, 2004.*

3. L Babai, N Nisan, M Szegedy, Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs *Journal of Computer and System Sciences, Volume 45 , Issue 2:204 - 232, 1992.*
4. R Beigel, J Tarui, On ACC, *Computational Complexity, 1994.*
5. Z Bar-Yossef, TS Jayram, I Kerenidis, Exponential Separation of Quantum and Classical One-Way Communication Complexity *Proceedings of 36th ACM STOC, 2004.*
6. H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting, *Physical Review Letters, 87(16), 2001.*
7. AK Chandra, ML Furst, RJ Lipton, Multi-party protocols In *Proceedings of the 15th annual ACM STOC,1983.*
8. F Chung Quasi-random classes of hypergraphs *Random Structures and Algorithms, 1990.*
9. V Grolmusz The BNS Lower Bound for Multi-Party Protocols is Nearly Optimal *Information and Computation, 1994.*
10. J Hastad, M Goldmann, On the power of small-depth threshold circuits *Computational Complexity 1:113-129, 1991.*
11. I. Kerenidis, R. de Wolf, Exponential Lower Bound for 2-Query Locally Decodable Codes via a Quantum Argument In *Proceedings of the 15th annual ACM STOC,2003.*
12. E. Kushilevitz, N. Nisan, Communication complexity *Cambridge University Press, 1997.*
13. M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, 2000.
14. R.Raz, The BNS-Chung Criterion for multi-party communication complexity *Journal of Computational Complexity 9(2) (2000), pp. 113-122.*
15. R Raz Exponential separation of quantum and classical communication complexity In *Proceedings of 31st ACM STOC, 1999.*
16. S. Wehner, R. de Wolf, Improved Lower Bounds for Locally Decodable Codes and Private Information Retrieval *32nd ICALP 2005), LNCS 3580, 1424-1436.*
17. R. de Wolf Lower Bounds on Matrix Rigidity via a Quantum Argument. In *33rd International Colloquium on Automata, Languages and Programming (ICALP'06), LNCS 4051, pp.62-71.*
18. AC Yao, On ACC and threshold circuits *Proc. 31st Ann. IEEE FOCS, 1990.*

Efficient Computation of Algebraic Immunity of Symmetric Boolean Functions*

Feng Liu and Keqin Feng

Department of Mathematical Sciences
Tsinghua University, Beijing 100084, China
fengliu03@gmail.com, kfeng@math.tsinghua.edu.cn

Abstract. The computation on algebraic immunity (AI) of symmetric boolean functions includes: determining the AI of a given symmetric function and searching all symmetric functions with $AI = d$ or $AI \geq d$, where $d \leq \lceil \frac{n}{2} \rceil$. In this paper we firstly showed a necessary and sufficient condition of AI of symmetric boolean functions and then proposed several efficient algorithms on computation of algebraic immunity of symmetric boolean functions. By these algorithms we could assess the vulnerability of symmetric boolean functions against algebraic/fast algebraic attacks efficiently, and find all symmetric functions having a given algebraic immunity $AI_n(f) = d$, for some $0 \leq d \leq n$.

Keywords: symmetric boolean function, algebraic immunity, cryptography.

1 Introduction

In the past few years several successful algebraic attacks on stream ciphers were proposed ([1,2,6,7,10]). As a response to this situation, Meier et al.[10] and Breaken [3] introduced the concept of algebraic immunity for a boolean function. Then how to determine the algebraic immunity d of boolean functions (or even symmetric boolean functions) has been paid particular attention ([2,4]).

Let B_n be the ring of n-variable boolean functions $f = f(x_1, \ldots, x_n) : \mathbb{F}_2^n \to \mathbb{F}_2$ and SB_n be the ring of n-variable symmetric boolean functions. For $f \in B_n$, the algebraic immunity of f, denoted by $AI_n(f)$, is defined by

$$AI_n(f) = \min\{\deg g \,|0 \neq g \in B_n, fg = 0 \text{ or } (f+1)g = 0\}$$

where $\deg g$ means the degree of g when g is uniquely expressed as polynomial

$$g(x_1, \ldots, x_n) = \sum_{a=(a_1,\ldots,a_n)\in\mathbb{F}_2^n} c(a) x_1^{a_1} \ldots x_n^{a_n} \quad (c(a) \in \mathbb{F}_2)\,.$$

For $g = 0$ we assume $\deg(0) = -\infty$. From $f(f+1) = f^2 + f = f + f = 0$ we know that $AI_n(f) \leq \deg f$. It is proved that $AI_n(f) \leq \lceil \frac{n}{2} \rceil$ for all $f \in B_n$ (see [9]).

* Supported by the National Fundamental Science Research Program 973 of China with No. 2004 CB3180004.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 318–329, 2007.
© Springer-Verlag Berlin Heidelberg 2007

A symmetric boolean function $f \in SB_n$ can be either uniquely expressed as

$$f = \sum_{i=0}^{n} \lambda_f(i)\sigma_i^{(n)} \quad (\lambda_f(i) \in \mathbb{F}_2)\,.$$

where $\sigma_i^{(n)}$ is the elementary symmetric polynomial of $x_1, \ldots, x_n$

$$\sigma_i^{(n)} = \sigma_i^{(n)}(x_1, \ldots, x_n) = \sum_{1 \le a_1 < \ldots < a_i \le n} x_{a_1} \ldots x_{a_i}$$

or characterized by

$$v_f = (v_f(0), \ldots, v_f(n)) \in \mathbb{F}_2^{n+1}$$

where $v_f(i)$ is $f(a)$ for vectors $a \in \mathbb{F}_2^n$ having Hamming weight $wt(a) = i$.

The relation between $\lambda_f = (\lambda_f(0), \ldots, \lambda_f(n)) \in \mathbb{F}_2^{n+1}$ and $v_f = (v_f(0), \ldots, v_f(n)) \in \mathbb{F}_2^{n+1}$ can be derived by the Lucas theorem. For integers $a, b \ge 0$ and their 2-adic expressions

$$a = a_0 + a_1 2 + \ldots + a_l 2^l\,,\ \ b = b_0 + b_1 2 + \ldots + b_l 2^l \quad (a_i, b_i \in \{0,1\} = \mathbb{F}_2)$$

we define $a \preceq b$ by $a_i \le b_i$ $(0 \le i \le l)$. Then the Lucas theorem says that

$$\binom{a}{b} \bmod 2 = 1 \in \mathbb{F}_2 \Leftrightarrow b \preceq a \tag{1}$$

Therefore for $f \in B_n$ and $0 \le i \le n$, we take a vector $v \in \mathbb{F}_2^n$ with $w_H(v) = i$, then

$$v_f(i) = f(v) = \sum_{j=0}^{n} \lambda_f(j)\sigma_j(v) = \sum_{j=0}^{n} \lambda_f(j)\binom{i}{j} = \sum_{j=0, j \preceq i}^{n} \lambda_f(j) \tag{2}$$

and then by inverse transformation,

$$\lambda_f(i) = \sum_{j=0, j \preceq i}^{n} v_f(j) \tag{3}$$

Moreover, if $f \in SB_n$ and $f \neq 0$, then $\deg f$ is the largest number i such that $\lambda_f(i) \neq 0$.

The computation on algebraic immunity of symmetric Boolean functions includes: determining $AI_n(f)$ of a given symmetric $f \in B_n$ and searching all symmetric $f \in B_n$ with $AI = d$ or $AI \ge d$, where $d \le \lceil \frac{n}{2} \rceil$. In order to determine $AI_n(f)$ of a given symmetric $f \in B_n$, a general idea is to assume a $g = \sum_{0 \le wt(a) \le \lceil \frac{n}{2} \rceil} c(a)x^a \in B_n$ such that $gf = 0$, and to solve the linear systems

$$\{g(\alpha) = 0 \,|\, f(\alpha) = b, \alpha \in \mathbb{F}_2^n\}$$

of variables $\{c(a) \,|\, 0 \le wt(a) \le \lceil \frac{n}{2} \rceil\}$ for each $b \in \mathbb{F}_2$. Then one will obtain some nonzero solutions (annihilators of f) of these systems and determine the $AI_n(f)$,

which is the minimal degree of these nonzero solutions. Until now the best algorithm known for computing the algebraic immunity d of an n-variable boolean function, works roughly $O(D^2)$, where $D \approx \binom{n}{d}$, which is derived from the idea by F. Armknecht in [2]. He also presented an efficient generic algorithm in Section 5 of [2] to assess the vulnerability of arbitrary boolean functions with respect to fast algebraic attacks, which is particularly efficient for symmetric Boolean functions to compute their algebraic immunity in $O(n^3)$. However, F. Armknecht also pointed out that his method is based on some sufficient conditions which implies that they can not deal with all symmetric functions. Another idea to compute $AI_n(f)$ of a given $f \in SB_n$ was introduced by An Braeken in [4], using a special set of polynomials $\sigma_k^{x_k} P_l^{x_l}$ with $k + l = \left\lceil \frac{n}{2} \right\rceil - 1$, $x_k + x_l = n$ and $0 \le x_l \le 2l$ (see [4] for $P_l^{x_l}$). But we can only obtain an upper bound on $AI_n(f)$ by her method. In this paper, we will present some algorithms based on a general observation on symmetric boolean functions which introduces a necessary and sufficient condition of $f \in SB_n$ with $AI_n(f) \ge d$. One of these algorithms can be applied to the question of computing the algebraic immunity of an n-variable symmetric boolean function in $O(n^4)$ time steps (see Algorithm 2 in Section 3). And there is also one efficient algorithm we will present, which can be used to determine $M_{n,d}$ for an integer pair (n, d) $(0 \le d \le \left\lceil \frac{n}{2} \right\rceil)$ (see Algorithm 4 in Section 4). Note that $M_{n,d}$ is denoted to be the set of all n-variable symmetric boolean functions having algebraic immunity at least d.

This paper is organized as follows. Firstly we give a general observation on $f \in SB_n$ with $AI_n(f) \ge d$ for general case n and $1 \le d \le \left\lceil \frac{n}{2} \right\rceil$ in Section 2. We introduce a conception on symmetric support of $f \in SB_n$ and show a result (Lemma 2.3) which plays an important role in this paper. Then in Section 3 and 4 we will present our algorithms for computing algebraic immunity of a given symmetric boolean function and for searching all symmetric boolean functions with a given algebraic immunity d. Finally, we will demonstrate the advantages of our algorithms in implementation and efficiency, by experiments in Section 5.

2 General Observation for $f \in SB_n$ with $AI_n(f) \ge d$

Let n be an arbitrary integer such that $n \ge 2$, $\Omega_0 = \{0, 1, \ldots, n\}$ and $\Omega_1 = \{1, \ldots, n\}$.

Definition 1. *For $f \in B_n$, the weight support of f is defined by*

$$WS(f) = \left\{ i \in \Omega_0 \,\middle|\, \begin{array}{l} \texttt{there exists } a \in \mathbb{F}_2^n \texttt{ with} \\ wt(a) = i \texttt{ such that } f(a) = 1 \end{array} \right\}$$

where $wt(a) = \#\{l \in \Omega_1 \,|a_l = 1\}$ is the Hamming weight of $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$. It is easy to see that for $f \in SB_n$, $WS(f) = \{i \in \Omega_0 \,|v_f(i) = 1\}$. And we also use the notation v_f to denote the vector $(v_f(0), \ldots, v_f(n)) \in \mathbb{F}_2^n$, where

$$v_f(i) = \begin{cases} 1 & i \in WS(f) \\ 0 & i \notin WS(f) \end{cases} \qquad \texttt{for each } i \in \Omega_0.$$

We define a partial order on B_n by, for $f, g \in B_n$, $f \preceq g$ if and only if $WS(f) \subseteq WS(g)$. Then it is easy to see that for $f \in SB_n$ and $g \in B_n$,

$$\begin{aligned} fg = 0 &\Leftrightarrow \texttt{For } a \in \mathbb{F}_2^n, wt(a) = i, g(a) = 1 \texttt{ implies that } v_f(i) = 0 \\ &\Leftrightarrow WS(g) \subseteq \overline{WS(f)} \end{aligned}$$

where $\overline{WS(f)} = \Omega_0 \backslash WS(f)$ is the complementary set of $WS(f)$ in Ω_0, and

$$(f+1)g = 0 \Leftrightarrow WS(g) \subseteq WS(f).$$

Therefore we have the following result.

Lemma 1. *Suppose that $n \geq 2$, $1 \leq d \leq n$. For a function $f \in SB_n$, $AI_n(f) \geq d$ if and only if for each $g \in B_n$ such that $0 \leq \deg g \leq d-1$, we have $WS(g) \nsubseteq WS(f)$ and $WS(g) \nsubseteq \overline{WS(f)}$.*

Next result is important in this paper. For each $l \geq 1$, let

$$p_l = p_l(x_1, \ldots, x_{2l}) = (x_1 + x_2)(x_3 + x_4) \ldots (x_{2l-1} + x_{2l}) \in B_{2l} \tag{4}$$

Then $WS(p_l) = \{l\}$.

Lemma 2. *Suppose that $n \geq 2$ and $f \in SB_n$. If there exists $0 \neq g \in B_n$ such that $fg = 0$, then there exists l, $0 \leq l \leq \left[\frac{n}{2}\right]$ and $0 \neq h = h(x_{x_{2l+1}, \ldots, x_n}) \in SB_{n-2l}$, $\deg h \leq \deg g - l$ such that $fhp_l = 0$.*

Proof. Let Σ_n be the group of permutation on Ω_1. For each permutation $\sigma \in \Sigma_n$, $g \in B_n$, we define $g^\sigma \in B_n$ by $g^\sigma(x_1, \ldots, x_n) = g(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$. For $1 \leq i < j \leq n$, we denote $X_{(i,j)} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)$. Then g can be expressed as

$$g = g(x_1, \ldots, x_n) = g_0(X_{(i,j)}) + x_i g_1(X_{(i,j)}) + x_j g_2(X_{(i,j)}) + x_i x_j g_3(X_{(i,j)})$$

If $g_1(X_{(i,j)}) = g_2(X_{(i,j)})$ for each integer pair (i,j) $(1 \leq i < j \leq n)$, then $g^\sigma = g$ for each permutation $\sigma = (ij)$ $(1 \leq i < j \leq n)$, and then $g^\sigma = g$ for each permutation $\sigma \in \Sigma_n$ since the group Σ_n is generated by the set of permutations (ij) $(1 \leq i < j \leq n)$. Since symmetric boolean functions can be defined as follows:

$$f \in SB_n \Leftrightarrow f^\sigma = f \texttt{ for all } \sigma \in \Sigma_n,$$

we know that g is symmetric and the proof is finished. Otherwise there exists at least one permutation $\sigma = (ij)$ $(1 \leq i < j \leq n)$ such that $g_1(X_{(i,j)}) \neq g_2(X_{(i,j)})$. Then we choose a permutation $\tau \in \Sigma_n$ such that $\tau(i) = 1$ and $\tau(j) = 2$, we have

$$g^\tau = g_0(x_3, \ldots, x_n) + x_1 g_1(x_3, \ldots, x_n) + x_2 g_2(x_3, \ldots, x_n) + x_1 x_2 g_3(x_3, \ldots, x_n)$$

and $g_1(x_3, \ldots, x_n) \neq g_2(x_3, \ldots, x_n)$ so that for $\sigma = (12)$,

$$g^\tau + g^{\tau\sigma} = (x_1 + x_2)(g_1(x_3, \ldots, x_n) + g_2(x_3, \ldots, x_n)) \neq 0$$

$$\deg(g_1 + g_2) \le \max\{\deg g_1, \deg g_2\} \le \deg g - 1$$

and

$$\begin{aligned} f(g^\tau + g^{\tau\sigma}) &= f^\tau g^\tau + f^{\tau\sigma} g^{\tau\sigma} \qquad \texttt{(since } f \texttt{ is symmetric)} \\ &= (fg)^\tau + (fg)^{\tau\sigma} = 0 \end{aligned}$$

If $g' = g_1(x_3, \ldots, x_n) + g_2(x_3, \ldots, x_n) \in B_{n-2}$ is symmetric, the proof is finished. Otherwise we repeat this procedure to get final conclusion.

Let

$$S_{n,d} = \left\{ f_{n,b} = h_{n-2b} p_b \,\middle|\, \begin{array}{l} h_{n-2b}(x_{2b+1}, \ldots, x_n) \in SB_{n-2b} \\ 0 \le \deg h_{n-2b} < d - b, 0 \le b \le d - 1 \end{array} \right\}$$

The following result can be obtained from Lemma 1 and 2 directly.

Lemma 3. *Suppose that* $1 \le d \le \lceil \frac{n}{2} \rceil$ *and* $f_n \in SB_n$. *Then* $AI_n(f) \ge d$ *if and only if*

$$WS(g) \nsubseteq WS(f) \ \texttt{and}\ WS(g) \nsubseteq \overline{WS(f)}$$

for all $g \in S_{n,d}$.

Furthermore, let $S_{\min}(n, d)$ be the set of minimal elements of the partial order set $(S(n, d), \preceq)$. Then lemma 3 is also true if $S(n, d)$ is replaced by $S_{\min}(n, d)$. If the set $S_{\min}(n, d)$ has simple structure, we may get a nice characterization on all $f \in SB_n$ having $AI_n(f) \ge d$.

3 Computing Algebraic Immunity of a Given Symmetric Boolean Function

Now suppose that $0 \ne f \in SB_n$ and let us present our algorithms for computing algebraic immunity of an given symmetric $f \in B_n$. For any two vectors $u = (u_0, \ldots, u_n)$ and $v = (v_0, \ldots, v_n)$ in $\mathbb{F}_2^{n+1}$, we define the &-operation by $(u\&v) = (u_0 v_0, \ldots, u_n v_n)$. Then we get the following equivalences: for any nonzero function $g \in B_n$,

$$\begin{aligned} WS(g) \nsubseteq WS(f) &\Leftrightarrow (v_{f+1}\&v_g) \ne \overline{0} \\ &\Leftrightarrow ((v_f \oplus \overline{1})\&v_g) \ne \overline{0} \\ WS(g) \nsubseteq \overline{WS(f)} &\Leftrightarrow (v_f\&v_g) \ne \overline{0} \end{aligned}$$

where $\overline{0}$ and $\overline{1}$ are the all-zeros and all-ones vectors in $\mathbb{F}_2^{n+1}$ respectively. Then Lemma 3 can be rewritten as follows: let $1 \le d \le \lceil \frac{n}{2} \rceil$ and $f_n \in SB_n$, and then $AI_n(f) \ge d$ if and only if $(v_g\&v_f) \ne \overline{0}$ and $(v_g\&(v_f \oplus \overline{1})) \ne \overline{0}$ for all $g \in S_{n,d}$. From the proof of Lemma 2, we see that for each $0 \ne g \in B_n$ with $fg = 0$ one can find a h such that $fh = 0$, $h \in S_{n,d}$ and $0 \le \deg h \le \deg g$. This is to say that if there is a nonzero annihilator g of f, then one can find another nonzero annihilator h of f with $0 \le \deg h \le \deg g$. So the basic algorithm for determining algebraic immunity of f can be designed as follows:

Algorithm 1. Determining Algebraic Immunity of A Given $f \in SB_n$

1. **Initialization**: v_f (the value vector of f), $S_{n,\lceil \frac{n}{2} \rceil}$
2. **Searching Step**: find the set S of all elements $g \in S_{n,\lceil \frac{n}{2} \rceil}$ which satisfy $(v_f \& v_g) = \overline{0}$ or $((v_f \oplus \overline{1}) \& v_g) = \overline{0}$.
3. **Output**: $AI_n(f)$, where $AI_n(f) = \min\{\deg g \,|g \in S\}$ for non-empty S or $AI_n(f) = \lceil \frac{n}{2} \rceil$ for empty S.

Proposition 1. *The complexity of Algorithm 1 is* $O(2^{\frac{n}{2}})$.

Proof. The number of & operator the main step needs is $2|S_{n,\lceil \frac{n}{2} \rceil}|$. And for any d, $0 \le d \le \lceil \frac{n}{2} \rceil$,

$$|S_{n,d}| = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \sum_{0 \le \deg h \le d-k-1} 1 = \sum_{k=0}^{d-1} (2^{d-k} - 1) = 2^{d+1} - d - 2$$

where h runs over SB_{n-2k}. Then this proposition follows from $O(|S_{n,d}|) = O(2^{\lceil \frac{n}{2} \rceil + 1}) = O(2^{\frac{n}{2}})$.

We should notice that by Algorithm 1 one can get one annihilator g of f with $\deg g = AI_n(f)$. Let $S_{n,\lceil \frac{n}{2} \rceil}(l) = \left\{ hp_l \in S_{n,\lceil \frac{n}{2} \rceil} \,\middle|\, h \in SB_{n-2l}, 0 \le \deg h < \lceil \frac{n}{2} \rceil - l \right\}$, and the function h can be expressed as the nonzero combination of $\sigma_i^{(n-2l)}$ $(0 \le i < \lceil \frac{n}{2} \rceil - l)$. For each $a \in \mathbb{F}_2$ and $0 \le l < \lceil \frac{n}{2} \rceil$, we define the linear systems of $\{c_i \,|0 \le i < \lceil \frac{n}{2} \rceil - l\}$ by

$$Sys(l,a) = \left\{ \sum_{i=0}^{\lceil \frac{n}{2} \rceil - l - 1} c_i \sigma_i^{n-2l}(\alpha) = 0 \,\middle|\, f(\alpha) = a, \alpha \in \mathbb{F}_2^n \right\},$$

whose complexity is about $O((l+1)^3)$. Then the main step of Algorithm 1 can be divided into $2\lceil \frac{n}{2} \rceil$ solving procedures of $Sys(l,a)$ $(0 \le l < \lceil \frac{n}{2} \rceil$ and $a \in \mathbb{F}_2)$. Therefore Algorithm 1 can be revised as follows and its complexity is about $O(n^4)$.

Algorithm 2. Determining Algebraic Immunity of A Given $f \in SB_n$

1. **Initialization**: f, $l \leftarrow 0$
2. **while** $l < \lceil \frac{n}{2} \rceil$ `do`
3. For each $a \in \mathbb{F}_2$, solve $Sys(l,a)$ and let $\deg_l$ be the minimal degree of $g = \sum_{i=0}^{l} c_i \sigma_i^{n-2l}$ where $\{c_i \,|0 \le i \le l\}$ is one nonzero root of $Sys(i,a)$ for some $a \in \mathbb{F}_2$.
4. $l \leftarrow l + 1$
5. **end while**
6. **Output**: $AI_n(f)$, the minimal value of $\{\deg_0, \ldots, \deg_{\lceil \frac{n}{2} \rceil - 1}\}$

Proposition 2. *The complexity of Algorithm 2 is* $O(n^4)$.

Proof. Let l be an integer such that $0 \le l < \lceil \frac{n}{2} \rceil$. The step of determining $\deg_l$ for each l and $a \in \mathbb{F}_2$ can be divided into two steps: Gauss eliminating of homogenous linear system $Sys(l,a)$ and finding the lowest degree of gs associated with nonzero solutions of $Sys(l,a)$. It is well known that the first step (Gauss elimination) has a complexity of $O((l+1)^3)$. Now let us consider the complexity of the second step above. Suppose that $c = (c_0, \ldots, c_l)$ is one nonzero solution of $Sys(l,a)$ and $g = \sum_{i=0}^{l} c_i \sigma_i^{n-2l}$, where σ_i^{n-2l} is the i-th elementary symmetric function. Then $\deg g = \max\{i \,|c_i \neq 0\}$. Therefore, to determine the minimum degree of the gs is to determine the minimum value i_0 such that there exist a nonzero solution $c = (c_0, \ldots, c_{i_0-1}, 1, 0, \ldots, 0)$ of $Sys(l,a)$. After the process of Gauss elimination, one can obtain a system $Sys'(l,a)$ as follows

$$\begin{array}{rrrrrrrrr}
c_0 + a_{1,0}c_1 & + \ldots + & a_{i_0-1,0}c_{i_0-1} & + & a_{i_0,0}c_{i_0} & + & a_{i_0+1,0}c_{i_0+1} & + \ldots = 0 \\
c_1 & + \ldots + & a_{i_0-1,1}c_{i_0-1} & + & a_{i_0,1}c_{i_0} & + & a_{i_0+1,1}c_{i_0+1} & + \ldots = 0 \\
& \ddots & \vdots & & \vdots & & \vdots & \vdots \\
& & c_{i_0-1,i_0-1} & + & a_{i_0,i_0-1}c_{i_0} & + & a_{i_0+1,i_0-1}c_{i_0+1} & + \ldots = 0 \\
& & & & 0 \times c_{i_0} & + & a_{i_0+1,i_0}c_{i_0+1} & + \ldots = 0 \\
& & & & & & \vdots & \vdots
\end{array}$$

Thus we can say that to find i_0 is equivalent to find the first zero position on the diagonal of coefficient matrix of $Sys'(l,a)$, whose complexity is $O(l)$. If the i_0 does not exist, then $Sys(l,a)$ (or $Sys'(l,a)$) has only one root $c = (0, \ldots, 0)$. So the entire complexity of Algorithm 2 is

$$2 \sum_{l=0}^{\lceil \frac{n}{2} \rceil - 1} \big(O((l+1)^3) + O(l)\big) = O\left(2 \sum_{l=0}^{\lceil \frac{n}{2} \rceil - 1} (l+1)^3\right) = O(n^4).$$

Remark 1. Not only the AI of f, but also an annihilator of f or $1+f$ with the minimal degree $AI_n(f)$ we can obtain by Algorithm 1 and 2. However, it is still a hard work to find all annihilators of f or $1+f$ with degree $AI_n(f)$. Then it will be an important direction of our future work to find an efficient method to do so.

By the following table, we will make a comparison between our algorithms and some known ones, which are mentioned in Section 1.

n		10	12	14	16	18	20
$(\sum_{i=0}^{\lceil \frac{n}{2} \rceil - 1} \binom{n}{i})^3$	(1)	386^3	1586^3	6476^3	26333^3	106762^3	431910^3
D^2	(2)	$\binom{10}{5}^2$	$\binom{12}{6}^2$	$\binom{14}{7}^2$	$\binom{16}{8}^2$	$\binom{18}{9}^2$	$\binom{20}{10}^2$
$\lvert AN_S \rvert^3$	(3)	83^3	177^3	367^3	749^3	1515^3	3049^3
n^3	(4)	10^3	12^3	14^3	16^3	18^3	20^3
$2\sum_{k=0}^{\lceil \frac{n}{2} \rceil - 1}(k+1)^3$	(5)	450	882	1568	2592	4050	6050

- The algorithm in case (2), where $D \approx \binom{n}{\lceil \frac{n}{2} \rceil}$, was presented by F. Armknecht in [2] to compute the algorithm immunity of a general n-variable boolean function.
- The algorithm in case (3) was introduced by A. Braeken in [4]. The set AN_S in (3) is a set of n-variable boolean functions with special algebraic normal forms such as $\sigma_k^{x_k} P_l^{x_l}$ with $k + l = \lceil \frac{n}{2} \rceil - 1$, $x_k + x_l = n$ and $0 \leq x_l \leq 2l$.
- The algorithm in case (4) is another algorithm presented by F.Armknecht in Section 5 of his paper [2], whose complexity is $O(n^3)$, and which can not deal with all symmetric functions although its complexity is slight lower than ours.
- Algorithms in case (1), (2) and (3) have much larger complexities than our $O(n^4)$ (the complexity of Algorithm 2). And the algorithm in case (4) is slight lower than $O(n^4)$.

4 Searching All Symmetric Boolean Functions with Algebraic Immunity $\geq d$

Let $M_{n,d}$ be the set of n-variable symmetric boolean functions f with $AI_n(f) \geq d$. Now let us present our algorithms of obtaining $M_{n,d}$ for any n and d ($0 \leq d \leq \lceil \frac{n}{2} \rceil$). Most known algorithms are based on the idea: Let $f \in SB_n$ be a possible function and suppose that$g = \sum_{i=0}^{d-1} \sum_{wt(a)=i} c_a x^a$ is an annihilator of f. Then we can determine the algebraic immunity of f as follows.

- Establish linear systems

$$\left\{ g(\alpha) = \sum_{i=0}^{d-1} \sum_{wt(a)=i} c_a \alpha^a = 0 \,\middle|\, \alpha \in \mathbb{F}_2^n, f(\alpha) = c \right\} \texttt{ for each } c \in \mathbb{F}_2,$$

  since $fg = 0$ or $(f+1)g = 0$.
- Solve these linear systems and let $g = \sum_{i=0}^{d-1} \sum_{wt(a)=i} c_a x^a$ be the nonzero solutions.
- Determine $AI_n(f)$, i.e. the minimal algebraic degree of these gs

If $AI_n(f) \geq d$, then f is a function in $M_{n,d}$. If f runs over all possible functions, then we will obtain the set $M_{n,d}$. Employing this idea, we obtain an efficient algorithm to do this work, since our algorithm 2 is more efficient than those previous ones. However, we can do better than before. Let $S_{\min}(n,d)$ the set of minimal elements of the partial order set $(S(n,d), \preceq)$. Then lemma 2.4 is also true if $S(n,d)$ is replaced by $S_{\min}(n,d)$. Since $|S_{\min}(n,d)|$ is much less than $|S_{n,d}|$ for most cases, then the algorithm can be improved more efficient while $S_{\min}(n,d)$ is known.

Now let us present this improved algorithm. For a given integer d ($0 \leq d \leq \lceil \frac{n}{2} \rceil$) suppose that $S_{\min}(n,d)$ is known.

Algorithm 3. Searching all $f \in SB_n$ with $AI_n(f) \geq d$.

1. **Initialization**: $v_f \leftarrow 0 \in \mathbb{Z}_{2^{n+1}}$, $V \leftarrow \{v_g : g \in S_{\min}(n,d)\}$, $M_n = \emptyset$
2. **while** $v_f < 2^{n+1}$ `do`
3. If $(v\&v_f) \neq \overline{0}$ and $(v\&(v_f \oplus \overline{1})) \neq \overline{0}$ for any $v \in V$, then $M_{n,d} \leftarrow M_{n,d} \cup \{f\}$.
4. $v_f \leftarrow v_f + 1$
5. **end while**
6. **Output**: $M_{n,d}$.

Proposition 3. *The complexity of Algorithm 3 is*

$$O(2^{n+2}|S_{n,d}|) = O(2^{n+2}(2^{d+1} - d - 2)) = O(2^{n+d}).$$

Proof. It is clear that the main step of Algorithm 3 consists of at most $2^{n+2}|S_{n,d}| = 2^{n+2}(2^{d+1} - d - 2)$ &-operations. Then the conclusion follows.

Remark 2. In Algorithm 3 the minimal property of $S_{\min}(n,d)$ is not necessary. Someone having some special reasons can replace $S_{\min}(n,d)$ by another subset S of $S_{n,d}$, which satisfies that: for any $f \in SB_n$ and for any $g \in S_{n,d}$, if $fg \neq 0$, then there exists a $h \in S$ such that $fh = 0$. For example, the special reasons may be that $|S_{\min}(n,d)| \approx |S_{n,d}|$, or $S_{\min}(n,d)$ is very complex to implement, or he has a very powerful computer.

Although one needs at most $2^{n+2}|S_{n,d}| = 2^{n+2}(2^{d+1} - d - 2)$ &-operations to implement Algorithm 3 in standard C, the complexity is still very large. Now let us discuss how to use the iterative method to improve this algorithm. Suppose $f \in SB_n$, $v_f = (v_0, \ldots, v_n)$. Let $f' \in SB_{n-2}$ and $v_{f'} = (v_1, \ldots, v_{n-1})$. Then for a given $d \leq \lceil \frac{n}{2} \rceil$,

$$\begin{aligned}
& AI_n(f) \geq d \\
\Leftrightarrow\ & fg \neq 0 \ \texttt{and}\ (1+f)g \neq 0 \ \texttt{for all}\ g \in S_{n,d} \\
\Leftrightarrow\ & \texttt{(1)}\ fh \neq 0 \ \texttt{and}\ (1+f)h \neq 0 \ \texttt{for all}\ h \in (x_1 + x_2)S_{n-2,d-1},\ \texttt{and} \\
& \texttt{(2)}\ fg \neq 0, (1+f)g \neq 0 \ \texttt{for all}\ g \in SB_n \cap S_{n,d} \\
\Leftrightarrow\ & \texttt{(1)}\ AI_{n-2}(f') \geq d-1,\ \texttt{and} \\
& \texttt{(2)}\ fg \neq 0 \ \texttt{and}\ (1+f)g \neq 0 \ \texttt{for all}\ g \in SB_n \cap S_{n,d} \\
\Leftrightarrow\ & \texttt{(1)}\ AI_{n-2}(f') \geq d-1,\ \texttt{and} \\
& \texttt{(2)}\ (v_f\&v_g) \neq \overline{0} \ \texttt{and}\ ((v_f \oplus \overline{1})\&v_g) \neq \overline{0} \ \texttt{for all}\ g \in SB_n \cap S_{n,d}
\end{aligned}$$

Thus, we can rewrite Algorithm 3 as follows:

Algorithm 4. Searching all $f \in SB_n$ with $AI_n(f) \geq d$ for $d \leq \lceil \frac{n}{2} \rceil$.

1. **Initialization**: $k \leftarrow 0$, $M_{n-2d,0} \leftarrow \mathbb{F}_2^{n-2d}$, $T_0 \leftarrow SB_{n-2d+2} \cap S_{n-2d+2,1}$
2. **while** $k < d$ do
3. (a) **while** v_k runs over $M_{n-2(d-k),k}$ do
   (b) For every $v_k \in M_{n-2(d-k),k}$, and every $a_0, a_1 \in \mathbb{F}_2^n$, let $v'_k = (a_0, v_k, a_1)$ and check that $(v'_k \& v_g) \neq \overline{0}$ and $((v'_k \oplus \overline{1}) \& v_g) \neq \overline{0}$ for all $g \in T_k$.
   (c) $M_{n-2(d-k-1),k+1} \leftarrow \left\{ v'_k = (a_0, v_k, a_1) \,\middle|\, v_k \in M_{n-2(d-k),k}, a_0, a_1 \in \mathbb{F}_2 \right\}$, whose elements satisfies $(v'_k \& v_g) \neq \overline{0}$ and $((v'_k \oplus \overline{1}) \& v_g) \neq \overline{0}$ for all $g \in T_k$.
   (d) **end while**
4. $k \leftarrow k+1$, $T_k \leftarrow SB_{n-2k+2} \cap S_{n-2k+2,k+1}$
5. **end while**
6. **Output**: $M_{n,d}$.

Remark 3. The complexity of this algorithm will depend on these numbers $\left| M_{n-2(d-k),k} \right|$, $0 \leq k \leq d-1$, and can not be efficiently approximated yet. Moreover, one can improve the step 3-b of this algorithm by the method of solving linear systems (which left to the reader).

Remark 4. **Determining $M_{n,\frac{n}{2}}$ for an even integer n.** For case n even, LongJiang Qu et al. in [12] presented many necessary conditions for symmetric boolean functions $f \in B_n$ having maximum $AI_n(f) = \frac{n}{2}$. And they also conjectured the tightness of their conditions for case $n = 2^m$ $(m \geq 2)$ which can be stated as follows: Suppose that $n = 2^m$ $(m \geq 2)$ and $f \in SB_n$, then $AI_n(f) = \frac{n}{2}$ if and only if the following two conditions satisfied

1) $v_f(i + \frac{n}{2} - 2^t) = v_f(i + \frac{n}{2}) + 1$ for all $1 \leq i \leq 2^t - 1$. $(1 \leq t \leq m-1)$
2) $\big(v_f(0), v_f(\frac{n}{2}), v_f(n)\big) \notin \{(0,0,0), (1,1,1)\}$.

We will give a proof of this conjecture in another subsequent paper of us. But the readers can check its truth with our lemma 3 themselves, which has been checked by us for case $m = 2, 3, 4, 5, 6, 7$. Then one can easily determine the function set $M_{2^m,2^{m-1}}$ and replace the initial set $M_{n-2d,0}$ to $M_{2^m,2^{m-1}}$ in Algorithm 4 for case even n, where $d = \frac{n}{2}$ and $2^m \leq n < 2^{m+1}$.

5 Our Experiments and Conclusions

In this section, we will illustrate the efficiency of our algorithms by the experiments of determining $M_{n,d}$ for given n and d. The usual method in searching $M_{n,d}$ is to check all possible functions and determine their algebraic immunity. For example, we can employ the algorithm in [2] as the method of determining

$AI_n(f)$ for a given $f \in SB_n$. But the complexities of all previous algorithms based on this method are much larger than ours. And the advantages of our algorithms can be summarized as follows:

- The main step of Algorithm 3 and 4 consists of &-operations, which is simpler to implement than those previous ones. And in most cases, $|S_{\min}(n,d)| \ll |S_{n,d}| \ll \left|S_{n,\lceil \frac{n}{2} \rceil}\right|$, so the practical complexity of Algorithm 3 and 4 is much less than our theoretic complexities.
- If $|S_{\min}(n,d)|$ is close to $|S_{n,d}|$, then Algorithm 3 and 4 will lose their advantages. And we can employ Algorithm 2 as the method of determining AI of a given f, whose complexity $O(n^4)$ is still much smaller than all previous algorithms we knew.
- In our algorithms the memory complexity can be neglected.

In her paper [4], A. Braeken combined her special idea with the conventional methods and found the sets $M_{n,d}$ for case of $n = 6, 8, \ldots, 16$ and $d = \frac{n}{2}$ (not for all possible d). These search results may be very close to the utmost limit of those previous algorithms if one works with a standard computer (1.0 ghz CPU, 256M ram). For example, it costs us about 14 (or 72) hours for case of $n = 18$ (or 20) respectively in the experiments of determining $M_{n,\frac{n}{2}}$ based on those previous algorithms in [2] and [4]. And it did not be finished for case of $n = 22$ because of its large complexity. However, by our algorithms we have obtained $M_{n,d}$ for $n = 4, 5, 6, \ldots, 26$ and for each $0 \le d \le \lceil \frac{n}{2} \rceil$, all of which we spent several hours in.

Until now, we have presented several efficient algorithms on computation of algebraic immunity of symmetric boolean functions. In comparison with all previous algorithms, our algorithms are much easier to implement and much faster to run. But two open problems can be raised as follows:

- Most of the previous algorithms can be applied to the case of general boolean functions. But our algorithms can not. Thus there is a question naturally: how to extend our methods in this paper to the cases of general boolean functions?
- By our algorithms, it is possible to find one annihilator g of f ($\deg g = AI_n(f)$), but it is impossible to find all annihilators g of f ($\deg g = AI_n(f)$). The problem is how to design an efficient algorithm to search all such functions from our knowledge of f.

To solve these open problems will be our next working directions.

References

1. F. Armknecht. *Improving Fast Algebraic Attacks.* In FSE 2004, LNCS 3017, pages 65-82. Springer Verlag, 2004.
2. F. Armknecht, et al. *Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks.* EUROCRYPT 2006, LNCS 4004, pages 147-164, 2006.

3. L. M. Breaken. *Algebraic Attack over $GF(q)$*. In Progress in Cryptology - INDOCRYPT2004, pages 84-91, LNCS 3348, Springer-Verlag.
4. An. Braeken and B. Preneel. *On the algebraic immunity of symmetric Boolean functions*. INDOCRYPT 2005, , 2005, pages 35-48, LNCS, Springer-Verlag.
5. C. Carlet, D. K. Dalai, K. C. Gupta and S. Maitra. *Algebraic Immunity for Cryptographically Significant Boolean Functions: Analysis and Construction*. Preprint 2006.
6. N. Courtois. *Fast algebraic attacks on stream ciphers with linear feedback*. In Advances in Cryptology - CRYPTO 2003, LNCS 2729, pages 176-194. Springer Verlag, 2003.
7. N. Courtois and W. Meier. *Algebraic attacks on stream ciphers with linear feedback*. In Advances in Cryptology - EUROCRYPT 2003, LNCS 2656, pages 345-359. Springer Verlag, 2003.
8. D. K. Dalai, K. C. Gupta and S. Maitra. *Results on Algebraic Immunity for Cryptographically Significant Boolean Functions*. In INDOCRYPT 2004, pages 92-106, LNCS 3348.
9. D. K. Dalai, S. Maitra and S. Sarkar. *Basic Theory in Construction of Boolean Functions with Maximum Possible Annihilator Immunity*. Cryptology ePrint Archive, http://eprint.iacr.org/, No. 2005/229, 15 July, 2005. To be published in Designs, Codes and Cryptography.
10. W. Meier, E. Pasalic and C. Carlet. *Algebraic attacks and decomposition of Boolean functions*. In Advances in Cryptology - EUROCRYPT 2004, number 3027 in Lecture Notes in Computer Science, pages 474-491. Springer Verlag, 2004.
11. Longjiang Qu, Chao Li and Keqin Feng. *A note on symmetric Boolean Functions with maximum algebraic immunity in odd number of variables*. Preprinted.
12. Longjiang Qu, Guozhu Feng, Chao Li and Keqin Feng. *On 2^m variables symmetric Boolean functions with maximum algebraic immunity*. Preprinted.

Improving the Average Delay of Sorting*

Andreas Jakoby [1], Maciej Liśkiewicz [1], Rüdiger Reischuk [1], and Christian Schindelhauer [2]

[1] Inst. für Theoretische Informatik, Universität zu Lübeck, Germany
{jakoby,liskiewi,reischuk}@tcs.mu-luebeck.de
[2] Dept. of Computer Science, Universität Freiburg,
schindel@informatik.uni-freiburg.de

Abstract. In previous work we have introduced an average case measure for the time complexity of Boolean circuits – that is the delay between feeding the input bits into a circuit and the moment when the results are ready at the output gates – and analysed this complexity measure for prefix computations. Here we consider the problem to sort large integers that are given in binary notation. Contrary to a *word comparator sorting circuit* C where a basic computational element, a comparator, is charged with a single time step to compare two elements, in a *bit comparator circuit* C' a comparison of two binary numbers has to be implemented by a Boolean subcircuit CM called *comparator module* that is built from Boolean gates of bounded fanin. Thus, compared to C, the depth of C' will be larger by a factor up to the depth of CM.

Our goal is to minimize the average delay of bit comparator sorting circuits. The worst-case delay can be estimated by the depth of the circuit. For this worst-case measure two topologically quite different designs seems to be appropriate for the comparator modules: a tree-like one if the inputs are long numbers, otherwise a linear array working in a pipelined fashion. Inserting these into a word comparator circuit we get bit level sorting circuits for binary numbers of length m for which the depth is either increased by a multiplicative factor of oder $\log m$ or by an additive term of order m.

We show that this obvious solution can be improved significantly by constructing efficient sorting and merging circuits for the bit model that only suffer a constant factor time loss on the average if the inputs are uniformly distributed. This is done by designing suitable hybrid architectures of tree compaction and pipelining. These results can also be extended to classes of nonuniform distributions if we put a bound on the complexity of the distributions themselves.

1 Introduction

For circuits, depth is normally used to measure the time a computation takes. This is a worst case estimation. In [JRS94] we have defined an average case measure for the time complexity of circuits called **delay**. It has been observed that in many cases critical paths of a given circuit, e.g. paths between input and output gates of maximal length,

* Supported by DFG research grant Re 672/3.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 330–341, 2007.
© Springer-Verlag Berlin Heidelberg 2007

have no influence on the final output. Hence, the output values of the circuit for some inputs can be obtained much earlier.

The average delay of basic functions like OR, ADDITION, PARITY, and THRESHOLD has been estimated precisely. These are special instances of the parallel prefix problem that has been investigated in detail in [J98]. In many cases we have found circuit designs that are exponentially faster on average than the optimal circuits for the worst-case [JRS94, JRSW94, JRS95]. On the other hand, we could show lower bounds saying that for certain functions, e.g. PARITY, the average delay remains asymptotically the same as in the worst case. A similar result holds for the problem to sort n bits that has worst-case complexity $\Theta(\log n)$. For the worst case, the lower bound follows from a simple counting argument, the upper bound has been established by a nontrivial construction of Ajtai, Komlos and Szemeredi [AKS83].

The delay of a sorting circuit may be smaller than its depth as can be seen in Fig. 1 showing a sorting circuit C_3 for 3 elements. The first picture shows the circuit consisting of 3 comparators A, B, C. Its depth is 3, too, since the line in the middle marked with input y goes through each comparator. The pictures show the flow of the inputs through the circuit starting with time $t = 0$ when all inputs are at the left end. However, for the given input vector $(1, 0, 0)$, on the critical path in the middle there does not occur a delay of 3. The reason is as follows: already in the first time step the lower 0 can be passed through comparator B to its upper output line although the second input for B has not arrived yet. No matter, what kind of bit this will be, comparator B can be set to an X that switches the inputs because we can be sure that a 0 must occur at the upper output. In the second phase this allows comparator C to do its job since both its inputs are already there and comparator B to finish its work by passing the input 1 on its upper line to the lower output line. Still, this saving in the computation time has no asymptotic effect as we have shown in [JRS94].

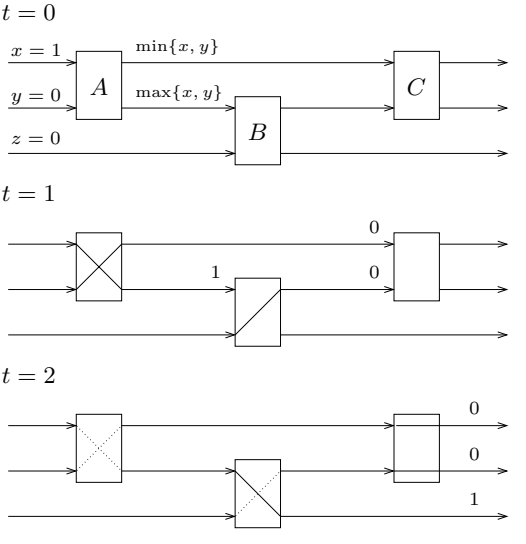


Fig. 1. The flow of inputs in a computation of C_3

Fact 1. *The average delay of a sorting circuit over an arbitrary finite basis with gates of bounded fanin that sorts n uniformly distributed bits is at least $\Omega(\log n)$.*

Thus, sorting n elements requires logarithmic time - even on the average. The complexity of sorting seems to be settled. But what happens if we do not have to sort single bits, but long binary numbers. Obviously, the depth has to increase since a binary circuit of bounded fanin cannot compare two long numbers in constant time.

Let n denote the number of elements that have to be sorted and let us start with a **word comparator circuit C_n**. A **comparator** has indegree and outdegree 2, and takes two elements of the sorting domain and outputs the minimum at the top and the maximum at the bottom output node. Let $\texttt{depth}(C_n)$ denote the depth of the circuit where each comparator is assumed to have depth 1 (see Fig. 1).

If the elements to be compared are binary numbers of some length m we call this the **(n, m)-sorting problem**. Now let us consider the physical realization of comparators. A comparator $\texttt{CM}_m$ that compares two m-bit numbers has to be built on the bit level. Such a subcircuit we will call a **comparator module**. There are two obvious alternatives how to design a comparator module and combine it with the topology of a word comparator circuit.

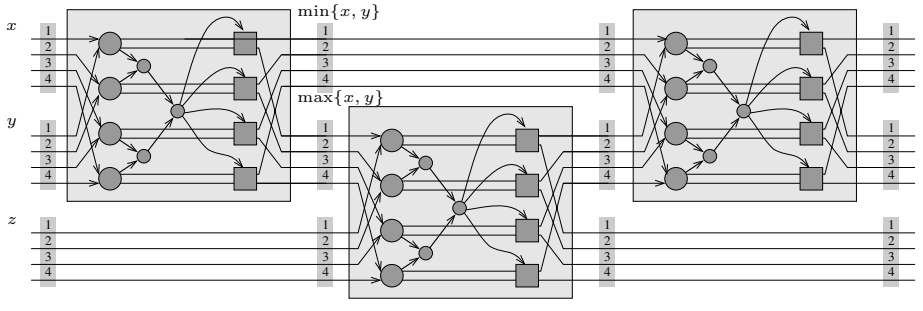


Fig. 2. The sorting circuit $C_{3,4}$ with comparators of a tree architecture

On the one hand, one can compare two numbers x, y bitwise. Every bit comparison generates a result $<, =$ oder $>$ and these results can be combined in binary tree-like fashion to determine the result ρ determining which number is the smaller or whether the numbers are equal. Each pair of bits of x, y is then routed to the appropriate output position by a switch that is driven by ρ. Assume that the combination of two bit comparison results can be performed by a subcircuit of depth δ. Then such a comparator $\texttt{CM}_m$ can be implemented by a binary circuit of depth $\delta \log m + O(1)$. This assumes no bound on the fanout of a Boolean gate (if one insists on fanout at most 2 the depth becomes $(\delta + 1) \log m + O(1)$). Thus, in total we get a bit level sorting circuit $C_{n,m}$ of depth $(\delta \cdot \log m + O(1)) \cdot \texttt{depth}(C_n)$ (see Fig. 2).

Alternatively, one could compare the bits of two numbers in a linear fashion starting with the leading bits. This requires depth linear in the number of bits, but has the advantage that after δ steps the leading bits of the two results of that comparator are already known. This pipelined construction increases the depth of the sorting circuit C_n only by an additive term $\delta \cdot m$ resulting in a bit level sorting circuit $C'_{n,m}$ of depth at most $\delta \cdot (m + \texttt{depth}(C_n) - 1)$ (see Fig. 3).

A detailed discussion of sorting in the bit model can be found in Section 1.1.2 of [L92]. Several papers have considered worst-case delay of Boolean sorting networks explicitly. In [AB93] Al-Hajery and Batcher constructed *bit serial bitonic sorting networks* (BBSN) of size $O(n \log n)$ that sort n numbers each of length $m = O(\log n)$ in $O(\log^2 n)$ steps. BBSN is a *periodic* network of depth $O(\log n)$ and size $O(n \log n)$

based on pipelining. The model of [AB93] differs, however, from the word comparator circuit in that BBSN is a network of bit processors which has the same topology as bitonic sorting network and which processes m-bit input strings in a bit serial fashion.

In [LP90] Leighton using methods due to Thompson [T79] proved an $\Omega(n+m)$ lower time bound for (n,m)-sorting on a $(m \times n)$–array of bit processors. In addition, several papers have discussed VLSI architectures for sorting (e.g. [T83, LO99, HL00, LDH03]).

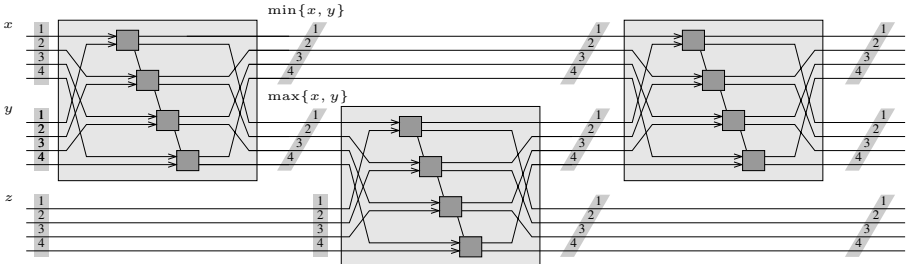


Fig. 3. The sorting circuit $C'_{3,4}$ based on linear arrays. The gray boxes on the links that connect the comparator modules illustrate that output bits of higher order are available before the output bits of lower order.

Using the topology of the asymptotically optimal AKS-network [AKS83] and noticing that $\delta = O(1)$, the pipelined construction by linear arrays described above sorts n numbers of length $m \leq O(\log n)$ in $O(\log n)$ depth. For $m \geq \log n \ \mathrm{loglog}\, n$, tree-like comparator modules seem to be better suited and give depth $O(\log n \ \log m)$. Because of huge constants, AKS-networks are only advantageous for very large n.

In [LP90] Leighton and Plaxton constructed *butterfly*-based sorting networks that sort correctly with probability close to 1. These networks have depth $7.45 \log n$. An implementation of this topology with binary gates yields a randomized sorting circuit of $O(m + \log n)$, resp. $O(\log n \ \log m)$ depth with small constant factors and low error probability.

In this paper we investigate the (n,m)-sorting problem for m significantly larger than $\log n$. New comparator modules that are hybrid versions of the two basic topologies will be constructed that speed up sorting networks on average assuming uniformly distributed inputs.

Theorem 1. *For every m, there are comparator modules $\mathtt{CM}_m$ with the following property: If a Boolean circuit $C_{n,m}$ for the (n,m)-sorting problem is derived from a word comparator circuit C_n by implementing its comparators as $\mathtt{CM}_m$ modules then assuming uniformly distributed inputs $C_{n,m}$ faces a delay of at most $O(\mathtt{depth}(C_n))$ with probability at least $1 - 1/n$. Even in the worst case, the delay does not exceed $O(\mathtt{depth}(C_n) \cdot \log(n+m))$. Thus, the average delay is at most a constant factor larger than the depth of the word comparator circuit independent of the length of the binary numbers.*

The proof will be given in Section 4. This construction requires gates of unbounded fanout to spread information about comparison results fast. If one requires a constant fanout restriction the delay bound becomes $O(\mathtt{depth}(C_n) + \log\, m)$ with probability $1 - 1/n$ and the average delay stays independent of m as long as $m \leq 2^n$. Thus even in case of strictly bounded fanout, for large numbers m we achieve the best combination of the simple architectures described above concerning the average delay: only a logarithmic increase $\log\, m$ instead of m, and this only by an additive term rather than a multiplicative factor.

Small average delay can also be achieved for merging lists of m-bit numbers. In particular, based on the odd-even merge topology we show that there exists a bit comparator circuit that merges two lists of $n/2$ numbers each in $O(\log\, n + \log\, m)$ steps on average. As a consequence we obtain

Theorem 2. *Let $\mathcal{M}_n$ be the odd-even merge sort word comparator circuit for n elements. Then for every m, there are comparator modules $\mathtt{CM}_m$ such that the Boolean circuit derived from $\mathcal{M}_n$ by replacing its comparators by $\mathtt{CM}_m$ modules solves the (n, m)-sorting problem and with probability at least $1 - 1/n$ its delay is bounded by $O(\log^2 n)$.*

Furthermore, the average delay of these circuits is bounded by $O(\log^2 n)$ and their worst-case behaviour is as good as that of worst-case optimal ones. This small average delay bound can also be obtained for families of nonuniform distributions of low complexity (Theorem 4 below). For this result our circuit design does not use any knowledge about the actual distribution μ, it works uniformly for all such μ.

The rest of this paper is organized as follows. Section 2 defines asynchronous Boolean circuits and their timing, in particular the complexity measure delay. The design of efficient comparator modules is described in Section 3. In Section 4 we construct specific bit comparator circuits for sorting and merging and analyse their average delay for the uniform distribution. This is extended to nonuniform distributions in Section 5.

2 Timing of a Boolean Circuit

In the following let $\log\, n := \lceil \log_2 n \rceil$ denote the binary logarithm rounded up.

If we want to exploit possibilities to speedup the computation of a Boolean circuit it has to work in an asynchronous fashion. For this mode one has to extend the binary logic to indicate when a Boolean value is *ready* of *valid.* How this can be done efficiently has been discussed in [JRS94]. To concentrate on the topological aspects of sorting circuits here we simply assume that each gate knows from its predecessors when their values are ready.

Let C be a Boolean circuit, and $V_{\text{in}},\ V_{\text{out}}$ denote its input, resp., output gates. For a gate g and input x of C let $\text{res}_g(x)$ denote the value that is generated by g on x. If g is the i-th input gate then $\text{res}_g(x) = x_i$. Otherwise, $\text{res}_g(x)$ is determined by the values $\text{res}_{g_i}(x)$ of its immediate predecessors g_i and the type of g.

Circuits that work in an asynchronous mode may not get all their input bits at the same time. Similar to [JS01] we therefore make the following definitions.

Definition 1. A **starting-line** for C is a function $\mathcal{S} : V_{\text{in}} \to \mathbb{N}$. Given a starting-line $\mathcal{S}$ for C, we define a function $\mathtt{time}^C_{\mathcal{S}}$ for pairs (g, x) where g is a gate of C and x an input as follows:

$$\mathtt{time}^C_{\mathcal{S}}(g, x) \;:=\; \begin{cases} \mathcal{S}(g) & \text{if } g \text{ is an input gate,} \\ 0 & \text{if } g \text{ is a constant gate,} \\ 1 + t_g(x) & \text{else,} \end{cases}$$

where $t_g(x)$ denotes the smallest time t, such that the values $\text{res}_{g_i}(x)$ of those immediate predecessors g_i of g with $\mathtt{time}^C_{\mathcal{S}}(g_i, x) \le t$ uniquely determine $\text{res}_g(x)$.

Thus, $\mathtt{time}^C_{\mathcal{S}}(g, x)$ denotes the earliest moment when g knows its value assuming that the inputs are available according to the starting time $\mathcal{S}$. For the circuit C itself we define the timing by $\mathtt{time}^C_{\mathcal{S}}(x) \;:=\; \max_{g \in V_{\text{out}}} \mathtt{time}^C_{\mathcal{S}}(g, x)$.

Let $\mathtt{time}^C(x)$ denote the timing if the starting-line $\mathcal{S}$ is identically 0.

Given a probability distribution μ on the input space, we define the **average delay of C** by $\mathbf{Etime}_{\mu}(C) \;:=\; \sum_x \mu(x) \cdot \mathtt{time}^C(x)$. □

Normally, all input bits are available at the beginning of a computation, that is at time 0. In the following we will also consider the case when some bits are delayed. For this purpose, for $k \in [1..m]$ let us define the function $\sigma_k : [1..m] \to \mathbb{N}$ by

$$\sigma_k(j) \;:=\; \begin{cases} j - 1 & \text{if } j \le k, \\ k + 1 & \text{else.} \end{cases}$$

3 Average Case Efficient Comparator Modules

Definition 2. Let $\mathbb{B} = \{0, 1\}$ denote the binary alphabet and $\Sigma_\rho := \{\mathtt{LE}, \mathtt{EQ}, \mathtt{GT}\}$ an alphabet to specify the result of a comparison of two elements, numbers or bits: *less, equal, or greater*. Σ_ρ will suitably be coded over $\mathbb{B}$ – for example by the 3 vectors $(1,0,0)$, $(0,1,0)$, $(0,0,1)$. In the following x, y, u, v will always denote variables that hold a binary value and $\rho, \rho_1, \rho', \ldots$ are variables that take values from Σ_ρ. The Boolean sorting circuits will be constructed from 2 basic types of gates, S-gates and R-gates (see Fig 4).

– ***S*-gate:** it takes 3 inputs ρ, x, y and generates the 3 outputs u, v, ρ'. The input-output relation is defined as follows:

for $\rho = \mathtt{EQ}$ and $x < y$:	$u = \min\{x, y\} = x$,	$v = \max\{x, y\} = y$,	$\rho' = \mathtt{LE}$,
for $\rho = \mathtt{EQ}$ and $x > y$:	$u = \min\{x, y\} = y$,	$v = \max\{x, y\} = x$,	$\rho' = \mathtt{GT}$,
for $\rho = \mathtt{EQ}$ and $x = y$:	$u = \min\{x, y\}$,	$v = \max\{x, y\}$,	$\rho' = \mathtt{EQ}$,
for $\rho = \mathtt{LE}$:	$u = x$,	$v = y$,	$\rho' = \mathtt{LE}$,
for $\rho = \mathtt{GT}$:	$u = y$,	$v = x$,	$\rho' = \mathtt{GT}$.

– ***R*-gate:** the inputs are ρ, ρ_1, ρ_2, the only output is ρ':

for $\rho \neq \mathtt{EQ}$:	$\rho' = \rho$,
for $\rho = \mathtt{EQ}$ and $\rho_1 \neq \mathtt{EQ}$:	$\rho' = \rho_1$,
else	$\rho' = \rho_2$.

□

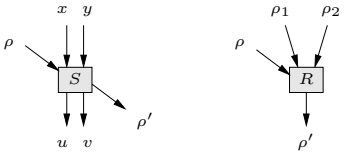


Fig. 4. S-gate and R-gate

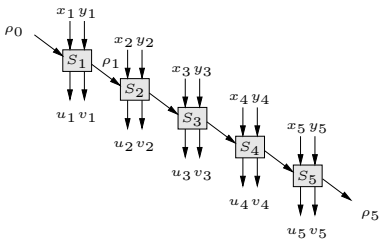


Fig. 5. A line comparator module

According to the Boolean basis and the coding of Σ_ρ both types of gates can be realized by small subcircuits of some fixed depth at most δ. For simplicity, through the rest of the paper we will assume that $\delta = 1$, otherwise one has to add this as a constant factor to all the circuit bounds stated below. For an S-gate it is important to note that depending on its input values, its 3 outputs may be ready at different times. Thus, the timing information should rather be attached to the output wires of a gate than to the gate itself. Since this will not be important in the following we stick to the simpler model.

Our circuit designs will also make use of simplified versions of an S-gate. An L-gate is an S-gate where the ρ'-output is not needed. An U-gate in addition does not need the ρ-input and behaves as if this input were `EQ`. Furthermore, an U-gate does not have the outputs u and v. Also some R-gates will have the input ρ be missing.

A comparator module $\texttt{CM}_m$ is a subcircuit built from S- and R-gates that takes two binary numbers $x = x_1 \dots x_m$ and $y = y_1 \dots y_m$ and produces two output strings u and v such that $u = \min\{x, y\}$ and $v = \max\{x, y\}$. We assume that x_1, resp. y_1 are the leading bits of the binary numbers. In addition, $\texttt{CM}_m$ outputs the result $\rho \in \Sigma_\rho$ of the comparison, that is either `LE`, `EQ` or `GT`. In the following, ρ will be called the **compare info** of $\texttt{CM}_m$. Let us first describe more formally the line- and tree-comparator module introduced above.

Definition 3. A **line comparator module** $\texttt{LCM}_m$ (see Fig. 5) is a comparator module consisting of a linear array of S-gates $S_1, \dots, S_m$ where each S_i gets the i-th bit of x and y and the compare result ρ_{i-1} of S_{i-1}. For S_1 we define $\rho_0 := \texttt{EQ}$. S_i outputs the two bits u_i and v_i and ρ_i as the result of comparing the prefixes $x_1, \dots, x_i$ and $y_1, \dots, y_i$. The compare info of $\texttt{CM}_m$ is ρ_m, i.e. the compare result of the last S_m. □

Even though some of the pairs u_i, v_i may be computed faster (if $x_i = y_i$), since the ρ_i form a linear chain, gate S_i always has to wait for the output ρ_{i-1} of its left neighbour in order to determine its output ρ_i. Thus, we get the following timing for a `LCM`.

Lemma 1. If $\mathcal{S}$ is a starting-time for $\texttt{LCM}_m$ such that $\mathcal{S}(x_i), \mathcal{S}(y_i) \le i - 1$ for all $i \in [1..m]$ then $\texttt{time}_{\mathcal{S}}^{\texttt{LCM}_m}(S_j, (x, y)) = j$ for all $j \in [1..m]$ and all input pairs (x, y).

Definition 4. A **tree comparator module** $\texttt{TCM}_m$ (see Fig. 6) makes all the comparisons of input pairs x_i, y_i in parallel by a sequence of U-gates $U_1, \dots, U_m$, and then combines their results $\rho_1, \dots, \rho_m$ by a binary tree of R-gates to obtain the compare

info ρ. The root of this tree will be denoted by $\hat{R}$. The compare info ρ at $\hat{R}$ is then used to drive m L-gates $L_1, \ldots, L_m$. L_i either leads the two inputs x_i, y_i simply through if ρ equals LE or EQ, or exchanges their order otherwise. Value ρ can either be forwarded to the L_i directly if we allow unbounded fanout or we have to use another binary tree to duplicate this information if the fanout is bounded. In the following we will consider the case of unbounded fanout, otherwise in the timing bounds below one has to add another additive term $\log m$. □

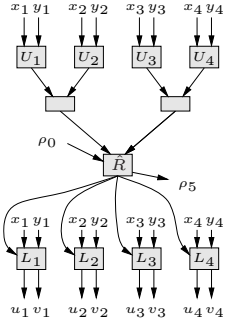


Fig. 6. A tree comparator module

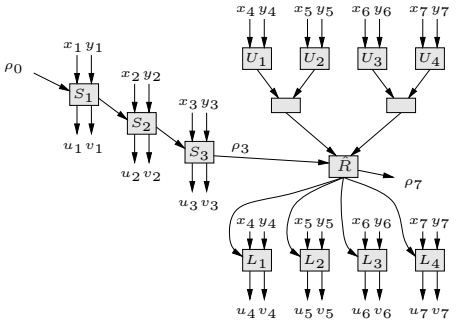


Fig. 7. A line tree comparator module

Lemma 2. For arbitrary $\mathcal{S}$ and all inputs (x, y) it holds for all $j \in [1..m]$

$$\mathtt{time}_{\mathcal{S}}^{\mathtt{TCM}_m}(L_j, (x, y)) \;=\; 2 + \log\, m + \max\{\mathcal{S}(x_i),\, \mathcal{S}(y_i) \mid i \in [1..m]\}\,.$$

To be more efficient in the average case we now define hybrid versions of these two architectures. They will depend on an additional parameter $k \in \mathbb{N}$. When applying to the sorting problem of n elements k typically will be of order $\log n$.

Definition 5. A ***k*-line tree comparator module** (see Fig. 7), $\mathtt{LTCM}_{m,k}$ for short, consists of a line comparator module $\mathtt{LCM}_k$ for the prefixes $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$, and a tree comparator module $\mathtt{TCM}_{m-k}$ for the suffixes $x_{k+1}, \ldots, x_m$ and $y_{k+1}, \ldots, y_m$ with the following modification. The root $\hat{R}$ of the tree comparator additionally gets the compare info ρ_k of $\mathtt{LCM}_k$ and if this result is EQ then it works as previously. Otherwise, $\hat{R}$ outputs this value as a result of the comparison between x and y and propagates this value to the L_i. □

A combination of the two timing bounds for LCM and TCM modules gives

Lemma 3. For all starting-lines $\mathcal{S}$ with $\mathcal{S}(x_i), \mathcal{S}(y_i) \le \sigma_k(i)$ and all input pairs (x, y) it holds $\mathtt{time}_{\mathcal{S}}^{\mathtt{LTCM}_{m,k}}(S_j, (x, y)) \;\le\; j$ for $j \le k$ and

$$\mathtt{time}_{\mathcal{S}}^{\mathtt{LTCM}_{m,k}}(L_j, (x, y)) \;\le\; \begin{cases} k + 2 + \log\,(m-k) & \text{for } \rho_k = \mathtt{EQ}, \\ k + 2 & \text{for } \rho_k \neq \mathtt{EQ}. \end{cases}$$

To achieve small average case delay for nonuniform distributions we need another type of comparator module that can be found in the full paper.

4 Average Case Delay for the Uniform Distribution

The previous section has shown that the delay of a comparator module depends on the length of the prefix up to which its two inputs x and y are identical. Therefore, we make the following definition.

Definition 6. Let $X = X^1, \ldots, X^n$ be a sequence of strings with $X^i = x_1^i \ldots x_m^i \in \{0,1\}^m$, $c \in \mathbb{N}$, and $w \in \{0,1\}^c$. We call w a **conflict prefix** of X if X contains two string X^i, X^j $(i \neq j)$ with prefix w. Let $\mathtt{conf}_c(X)$ denote the number of different conflict prefixes in X of length c.

A **c-congestion** of X is a subsequence of X such that all its members have identical prefixes of length c. Let $\mathtt{con}_c(X)$ denote the maximal size (number of elements of the subsequence) of a c-congestion of X.

Obviously, the values $\mathtt{conf}_c(X)$ are monotonically decreasing with c. If $\mathtt{con}_c(X) = 1$ then the strings in X have pairwise different prefixes of length c.

In this section we assume that $\mathcal{X}^{n,m}$ is a uniformly distributed random variable generating an independent sequence $X^1, \ldots, X^n$ of binary numbers of length m each. We can upperbound conflicts and congestion as follows.

Lemma 4. [Conflict Prefix and Congestion Bound] For every $c,\ \beta,\ \gamma \in \mathbb{N}$ it holds:

$$\Pr[\mathtt{conf}_c(\mathcal{X}^{n,m}) \geq \beta] \ \leq\ 2^{-\beta(c-2\log\, n)}$$

and

$$\Pr[\mathtt{con}_c(\mathcal{X}^{n,m}) \geq \gamma] \ \leq\ 2^{-\gamma(c-\log\, n)+c}\ .$$

We will use k-line tree comparators with different parameters k. Circuits of such comparator modules work efficiently if the k-congestion of the input strings is small. From the lemma above follows that for $c \geq 3\log\, n$ the c-congestion does not exceed 1 with high probability, in particular $\Pr[\mathtt{con}_{3\log\, n}(\mathcal{X}^{n,m}) > 1] \ \leq\ 1/n$.

On the other hand for $c \leq (1-\epsilon)\log\, n$ with $\epsilon > 0$, the c-congestion may typically be quite large. Hence, our circuit designs will choose the parameter k in the interval $[2\log\, n + \varepsilon..3\log\, n]$. For the rest of this section we will choose $k\ :=\ 3\log\, n$ and assume that $m \geq k$ is large enough.

Let X^i, X^j be inputs of an $\mathtt{LTCM}_{m,k}$. If the prefixes of length k of X^i, X^j are different then the module can obtain the compare info in $O(k)$ steps. In this case, we say that it *gets the result fast,* otherwise it gets the result *slowly*.

Using the function σ_k introduced at the end of Section 2, we define a starting line $\mathcal{S}_k$ for a sorting circuit C as follows. For $i \in [1..n]$ and $j \in [1..m]$ let $x_{i,j}$ denote the gate that gets the j-th input bit of the i-th number X^i, and $y_{i,j}$ the corresponding output gate. Then $\mathcal{S}_k(x_{i,j}) \ =\ \sigma_k(j)$. Note that an input sequence $X = X^1, \ldots, X^n$ with $\mathtt{con}_k(X) = 1$ can be sorted by comparing the prefixes of length k and exchange the remaining part of the strings according to the compare info of these prefixes.

Lemma 5. Let C be a circuit for the (n,m)-sorting problem that is obtained from an arbitrary word comparator circuit C_n by implementing its comparators as $\mathtt{LTCM}_{m,k}$. Then with probability at least $1 - 1/n$, for every output gate $y_{i,j}$ of C it holds $\mathtt{time}^C_{\mathcal{S}_k}(y_{i,j}, \mathcal{X}^{n,m}) \ \leq\ \sigma_k(j) + \mathtt{depth}(C_n)$.

From this lemma we get that all output gates can compute their values by time step $k + \mathtt{depth}(C_n) + 1$. This proves Theorem 1.

For circuits with fanout at most 2 one obains a slightly worse estimation of the form

$$\mathtt{time}^C_{\mathcal{S}_k}(y_{i,j}, \mathcal{X}^{n,m}) \;\le\; \begin{cases} \sigma_k(j) + \mathtt{depth}(C_n) & \text{if } j \le k, \\ \sigma_k(j) + \mathtt{depth}(C_n) + \log(m-k) & \text{else.} \end{cases}$$

In the rest of this section we will concentrate on particular sorting and merging circuits, namely on odd-even merge and bitonic architectures. We start by considering the (n, m)-merging problem for binary numbers of length m.

Lemma 6. Let C_n be an odd-even-merge word comparator circuit merging two sorted m-bit sequences of $n/2$ elements each. For $k \le m$ let C be derived from C_n by replacing its comparators by $\mathtt{LTCM}_{m,k}$. Then for every integer $\gamma \ge 1$, every input X with $\mathtt{con}_{k+1}(X) = \gamma$ and $\mathtt{conf}_{k+1}(X) = \beta$ and for every output gate $y_{i,j}$ of C it holds

$$\mathtt{time}^C_{\mathcal{S}_k}(y_{i,j}, X) \;\le\; \begin{cases} \sigma_k(j) + \log n & \text{if } j \le k, \\ \sigma_k(j) + \log n + \log(m-k) \cdot (\beta + \log \gamma) & \text{else.} \end{cases}$$

The proof of the lemma above is based on the following properties of odd-even-merge circuits:

- Let X be an input of length n for odd-even -merge and X' be one of the two sorted subsequences of length $n/2$. Then within the first ℓ steps of the recursive problem division X' is partitioned into 2^ℓ subsequences $X'_1, \ldots, X'_{2^\ell}$.
- Let $B_1, \ldots, B_r$ be a partition of X' into consecutive strings. After $\log \max_i |B_i|$ recursive steps every subsequence X'_i contains at most one element from each B_j.
- Every pair of input strings X^i and X^j of X is compared at most once.

Theorem 3 (Odd-Even Merge). Let C_n be an odd-even-merge word comparator circuit merging two sorted m-bit sequences of $n/2$ elements. Let C a Boolean circuit derived from C_n by implementing its comparators as $\mathtt{LTCM}_{m,k}$ modules. Given a sequence $\mathcal{X}^{n,m}$, let $Z_1, \ldots, Z_{n/2}$ be a permutation of the subsequence $X^1, \ldots, X^{n/2}$ sorted in nondecreasing order, and similarly $Z^{n/2+1}, \ldots, Z^n$ for $X^{n/2+1}, \ldots, X^n$. Then with probability at least $1 - 1/n$: $\mathtt{time}^C(Z^1, \ldots, Z^n) \;\le\; 5 \cdot \log n$.

The proof follows from the Congestion-Bound and the lemma above. This theorem implies also the result for the sorting problem as stated in Theorem 2 in Section 1. A similar bound can be obtained for bitonic circuits.

5 Average Case Delay for Nonuniform Distributions

This section will extend the previous results to nonuniform distributions. We have to bound the complexity of distributions somehow, because otherwise the average case would equal the worst case. This will be done within the circuit model itself.

Definition 7. A *distribution generating circuit* is a Boolean circuit D of fanin and fanout at most 2. If D has r input gates and n output gates it performs a transformation of a random variable $\mathcal{Z}$ uniformly distributed over $\{0,1\}^r$ into a random variable $\mathcal{X}$ over $\{0,1\}^n$. The input vector for D is chosen according to $\mathcal{Z}$, and the distribution of $\mathcal{X}$ is given by the distribution of the values obtained at the output gates. □

In the following we will identify a distribution over $\{0,1\}^{n\cdot m}$ with a corresponding random vector variable $\mathcal{X}$. Let $\mathcal{X} = (X^1, \ldots, X^n)$ with $X^i = X^i_1 \ldots X^i_m \in \{0,1\}^m$.

Definition 8. Let $\mathcal{D}_{n,m}$ denote the set of all probability distributions μ on $\{0,1\}^{n\cdot m}$. For $\mu \in \mathcal{D}_{n,m}$ let $\mathrm{Supp}(\mu)$ be the set of all vectors $X \in \{0,1\}^{n\cdot m}$ with nonzero probability $\mu(X)$. We call a distribution in $\mathcal{D}_{n,m}$ *strictly positive* if $\mathrm{Supp}(\mu) = \{0,1\}^{n\cdot m}$ and let $\mathcal{D}^+_{n,m}$ denote the set of such distributions. Finally define

$$\mathtt{Depth}_{n,m}(d) := \{\mu \in \mathcal{D}^+_{n,m} \mid \exists \text{ an } r\text{-input and } (n\cdot m)\text{-output Boolean circuit } D \text{ of depth } d \text{ that transforms a uniformly distributed random variable } \mathcal{Z} \text{ over } \{0,1\}^r \text{ into a random variable } \mathcal{X} \text{ with distribution } \mu, \text{ where } r \text{ may be any integer}\}\ .$$ □

By definition, $\mathtt{Depth}_{n,m}(d)$ contains strictly positive probability distributions only. In our setting where a single circuit should have good average case behaviour for every distribution in this class this is obviously necessary to exclude trivial cases. Otherwise one could concentrate the probability mass on the worst-case inputs and average case complexity would equal worst-case complexity. The same problem would arise if the distribution generating circuits may use gates of unbounded fanin or fanout.

To guarantee small average delay the congestion has to be low as seen above. Below we establish a bound on the congestion of a random variable generated by a circuit of small depth.

Lemma 7. Let $\mathcal{X} \in \mathtt{Depth}_{n,m}(d)$ and $c \geq 3 \cdot 2^{2^{d+1}+2d+1} \log n$. Then it holds $\Pr[\mathtt{con}_c(\mathcal{X}) \geq 2] \leq \frac{1}{n}$ and $\Pr[\mathtt{conf}_c(\mathcal{X}) \geq 1] \leq \frac{1}{n}$.

For small d, i.e. $d = \mathrm{loglog}\log n$, the bound given in Lemma 7 implies $\Pr[\mathtt{con}_c(\mathcal{X}) \geq 2] \leq \frac{1}{n}$ for $c \in \Theta(\log^2 n \cdot \mathrm{loglog}\, n)$. One should note that even with such a small depth bound d one can construct highly biased bits x (for example such that $\Pr[x = 1] = 1/\log n$) and also a lot of dependencies among subsets of bits.

Theorem 4. Let $C_{n,m}$ be a Boolean circuit for the (n,m)-sorting problem derived from the word comparator odd-even merge sort circuit C_n by replacing its comparators by a specific family of comparator modules $\mathtt{CM}$. Then for $\mathcal{X} \in \mathtt{Depth}_{n,m}(\mathrm{loglog}\log n)$, with probability greater than $1 - 1/n$ it holds $\mathtt{time}^{C_{n,m}}(\mathcal{X}) \leq 5 \log^2 n \ \mathrm{loglog}\log n$.

That a tiny depth bound is indeed necessary can be seen as follows. For $d = \mathrm{loglog}\, n$ one can construct $\mathcal{X} \in \mathtt{Depth}_{n,m}(d)$ such that $\Pr[\mathtt{con}_{m^{\varepsilon/2}}(\mathcal{X}) \geq n^{\varepsilon/2}] \geq \frac{1}{2}$ for some $\varepsilon > 0$. In this case a larger delay has to occur even in line tree comparator modules.

6 Conclusion

We have presented new topologies for bit level comparators. Using these modules to replace the comparators of a word level sorting circuit yields sorting circuits that are highly efficient on the average. For odd-even sorting circuits we could show that one can achieve an average delay on the bit level that is asymptotically the same as on the word level.

The question arises whether simular results can be shown for other computational problems that can be realized on the word as well as on the bit level.

References

[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi, *Sorting in* $c \log n$ *parallel steps*, Combinatorica 3, 1983, 1-19.

[AB93] M. Al-Hajery and K. Batcher, *On the bit-level complexity of bitonic sorting networks*, Proc. 22. Int. Conf. on Parallel Processing, 1993, III.209 – III.213.

[HL00] I. Hatirnaz and Y. Leblebici, *Scalable binary sorting architecture based on rank ordering withlinear area-time complexity*, Proc. 13. IEEE ASIC/SOC Conference, 2000, 369-373.

[J98] A. Jakoby, *Die Komplexität von Präfixfunktionen bezüglich ihres mittleren Zeitverhaltens,* Dissertation, Universität zu Lübeck, 1998.

[JRS94] A. Jakoby, R. Reischuk, and C. Schindelhauer, *Circuit complexity: from the worst case to the average case*, Proc. 26. ACM STOC, 1994, 58-67.

[JRS95] A. Jakoby, R. Reischuk, and C. Schindelhauer, *Malign distributions for average case circuit complexity*, Proc. 12. STACS, 1995, Springer LNCS 900, 628-639.

[JRSW94] A. Jakoby, R. Reischuk, C. Schindelhauer, and S. Weis, *The average case complexity of the parallel prefix problem*, Proc. 21. ICALP, 1994, Springer LNCS 820, 593-604.

[JS01] A. Jakoby, C. Schindelhauer, *Efficient Addition on Field Programmable Gate Arrays*, Proc. 21. FSTTCS, 2001, 219-231.

[LDH03] Y. Leblebici, T. Demirci, and I. Hatirnaz, *Full-Custom CMOS Realization of a High-Performance Binary Sorting Engine with Linear Area-Time Complexity,* Proc. IEEE Int. Symp. on Circuits and Systems 2003.

[L92] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[LP90] T. Leighton and C. G. Plaxton, *A (fairly) simple circuit that (usually) sorts*, Proc. 31. IEEE FOCS, 1990, 264-274.

[LO99] R. Lin and S. Olariu, *Efficient VLSI architecture for Columnsort*, IEEE Trans. on VLSI 7, 1999, 135-139.

[T79] C.D. Thompson, *Area-Time Complexity for VLSI*, Proc. 11. ACM STOC 1979, 81-88.

[T83] C.D. Thompson, *The VLSI Complexity of Sorting*, IEEE Trans. Comp. 32, 1983, 1171-1184.

Approximating Capacitated Tree-Routings in Networks

Ehab Morsy and Hiroshi Nagamochi

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Yoshida Honmachi, Sakyo, Kyoto 606-8501, Japan
{ehab,nag}@amp.i.kyoto-u.ac.jp

Abstract. The capacitated tree-routing problem (CTR) in a graph $G = (V, E)$ consists of an edge weight function $w : E \to R^+$, a sink $s \in V$, a terminal set $M \subseteq V$ with a demand function $q : M \to R^+$, a routing capacity $\kappa > 0$, and an integer edge capacity $\lambda \geq 1$. The CTR asks to find a partition $\mathcal{M} = \{Z_1, Z_2, \ldots, Z_\ell\}$ of M and a set $\mathcal{T} = \{T_1, T_2, \ldots, T_\ell\}$ of trees of G such that each T_i spans $Z_i \cup \{s\}$ and satisfies $\sum_{v \in Z_i} q(v) \leq \kappa$. A subset of trees in $\mathcal{T}$ can pass through a single copy of an edge $e \in E$ as long as the number of these trees does not exceed the edge capacity λ; any integer number of copies of e are allowed to be installed, where the cost of installing a copy of e is $w(e)$. The objective is to find a solution $(\mathcal{M}, \mathcal{T})$ that minimizes the installing cost $\sum_{e \in E} \lceil |\{T \in \mathcal{T} \mid T \text{ contains } e\}|/\lambda \rceil w(e)$. In this paper, we propose a $(2 + \rho_{\mathrm{ST}})$-approximation algorithm to the CTR, where ρ_{ST} is any approximation ratio achievable for the Steiner tree problem.

Keywords: Approximation Algorithm, Graph Algorithm, Routing Problems, Network Optimization, Tree Cover.

1 Introduction

We propose an extension model of routing problems in networks which includes a set of important routing problems as its special cases. This extension generalizes two different routing protocols in networks. Given an edge-weighted graph, a set of terminals with demands > 0, a designated vertex s, and a number $\kappa > 0$, the first protocol consists in finding a set $\mathcal{T}$ of trees rooted at s each of which contains terminals whose total demand does not exceed κ. Each terminal must be assigned to a tree in $\mathcal{T}$. The goal is to minimize the total cost of all trees in $\mathcal{T}$. Given a network with a designated vertex s and an edge capacity λ, the second protocol consists in finding a set $\mathcal{P}$ of paths P_v from each terminal v to s. Each terminal v has a demand > 0 which should be routed via P_v to s. A subset of paths of $\mathcal{P}$ can pass through an edge in the underlying network as long as the total demand of the paths in this subset does not exceed the capacity λ. For any edge e, any number of copies of e can be installed. The goal is to

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 342–353, 2007.
© Springer-Verlag Berlin Heidelberg 2007

minimize the total weight of all edges installed in the network. These protocols play an important role in many applications such as communication networks supporting multimedia and the design of telecommunication and transportation networks. Our new problem formulation can be applied to possible extensions of these applications.

In this paper, we consider a capacitated routing problem under a multi-tree model. Under this model, we are interested in constructing a set $\mathcal{T}$ of tree-routings that connects given terminals to a sink s in a network with a routing capacity $\kappa > 0$ and an edge capacity $\lambda > 0$. A network is modeled with an edge-weighted undirected graph. Each terminal has a demand > 0, and a tree in the graph can connect a subset of terminals to s if the total demands in the subset does not exceed the routing capacity κ. The weight of an edge in a network stands for the cost of installing a copy of the edge. A subset of trees can pass through a single copy of an edge e as long as the number of these trees does not exceed the edge capacity λ; any integer number of copies of e are allowed to be installed. The goal is to find a feasible set of tree-routings that minimizes the total weight of edges installed in the network. We call this problem the *capacitated tree-routing problem* (CTR for short), which can be formally stated as follows, where we denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively, and R^+ denotes the set of nonnegative reals.

Capacitated Tree-Routing Problem (CTR)
Input: A graph G, an edge weight function $w : E(G) \to R^+$, a sink $s \in V(G)$, a set $M \subseteq V(G)$ of terminals, a demand function $q : M \to R^+$, a routing capacity $\kappa > 0$, and an integer edge capacity $\lambda \geq 1$.
Feasible solution: A partition $\mathcal{M} = \{Z_1, Z_2, \ldots, Z_\ell\}$ of M and a set $\mathcal{T} = \{T_1, T_2, \ldots, T_\ell\}$ of trees of G such that $Z_i \cup \{s\} \subseteq V(T_i)$ and $\sum_{v \in Z_i} q(v) \leq \kappa$ hold for each i.
Goal: Minimize the sum of weights of edges to be installed under the edge capacity constraint, that is,

$$\sum_{e \in E(G)} h_{\mathcal{T}}(e) w(e),$$

where $h_{\mathcal{T}}(e) = \lceil |\{T \in \mathcal{T} \mid e \in E(T)\}| / \lambda \rceil$, $e \in E$.

The CTR is our new problem formulation which includes several important routing problems as its special cases. First of all, the CTR with $\lambda = 1$ and $\kappa = +\infty$ is equivalent to the *Steiner tree problem*. Given an edge-weighted graph G and a subset $Z \subseteq V(G)$, the Steiner tree problem asks to find a minimum weighted tree T of G with $Z \subseteq V(T)$. The Steiner tree problem is NP-hard, and the current best approximation ratio for the Steiner tree problem is about 1.55 [6].

Secondly the CTR is closely related to the *capacitated network design problem* (CND), which has received a number of attentions in the recent study [2,4,7]. The problem is described as follows.

Capacitated Network Design Problem (CND)
Input: A graph G, an edge weight function $w : E(G) \to R^+$, a sink $s \in V(G)$, a

set $M \subseteq V(G)$ of sources, a demand function $q : M \to R^+$, and an integer edge capacity $\lambda \geq 1$.
Feasible solution: A set $\mathcal{P} = \{P_v \mid v \in M\}$ of paths of G such that $\{s, v\} \subseteq V(P_v)$ holds for each $v \in M$.
Goal: Minimize the sum of weights of edges to be installed, that is,

$$\sum_{e \in E(G)} h_{\mathcal{P}}(e) w(e),$$

where $h_{\mathcal{P}}(e) = \lceil \sum_{v:e \in E(P_v)} q(v)/\lambda \rceil$, $e \in E$.

Salman et al. [7] designed a 7-approximation algorithm for the CND by using approximate shortest path trees. Afterwards Hassin et al. [2] gave a $(2 + \rho_{\mathrm{ST}})$-approximation algorithm, where ρ_{ST} is any approximation ratio achievable for the Steiner tree problem. By using of a slight intricate version of this algorithm, they improved the approximation ratio to $(1 + \rho_{\mathrm{ST}})$ when every source has unit demand. Note that the CTR and the CND are equivalent in the case where $\kappa = 1$ and $q(v) = 1$ for every $v \in M$.

The third variant of the CTR is the *capacitated multicast tree routing problem* (CMTR) which can be formally stated as follows.

Capacitated Multicast Tree Routing Problem (CMTR)
Input: A graph G, an edge weight function $w : E(G) \to R^+$, a source s, a set $M \subseteq V(G)$ of terminals, a demand function $q : M \to R^+$, and a routing capacity $\kappa > 0$.
Feasible solution: A partition $\mathcal{M} = \{Z_1, Z_2, ..., Z_\ell\}$ of M and a set $\mathcal{T} = \{T_1, T_2, ..., T_\ell\}$ of trees induced on vertices of G such that $Z_i \cup \{s\} \subseteq V(T_i)$ and $\sum_{v \in Z_i} q(v) \leq \kappa$ hold for each i.
Goal: Minimize

$$\sum_{e \in E(G)} h_{\mathcal{T}}(e) w(e) = \sum_{T_i \in \mathcal{T}} w(T_i),$$

where $h_{\mathcal{T}}(e) = |\{T \in \mathcal{T} \mid e \in E(T)\}|$, $e \in E$, and $w(T_i)$ denotes the sum of weights of all edges in T_i.

Observe that the CMTR is equivalent to the CTR with $\lambda = 1$. For the CMTR with a general demand, a $(2 + \rho_{\mathrm{ST}})$-approximation algorithm is known [3]. If $q(v) = 1$ for all $v \in M$, and κ is a positive integer in an instance of the CMTR, then we call the problem of such instances the *unit demand case* of the CMTR. For the unit demand case of the CMTR, Cai et al. [1] gave a $(2 + \rho_{\mathrm{ST}})$-approximation algorithm, and Morsy and Nagamochi [5] recently proposed a $(3/2 + (4/3)\rho_{\mathrm{ST}})$-approximation algorithm.

As observed above, the CTR is a considerably general model for routing problems. In this paper, we prove that the CTR admits a $(2 + \rho_{\mathrm{ST}})$-approximation algorithm. For this, we derive a new result on tree covers in graphs.

The rest of this paper is organized as follows. Section 2 introduces some notations and two lower bounds on the optimal value of the CTR. Section 3 describes some results on tree covers. Section 4 presents our approximation algorithm for the CTR and analyzes its approximation factor. Section 5 concludes.

2 Preliminaries

This section introduces some notations and definitions. Let G be a simple undirected graph. We denote by $V(G)$ and $E(G)$ the sets of vertices and edges in G, respectively. For two subgraphs G_1 and G_2 of a graph G, let $G_1 + G_2$ denote the subgraph induced from G by $E(G_1) \cup E(G_2)$. An edge-weighted graph is a pair (G, w) of a graph G and a nonnegative weight function $w : E(G) \to R^+$. The length of a shortest path between two vertices u and v in (G, w) is denoted by $d_{(G,w)}(u, v)$. Given a vertex weight function $q : V(G) \to R^+$ in G, we denote by $q(Z)$ the sum $\sum_{v \in Z} q(v)$ of weights of all vertices in a subset $Z \subseteq V(G)$.

Let T be a tree. A *subtree* of T is a connected subgraph of T. A set of subtrees in T is called a *tree cover* of T if each vertex in T is contained in at least one of the subtrees. For a subset $X \subseteq V(T)$ of vertices, let $T\langle X \rangle$ denote the minimal subtree of T that contains X (note that $T\langle X \rangle$ is uniquely determined).

Now let T be a rooted tree. We denote by $L(T)$ the set of leaves in T. For a vertex v in T, let $Ch(v)$ and $D(v)$ denote the sets of children and descendants of v, respectively, where $D(v)$ includes v. A *subtree* T_v *rooted* at a vertex v is the subtree induced by $D(v)$, i.e., $T_v = T\langle D(v) \rangle$. For an edge $e = (u, v)$ in a rooted tree T, where $u \in Ch(v)$, the subtree induced by $\{v\} \cup D(u)$ is denoted by T_e, and is called a *branch* of T_v. For a rooted tree T_v, the *depth* of a vertex u in T_v is the length (the number of edges) of the path from v to u.

The rest of this section introduces two lower bounds on the optimal value to the CTR. The first lower bound is based on the Steiner tree problem.

Lemma 1. *Given a* CTR *instance* $I = (G, w, s, M, q, \kappa, \lambda)$*, the minimum cost of a Steiner tree to* $(G, w, M \cup \{s\})$ *is a lower bound on the optimal value to the* CTR *instance* I*.*

Proof. Consider an optimal solution $(\mathcal{M}^*, \mathcal{T}^*)$ to the CTR instance I. Let $E^* = \cup_{T' \in \mathcal{T}^*} E(T') (\subseteq E(G))$, i.e., the set of all edges used in the optimal solution. Then the edge set E^* contains a tree T that spans $M \cup \{s\}$ in G. We see that the cost $w(T)$ of T in G is at most that of the CTR solution. Hence the minimum cost of a Steiner tree to $(G, w, M \cup \{s\})$ is no more than the optimal value to the CTR instance I. □

The second lower bound is derived from an observation on the distance from vertices to sink s.

Lemma 2. *Let* $I = (G, w, s, M, q, \kappa, \lambda)$ *be an instance of* CTR*. Then,*

$$\sum_{v \in M} d_{(G,w)}(s, v) q(v) / (\kappa \lambda)$$

is a lower bound on the optimal value to the CTR *instance* I*.*

Proof. Consider an optimal solution $(\mathcal{M}^* = \{Z_1, \ldots, Z_p\}, \mathcal{T}^* = \{T_1, \ldots, T_p\})$ to the CTR instance I. Let $opt(I) = \sum_{e \in E(G)} h_{\mathcal{T}^*}(e) w(e)$ be the optimal value

of the CTR instance I. Since $|\{T' \in \mathcal{T}^* \mid e \in E(T')\}| \leq \lambda h_{\mathcal{T}^*}(e)$ holds for all $e \in E(G)$, we see that

$$\sum_{T_i \in \mathcal{T}^*} w(T_i) \leq \sum_{e \in E(G)} \lambda h_{\mathcal{T}^*}(e) w(e) = \lambda opt(I). \tag{1}$$

On the other hand, for each tree $T_i \in \mathcal{T}^*$, we have

$$\sum_{v \in Z_i} d_{(G,w)}(s,v) q(v) \leq w(T_i) \sum_{v \in Z_i} q(v) \leq \kappa w(T_i), \tag{2}$$

since $w(T_i) \geq d_{(G,w)}(s,v)$ for all $v \in V(T_i)$. Hence by summing (2) overall trees in $\mathcal{T}^*$ and using (1), we conclude that $\sum_{v \in M} d_{(G,w)}(s,v) q(v) \leq \kappa \lambda opt(I)$, which completes the proof. □

3 Tree Cover

The purpose of this section is to present some results on the existence of tree covers, based on which we design our approximation algorithm to the CTR in the next section.

We first review a basic result on tree covers.

Lemma 3. [3] Given a tree T rooted at r, an edge weight function $w : E(T) \to R^+$, a terminal set $M \subseteq V(T)$, a demand function $q : M \to R^+$, and a routing capacity κ with $\kappa \geq q(v)$, $v \in M$, there is a partition $\mathcal{Z} = \mathcal{Z}_1 \cup \mathcal{Z}_2$ of M such that:

(i) For each $Z \in \mathcal{Z}$, there is a child $u \in Ch(r)$ such that $Z \subseteq V(T_u)$. Moreover, $|\{Z \in \mathcal{Z}_1 \mid Z \subseteq V(T_u)\}| \leq 1$ for all $u \in Ch(r)$;
(ii) $q(Z) < \kappa/2$ for all $Z \in \mathcal{Z}_1$;
(iii) $\kappa/2 \leq q(Z) \leq \kappa$ for all $Z \in \mathcal{Z}_2$; and
(iv) Let $\mathcal{T} = \{T\langle Z \cup \{r\}\rangle \mid Z \in \mathcal{Z}_1\} \cup \{T\langle Z\rangle \mid Z \in \mathcal{Z}_2\}$. Then $E(T') \cap E(T'') = \emptyset$ for all $T', T'' \in \mathcal{T}$, and hence $\sum_{T' \in \mathcal{T}} w(T') \leq w(T)$.

Furthermore, such a partition $\mathcal{Z}$ can be obtained in polynomial time. □

From the construction of a partition $\mathcal{Z}$ in Lemma 3, the following corollary is straightforward.

Corollary 1. *Let $\mathcal{Z} = \mathcal{Z}_1 \cup \mathcal{Z}_2$ be defined as in Lemma 3 to (T, r, w, M, q, κ). Then:*

(i) *$E(T\langle Z\rangle) \cap E(T\langle \cup_{Z \in \mathcal{Z}_1} Z\rangle) = \emptyset$ for all $Z \in \mathcal{Z}_2$.*
(ii) *Let $Z_0 \in \mathcal{Z}_1$ be a subset such that $Z_0 \subseteq V(T_u)$ for some $u \in Ch(r)$. If $\mathcal{Z}' = \{Z \in \mathcal{Z}_2 \mid Z \subseteq V(T_u)\} \neq \emptyset$, then $\mathcal{Z}'$ contains a subset Z' such that $E(T\langle Z_0 \cup Z'\rangle) \cap E(T\langle Z\rangle) = \emptyset$ for all $Z \in \mathcal{Z} - \{Z_0, Z'\}$.* □

We now describe a new result on tree covers. For an edge weighted tree T rooted at s, a set $M \subseteq V(T)$ of terminals, and a vertex weight function $d : M \to R^+$, we want to find a partition $\mathcal{M}$ of M and to construct a set of induced trees $T\langle Z \cup \{t_Z\}\rangle$, $Z \in \mathcal{M}$ by choosing a vertex $t_Z \in V(T)$ for each subset $Z \in \mathcal{M}$, where we call such a vertex t_Z the *hub vertex* of Z. To find a "good" hub vertex t_Z for each $Z \in \mathcal{M}$, the following lemma classifies a partition $\mathcal{M}$ of M into disjoint collections $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_f$ and then computes t_Z, $Z \in \mathcal{M}$, such that $t_Z = \text{argmin}\{d(t) \mid t \in \cup_{Z \in \mathcal{C}_j} Z\}$ for each $Z \in \mathcal{C}_j$, $j \leq f-1$, and $t_Z = s$ for each $Z \in \mathcal{C}_f$.

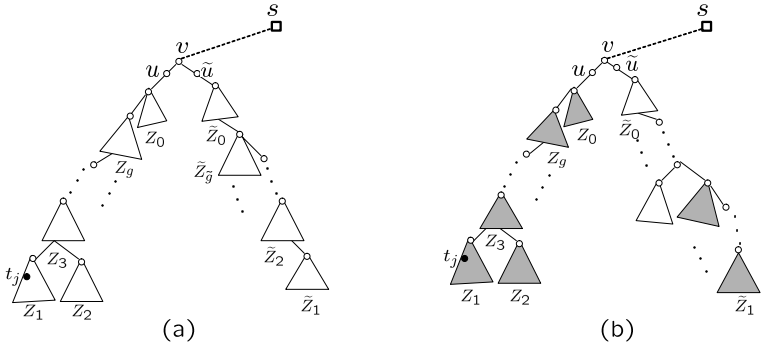


Fig. 1. Illustration of the case of $|\mathcal{Z}_2| = g + \tilde{g} \geq \lambda$ in an iteration of algorithm TreeCover; (a) Line 3.4 identifies a terminal $t_j \in V(T_v)$ with the minimum vertex weight d, where $t_j \in V(T_u)$ in this figure; (b) Line 3.6.3 or 3.6.4 constructs $\mathcal{C}_j$ that contains all subsets in $\{Z_0, Z_1, \ldots, Z_g\}$ and some subsets in $\{\widetilde{Z}_1, \ldots, \widetilde{Z}_{\tilde{g}}\}$ so that $|\mathcal{C}_j| = \lambda$, where the gray subtrees indicate the subsets in $\mathcal{C}_j$. Line 5 then removes all the terminals in $\cup_{Z \in \mathcal{C}_j} Z$ from the terminal set M, and hence no vertices of $V(T_u)$ will be chosen as hub vertices in the subsequent iterations.

Lemma 4. *Given a tree T rooted at s, an edge weight function $w : E(T) \to R^+$, a terminal set $M \subseteq V(T)$, a demand function $q : M \to R^+$, a vertex weight function $d : M \to R^+$, a real κ with $\kappa \geq q(v)$, $v \in M$, and a positive integer λ, there exist a partition $\mathcal{M} = \cup_{1 \leq j \leq f} \mathcal{C}_j$ of M, and a set $\mathcal{B} = \{t_j = \text{argmin}\{d(t) \mid t \in \cup_{Z \in \mathcal{C}_j} Z\} \mid j \leq f-1\} \cup \{t_f = s\}$ of hub vertices such that:*

(i) $|\mathcal{C}_j| \leq \lambda$ *for all* $j = 1, 2, \ldots, f$;
(ii) $q(Z) \leq \kappa$ *for all* $Z \in \mathcal{C}_j$, $j = 1, 2, \ldots, f$;
(iii) $\sum_{Z \in \mathcal{C}_j} q(Z) \geq \kappa\lambda/2$ *for all* $j = 1, 2, \ldots, f-1$;
(iv) $E(T\langle Z\rangle) \cap E(T\langle Z'\rangle) = \emptyset$ *for all distinct* $Z, Z' \in \mathcal{M}$; *and*
(v) *Let* $\mathcal{T}' = \{T\langle Z \cup \{t_j\}\rangle \mid Z \in \mathcal{C}_j,\ 1 \leq j \leq f\}$, *and let all edges of each* $T\langle Z \cup \{t_j\}\rangle \in \mathcal{T}'$, $Z \in \mathcal{C}_j$, $1 \leq j \leq f$ *be directed toward* t_j. *Then for each edge* $e \in E(T)$, *the number of trees in* $\mathcal{T}'$ *passing through* e *in each direction is at most* λ.

Furthermore, a tuple $(\mathcal{M}, \mathcal{B}, \mathcal{T}')$ *can be computed in polynomial time.* □

To prove Lemma 4, we can assume without loss of generality that in a given tree T, (i) all terminals are leaves, i.e., $M = L(T)$, and (ii) $|Ch(v)| = 2$ holds for every non-leaf $v \in V(T)$, i.e., T is a binary tree rooted at s, by splitting vertices of degree more than 3 with new edges of weight zero [5]. We prove Lemma 4 by showing that the next algorithm actually delivers a desired tuple $(\mathcal{M}, \mathcal{B}, \mathcal{T}')$. The algorithm constructs collections $\mathcal{C}_1, \mathcal{C}_2, \ldots$, by repeating a procedure that first chooses a certain vertex v in the current tree, computes a partition $\mathcal{Z}$ of the set of terminals in the subtree rooted at v by Lemma 3, and then selects several subsets in $\mathcal{Z}$ to form the next new collection $\mathcal{C}_j$.

Algorithm. TreeCover
Input: A binary tree $\widehat{T}$ rooted at s, an edge weight function $w : E(\widehat{T}) \to R^+$, a terminal set $M = L(\widehat{T})$ with $q(v) \geq 0$, $v \in M$, a vertex weight function $d : M \to R^+$, a routing capacity κ with $\kappa \geq q(v)$, $v \in M$, and a positive integer λ.
Output: A tuple $(\mathcal{M}, \mathcal{B}, \mathcal{T}')$ that satisfies Conditions (i)-(v) in Lemma 4.
Initialize: $T := \widehat{T}$, $\mathcal{T}' := \emptyset$, and $j := 0$.
1. Choose a maximum depth vertex $v \in V(T)$ with $q(V(T_v) \cap M) \geq \kappa\lambda/2$, and let $j := j + 1$.
2. If v is a leaf of T, then let $Z := \{v\}$, $\mathcal{C}_j := \{Z\}$, and $t_j := v$.
3. If v is not a leaf of T, then
   3.1. Denote $Ch(v) = \{u, \tilde{u}\}$ and $Z_v = V(T_v) \cap M$.
   3.2. Find a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ of Z_v by applying Lemma 3 with $(T_v, w, v, Z_v, q, \kappa)$.
   3.3. Denote $\mathcal{Z}_1 = \{Z_0, \widetilde{Z}_0\}$ and $\mathcal{Z}_2 = \{Z_1, \ldots, Z_g\} \cup \{\widetilde{Z}_1, \ldots, \widetilde{Z}_{\tilde{g}}\}$, where $Z_0 \cup Z_1 \cup \cdots \cup Z_g \subseteq V(T_u)$ and $\widetilde{Z}_0 \cup \widetilde{Z}_1 \cup \cdots \cup \widetilde{Z}_{\tilde{g}} \subseteq V(T_{\tilde{u}})$ (see Fig. 1).
   3.4. Let $t_j \in Z_v$ be such that $d(t_j)$ is minimum, where $t_j \in V(T_u)$ w.o.l.g.
   3.5. If the number of subsets in $\mathcal{Z}_2$, i.e., $g + \tilde{g}$, is less than λ, then let $\mathcal{C}_j := \{Z_0 \cup \widetilde{Z}_0, Z_1, \ldots, Z_g, \widetilde{Z}_1, \ldots, \widetilde{Z}_{\tilde{g}}\}$.
   3.6. If the number of subsets in $\mathcal{Z}_2$ is at least λ, then
      3.6.1. Find $Z_b \in \{Z_1, \ldots, Z_g\}$ such that $E(T\langle Z\rangle) \cap E(T\langle Z_0 \cup Z_b\rangle) = \emptyset$ for all $Z \in \mathcal{Z}_1 \cup \mathcal{Z}_2 - \{Z_0, Z_b\}$, by using Corollary 1(ii).
      3.6.2. Let $\tilde{x}_i \in V(T\langle \widetilde{Z}_i\rangle)$, $i = 1, 2, \ldots, \tilde{g}$ be the vertex closest to v in T, where the distance from $\tilde{x}_{i+1}$ to v in T is not larger than that from $\tilde{x}_i$ to v, $1 \leq i \leq \tilde{g} - 1$, w.o.l.g.
      3.6.3. If $q(Z_0 \cup Z_b) \leq \kappa$, then let
      $$\mathcal{C}_j := \{Z_1, \ldots, Z_{b-1}, Z_0 \cup Z_b, Z_{b+1}, \ldots, Z_g\} \cup \{\widetilde{Z}_1, \ldots, \widetilde{Z}_{\lambda-g}\}.$$
      3.6.4. If $q(Z_0 \cup Z_b) > \kappa$, then let
      $$\mathcal{C}_j := \{Z_0, Z_1, Z_2, \ldots, Z_g\} \cup \{\widetilde{Z}_1, \ldots, \widetilde{Z}_{\lambda-g-1}\}.$$
4. For each $Z \in \mathcal{C}_j$, let $t_Z := t_j$ and $\mathcal{T}' := \mathcal{T}' \cup \{T\langle Z \cup \{t_Z\}\rangle\}$.
5. Remove the set of terminals in $\mathcal{C}_j$ from M and let $T := T\langle M \cup \{s\}\rangle$.
6. Repeat steps in lines 1-5 with the current tree T as long as $q(M) \geq \kappa\lambda/2$.
7. Let $f := j + 1$, $t_f := s$, and $\mathcal{C}_f := \emptyset$.
8. If M is not empty, then
   8.1. Find a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ of M by applying Lemma 3 with (T, w, s, M, q, κ).

8.2. Let $\mathcal{C}_f := \{Z_0 \cup \widetilde{Z}_0\} \cup \mathcal{Z}_2$, where $\mathcal{Z}_1 = \{Z_0, \widetilde{Z}_0\}$.
9. Let $\mathcal{M} := \cup_{1 \le j \le f} \mathcal{C}_j$, $\mathcal{B} := \{t_j \mid 1 \le j \le f\}$, and
$\mathcal{T}' := \mathcal{T}' \cup \{T\langle Z \cup \{s\}\rangle \mid Z \in \mathcal{C}_f\}$.
10. Output $(\mathcal{M}, \mathcal{B}, \mathcal{T}')$.

Now we prove that the tuple $(\mathcal{M}, \mathcal{B}, \mathcal{T}')$ output from algorithm TreeCover satisfies Conditions (i)-(v) in Lemma 4.

(i) Clearly, $|\mathcal{C}_j| = 1 \le \lambda$ for any collection $\mathcal{C}_j$ computed in line 2. Consider a collection $\mathcal{C}_j$ computed in line 3.5. We have $|\mathcal{C}_j| = g + \tilde{g} + 1 \le \lambda$ since $g + \tilde{g} < \lambda$. For any collection $\mathcal{C}_j$ computed in line 3.6.3 or 3.6.4, it is easy to see that $|\mathcal{C}_j| = \lambda$ holds. Note that $|\mathcal{Z}_2| < \lambda$ in a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ of the current M computed in line 8.1 since $q(Z) \ge \kappa/2$, $Z \in \mathcal{Z}_2$ and $q(M) < \kappa\lambda/2$. Hence $|\mathcal{C}_f| = |\mathcal{Z}_2| + 1 \le \lambda$ for a collection $\mathcal{C}_f$ computed in line 8.2. This proves (i).

(ii) For a collection $\mathcal{C}_j$ computed in line 2, $q(Z) \le \kappa$, $Z \in \mathcal{C}_j$, by the assumption that $q(v) \le \kappa$ for all $v \in M$. Consider a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ computed in line 3.2 by applying Lemma 3 to $(T_v, w, v, Z_v, q, \kappa)$. Lemma 3(ii)-(iii) implies that $q(Z_0 \cup \widetilde{Z}_0) < \kappa$ and $q(Z) \le \kappa$ for all $Z \in \mathcal{Z}_2$. Furthermore, for a collection $\mathcal{C}_j$ computed in line 3.6.3, we have $q(Z_0 \cup Z_b) \le \kappa$. Hence each subset Z added to $\mathcal{C}_j$ in line 3.5, 3.6.3, or 3.6.4 has demand at most κ. Lemma 3(ii)-(iii) implies also that each subset of $\mathcal{C}_f$ computed in line 8.2 has demand at most κ. This proves (ii).

(iii) This condition holds for a collection $\mathcal{C}_j$ computed in line 2 since $q(v) = q(V(T_v) \cap M) \ge \kappa\lambda/2$. Consider a collection $\mathcal{C}_j$ computed in line 3.5. We have $\sum_{Z \in \mathcal{C}_j} q(Z) = \sum_{Z \in \mathcal{Z}_1 \cup \mathcal{Z}_2} q(Z) = q(Z_v) \ge \kappa\lambda/2$ since $\mathcal{Z}_1 \cup \mathcal{Z}_2$ computed in line 3.2 is a partition of Z_v and $q(Z_v) \ge \kappa\lambda/2$ by using the condition in line 1. For a collection $\mathcal{C}_j$ computed in line 3.6.3, Lemma 3(iii) implies that $\sum_{Z \in \mathcal{C}_j} q(Z) \ge \lambda(\kappa/2)$ since $q(Z) \ge \kappa/2$, $Z \in \mathcal{C}_j$. For a collection $\mathcal{C}_j$ computed in line 3.6.4, we have $\sum_{Z \in \mathcal{C}_j} q(Z) = \sum_{1 \le i \le b-1} q(Z_i) + q(Z_0 \cup Z_b) + \sum_{b+1 \le i \le g} q(Z_i) + \sum_{1 \le i \le \lambda - g - 1} q(\widetilde{Z}_i) > (b-1)\kappa/2 + \kappa + ((g-b) + (\lambda - g - 1))\kappa/2 = \kappa\lambda/2$ since $q(Z_0 \cup Z_b) > \kappa$. This completes the proof of property (iii).

(iv) Consider the execution of the jth iteration of the algorithm. By the construction of $\mathcal{C}_j$ and Lemma 3(iv), we have $E(T\langle Z_1\rangle) \cap E(T\langle Z_2\rangle) = \emptyset$ for all distinct $Z_1, Z_2 \in \mathcal{C}_j$. Moreover, since any collection computed in line 2, 3.5, or 8.2 contains all subsets in a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ of Z_v computed in line 3.2 and by the assumption in line 3.6.2 used in constructing $\mathcal{C}_j$ in line 3.6.3 or 3.6.4, we conclude that $E(T\langle Z'\rangle) \cap E(T\langle M - \cup_{Z \in \mathcal{C}_j} Z\rangle)) = \emptyset$ for all $Z' \in \mathcal{C}_j$. Hence for any distinct subsets $Z_1, Z_2 \in \mathcal{M}$, we have $E(\widehat{T}\langle Z_1\rangle) \cap E(\widehat{T}\langle Z_2\rangle) = \emptyset$ since a partition $\mathcal{M}$ of M output from the algorithm is a union of collections $\mathcal{C}_j$, $j = 1, 2, \ldots, f$. This proves (iv).

Before proving the property (v), we can show the following lemma (the proof is omitted due to space limitation).

Lemma 5. *Let $(\mathcal{M}, \mathcal{B}, \mathcal{T}')$ be a tuple obtained from a binary tree $\widehat{T}$ by algorithm* TreeCover. *Then for each edge $e = (x, y) \in E(\widehat{T})$, where $y \in Ch_{\widehat{T}}(x)$, we have*

(i) *For $\mathcal{T}'_1 = \{\widehat{T}\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, Z \cap V(\widehat{T}_y) \neq \emptyset \neq Z \cap (V(\widehat{T}) - V(\widehat{T}_y))\}$, it holds $|\mathcal{T}'_1| \le 1$;*

(ii) $|\{\widehat{T}\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, Z \subseteq V(\widehat{T}) - V(\widehat{T}_y), t_Z \in V(\widehat{T}_y)\}| \leq \lambda - 1$; *and*

(iii) $|\{\widehat{T}\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, Z \subseteq V(\widehat{T}_y), t_Z \in V(\widehat{T}) - V(\widehat{T}_y)\}| \leq \lambda - |\mathcal{T}_1'|$. □

We are ready to prove property (v) in Lemma 4. Let $e = (x, y)$ be an arbitrary edge of $\widehat{T}$, where $y \in Ch_{\widehat{T}}(x)$. Let all edges of $\widehat{T}\langle Z \cup t_Z\rangle \in \mathcal{T}'$, $Z \in \mathcal{M}$, be directed toward t_Z, and let $\mathcal{T}_1'$ be as defined in Lemma 5. The number of trees in $\mathcal{T}'$ passing through e toward y is at most the sum of the number of trees in $\mathcal{T}_1'$ and trees in $\{\widehat{T}\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, Z \subseteq V(\widehat{T}) - V(\widehat{T}_y), t_Z \in V(\widehat{T}_y)\}$. Similarly, the number of trees in $\mathcal{T}'$ passing through e toward x is at most the sum of the number of subsets in $\mathcal{T}_1'$ and trees in $\{\widehat{T}\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, Z \subseteq V(\widehat{T}_y), t_Z \in V(\widehat{T}) - V(\widehat{T}_y)\}$. Hence Lemma 5(i)-(iii) completes the proof of (v). □

4 Approximation Algorithm to CTR

This section presents an approximation algorithm for an instance $I = (G, w, s, M, q, \kappa, \lambda)$ of the CTR problem based on results on tree covers in the previous section. The basic idea of the algorithm is to compute an approximate Steiner tree T in $(G, w, M \cup \{s\})$, find a tree cover $\mathcal{T}'$ of the tree T such that $|\{T' \in \mathcal{T}' \mid e \in E(T')\}| \leq \lambda$ for each $e \in E(T)$, and finally connect each tree in $\mathcal{T}'$ to s in order to get a tree-routings $\mathcal{T}$ in the instance I.

Algorithm. ApproxCTR
Input: An instance $I = (G, w, s, M, q, \kappa, \lambda)$ of the CTR.
Output: A solution $(\mathcal{M}, \mathcal{T})$ to I.

Step 1. Compute a ρ_{ST}-approximate solution T to the Steiner tree problem in G that spans $M \cup \{s\}$ and then regard T as a tree rooted at s.
Define a function $d : M \to R^+$ by setting

$$d(t) := d_{(G,w)}(s, t), \quad t \in M.$$

Step 2. Apply Lemma 4 to $(T, w, s, M, q, d, \kappa, \lambda)$ to get a partition $\mathcal{M} = \cup_{1 \leq j \leq f} \mathcal{C}_j$ of M, a set $\mathcal{B} = \{t_1, t_2, \ldots, t_f\}$ of hub vertices, where $t_Z = t_j$ for each $Z \in \mathcal{C}_j$, $j = 1, 2, \ldots, f$, and a set $\mathcal{T}' = \{T\langle Z \cup \{t_Z\}\rangle \mid Z \in \mathcal{M}\}$ of subtrees of T that satisfy Conditions (i)-(v) of the lemma.

Step 3. For each edge $e = (u, v)$ of T, $v \in Ch_T(u)$, with $|\{T' \in \mathcal{T}' \mid e \in E(T')\}| > \lambda$,
Define $\mathcal{C}_{in}(e) := \{Z \in \mathcal{M} \mid Z \subseteq V(T) - V(T_v), t_Z \in V(T_v)\}$ and
$\mathcal{C}_{out}(e) := \{Z \in \mathcal{M} \mid Z \subseteq V(T_v), t_Z \in V(T) - V(T_v)\}$.
while $|\{T' \in \mathcal{T}' \mid e \in E(T')\}| > \lambda$ **do**
Choose two arbitrary subsets $Z \in \mathcal{C}_{in}(e)$ and $Z' \in \mathcal{C}_{out}(e)$,
where $Z \in \mathcal{C}_j$ and $Z' \in \mathcal{C}_{j'}$, $1 \leq j, j' \leq f$.
Let $\mathcal{C}_j := (\mathcal{C}_j - \{Z\}) \cup \{Z'\}$, $\mathcal{C}_{j'} := (\mathcal{C}_{j'} - \{Z'\}) \cup \{Z\}$,

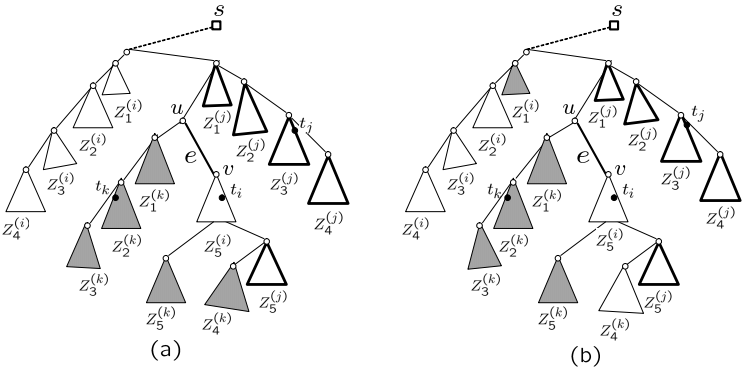


Fig. 2. Illustration of a swapping process in Step 3 of ApproxCTR with $\lambda = 5$; (a) $\mathcal{C}_i = \{Z_1^{(i)}, \ldots, Z_5^{(i)}\}$, $\mathcal{C}_j = \{Z_1^{(j)}, \ldots, Z_5^{(j)}\}$, $\mathcal{C}_k = \{Z_1^{(k)}, \ldots, Z_5^{(k)}\}$, $\mathcal{C}_{in}(e) = \{Z_1^{(i)}, Z_2^{(i)}, Z_3^{(i)}, Z_4^{(i)}\}$, $\mathcal{C}_{out}(e) = \{Z_5^{(j)}, Z_4^{(k)}, Z_5^{(k)}\}$, and t_i, t_j, and t_k are the hub vertices of $\mathcal{C}_i$, $\mathcal{C}_j$, and $\mathcal{C}_k$, respectively. Note that $|\{T\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, e \in E(T\langle Z \cup \{t_Z\}\rangle)\}| = 7 > \lambda$; (b) $\mathcal{C}_i := \mathcal{C}_i - \{Z_1^{(i)}\} \cup \{Z_4^{(k)}\}$, $\mathcal{C}_k := \mathcal{C}_k - \{Z_4^{(k)}\} \cup \{Z_1^{(i)}\}$, $\mathcal{C}_{in}(e) := \mathcal{C}_{in}(e) - \{Z_1^{(i)}\}$, and $\mathcal{C}_{out}(e) := \mathcal{C}_{out}(e) - \{Z_4^{(k)}\}$. Moreover, $\mathcal{T}' := \mathcal{T}' - \{T\langle Z_1^{(i)} \cup \{t_i\}\rangle, T\langle Z_4^{(k)} \cup \{t_k\}\rangle\} \cup \{T\langle Z_4^{(k)} \cup \{t_i\}\rangle, T\langle Z_1^{(i)} \cup \{t_k\}\rangle\}$. Therefore, $|\{T\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, e \in E(T\langle Z \cup \{t_Z\}\rangle)\}| = 5 = \lambda$ for the current $\mathcal{T}'$.

$t_Z := t_{j'}$, and $t_{Z'} := t_j$.
Let $\mathcal{C}_{in}(e) := \mathcal{C}_{in}(e) - \{Z\}$ and $\mathcal{C}_{out}(e) := \mathcal{C}_{out}(e) - \{Z'\}$.
Let $\mathcal{T}' := (\mathcal{T}' - \{T\langle Z \cup \{t_j\}\rangle, T\langle Z' \cup \{t_{j'}\}\rangle\}) \cup \{T\langle Z \cup \{t_Z\}\rangle, T\langle Z' \cup \{t_{Z'}\}\rangle\}$.

(See Fig. 2 for an example that illustrates this process).

Step 4. For each $j = 1, 2, \ldots, f-1$, choose a shortest path $SP(s, t_j)$ between s and t_j in (G, w) and join t_j to s by installing a copy of each edge in $SP(s, t_j)$. Let $\mathcal{T} := \{T\langle Z \cup \{t_Z\}\rangle + SP(s, t_Z) \mid Z \in \mathcal{M}\}$ and output $(\mathcal{M}, \mathcal{T})$.

The idea of Step 3 in ApproxCTR is originated from a procedure of swapping paths in the algorithm for the CND due to Hassin et al. [2].

Consider applying algorithm ApproxCTR to an instance $I = (G, w, s, M, q, \kappa, \lambda)$ of the CTR problem, and let $\mathcal{T}'$ be the set of subtrees of the Steiner tree T computed in Step 2 of the algorithm. For any edge $e = (u, v) \in E(T)$, $v \in Ch(u)$, let $\mathcal{C}_{in}(e)$ and $\mathcal{C}_{out}(e)$ be as defined in the algorithm. Note that Lemma 5 implies that

$$|\mathcal{C}_{in}(e)|, |\mathcal{C}_{out}(e)| \leq \lambda - |\mathcal{T}_1'|,$$

where $\mathcal{T}_1'$ is defined as in the lemma. On the other hand, the number of trees in $\mathcal{T}'$ passing through e in both directions equals $|\mathcal{C}_{in}(e)| + |\mathcal{C}_{out}(e)| + |\mathcal{T}_1'|$. Therefore, $\mathcal{C}_{in}(e) \neq \emptyset$ and $\mathcal{C}_{out}(e) \neq \emptyset$ in the case where the total number of trees in $\mathcal{T}'$ passing through e exceeds λ. In this case, we swap an arbitrary subset $Z \in \mathcal{C}_{in}(e)$ with another subset $Z' \in \mathcal{C}_{out}(e)$, where we assume that Z and Z' belong to collections $\mathcal{C}_j$ and $\mathcal{C}_{j'}$, respectively, and then reassign the

hub vertices of Z and Z' such that $t_Z = t_{j'}$ and $t_{Z'} = t_j$. As a result, $\mathcal{C}_{in}(e)$, $\mathcal{C}_{out}(e)$, $\mathcal{C}_j$, $\mathcal{C}_{j'}$, and $\mathcal{T}'$ are updated as described in the algorithm. This swapping operation decreases the number of trees in $\mathcal{T}'$ passing through each edge in $E(T\langle Z \cup \{t_j\}\rangle) \cap E(T\langle Z' \cup \{t_{j'}\}\rangle)$ (which includes e), where t_j and $t_{j'}$ were the previous hub vertices of Z and Z', respectively, and hence $|\mathcal{C}_{in}(e)|, |\mathcal{C}_{out}(e)| \leq \lambda - |\mathcal{T}'_1|$ still holds. Note that, the number of trees in $\mathcal{T}'$ passing through each of the remaining edges of T never increases. This swapping process is repeated as long as the number of trees in the current $\mathcal{T}'$ passing through e exceeds λ. Hence we proved the following lemma.

Lemma 6. *Let $\mathcal{T}'$ be output by algorithm* ApproxCTR *applied to an instance $I = (G, w, s, M, q, \kappa, \lambda)$ of the* CTR *problem. Then for any edge e of the Steiner tree T, we have $|\{T\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, e \in E(T\langle Z \cup \{t_Z\}\rangle)\}| \leq \lambda$.* □

Next we show the feasibility and compute the approximation factor of the approximate solution $(\mathcal{M}, \mathcal{T})$ output from algorithm ApproxCTR.

Theorem 1. *For an instance $I = (G, w, s, M, q, \kappa, \lambda)$ of the* CTR*, algorithm* ApproxCTR *delivers a $(2+\rho_{\rm ST})$-approximate solution $(\mathcal{M}, \mathcal{T})$, where $\rho_{\rm ST}$ is the approximation ratio of solution T to the Steiner tree problem.*

Proof. Since $\mathcal{M} = \cup_{1 \leq j \leq f} \mathcal{C}_j$, Lemma 4(ii) implies that $q(Z) \leq \kappa$ for all $Z \in \mathcal{M}$. That is, $(\mathcal{M}, \mathcal{T})$ satisfies the routing capacity constraint on each tree. Now we show that $\mathcal{T}$ satisfies the edge capacity constraint, that is, $|\{T' \in \mathcal{T} \mid e \in E(T')\}| \leq \lambda h_{\mathcal{T}}(e)$ for any $e \in E(G)$. Note that each tree in $\mathcal{T}$ is a tree $T\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}'$, $Z \in \mathcal{M}$, plus the shortest path $SP(s, t_Z)$ between s and t_Z in (G, w). By Lemma 6, installing one copy on each edge of the Steiner tree T implies that $|\{T\langle Z \cup \{t_Z\}\rangle \in \mathcal{T}' \mid Z \in \mathcal{M}, e \in E(T\langle Z \cup \{t_Z\}\rangle)\}| \leq \lambda$ for any $e \in E(T)$. On the other hand, each collection $\mathcal{C}_j$, $j \leq f$, contains at most λ subsets of $\mathcal{M}$, all of which are assigned to a common hub vertex t_j. Hence it is enough to install one copy of each edge in a shortest path $SP(s, t_j)$ between s and t_j in (G, w), $j \leq f-1$ ($t_f = s$), in order to get a feasible set $\mathcal{T}$ of tree-routings. This implies that the number of trees in $\mathcal{T}$ passing through a copy of each installed edge on the network is at most λ. Thereby $(\mathcal{M}, \mathcal{T})$ is feasible to I and the total weight of the installed edges on the network is bounded by

$$w(T) + \sum_{1 \leq j \leq f-1} d(t_j).$$

For a minimum Steiner tree T^* that spans $M \cup \{s\}$, we have $w(T) \leq \rho_{\rm ST} \cdot w(T^*)$ and $w(T^*) \leq opt(I)$ by Lemma 1, where $opt(I)$ denotes the weight of an optimal solution to the CTR. Hence $w(T) \leq \rho_{\rm ST} \cdot opt(I)$ holds. To prove the theorem, it suffices to show that

$$\sum_{1 \leq j \leq f-1} d(t_j) \leq 2opt(I). \tag{3}$$

Consider a collection $\mathcal{C}_j$, $j \leq f-1$ obtained by applying Lemma 4 to $(T, w, s, M, q, d, \kappa, \lambda)$ in Step 2. Note that even if some subsets of $\mathcal{C}_j$ are applied by

swapping in Step 3, the hub vertex of the updated collection remains unchanged. That is, the set $\mathcal{B}$ of hub vertices computed in Step 2 does not change throughout the algorithm. Hence Lemma 4(iii) implies that

$$\sum_{t \in Z \in \mathcal{C}_j} q(t)d(t) \geq d(t_j) \sum_{t \in Z \in \mathcal{C}_j} q(t) \geq (\kappa\lambda/2)d(t_j). \tag{4}$$

By summing inequality (4) overall $\mathcal{C}_j$'s (computed in Step 2), $j \leq f-1$, we have

$$(1/2) \sum_{1 \leq j \leq f-1} d(t_j) \leq \sum_{1 \leq j \leq f-1} \sum_{t \in Z \in \mathcal{C}_j} (q(t)/(\kappa\lambda))d(t) \leq \sum_{t \in M} (q(t)/(\kappa\lambda))d(t).$$

Hence Lemma 2 completes the proof of (3). □

5 Conclusion

In this paper, we have studied the capacitated tree-routing problem (CTR), a new routing problem formulation under a multi-tree model which unifies several important routing problems such as the capacitated network design problem (CND) and the capacitated multicast tree routing problem (CMTR). We have proved that the CTR is $(2 + \rho_{\text{ST}})$-approximable based on some new results on tree covers, where ρ_{ST} is any approximation factor achievable for the Steiner tree problem. Future work may include design of approximation algorithms for further extensions of our tree-routing model.

References

1. Z. Cai, G.-H Lin, and G. Xue. Improved approximation algorithms for the capacitated multicast routing problem. LNCS 3595, (2005) 136-145.
2. R. Hassin, R. Ravi, and F. S. Salman. Approximation algorithms for a capacitated network design problem. Algorithmica, **38**, (2004) 417-431.
3. R. Jothi and B. Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. In proceedings of ICALP 2004, LNCS 3142, (2004) 805-818.
4. Y. Mansour and D. Peleg. An approximation algorithm for minimum-cost network design. Tech. Report Cs94-22, The Weizman Institute of Science, Rehovot, (1994); also presented at the DIMACS Workshop on Robust Communication Network, (1998).
5. E. Morsy and H. Nagamochi. An improved approximation algorithm for capacitated multicast routings in networks. In Proceedings of International Symposium on Scheduling 2006, Tokyo, Japan, (2006) 12-17.
6. G. Robins and A. Z. Zelikovsky. Improved Steiner tree approximation in graphs. In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms-SODA'2000, (2000) 770-779.
7. F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. SIAM J. Optim., **11**, (2000) 595-610.

Feedback Arc Set Problem in Bipartite Tournaments

Sushmita Gupta

Department of Computer Science,
Simon Fraser University, Canada
gupta@cs.sfu.ca

Abstract. In this paper we give ratio 4 deterministic and randomized approximation algorithms for Feedback Arc Set problem in bipartite tournaments. We also generalize these results to give a factor 4 deterministic approximation algorithm for Feedback Arc Set problem in multipartite tournaments.

Keywords: Approximation algorithms, feedback arc set, bipartite tournaments.

1 Introduction

This paper deals with approximating feedback arc set problem in multipartite tournaments. A directed graph, $D = (V, A)$, is a *multipartite tournament* if D is a directed orientation of a complete k-partite graph. When $k = 1$, it is an orientation of complete undirected graphs and is known as *tournaments*. For $k = 2$ it is called *bipartite tournaments*. In feedback set problems the task is, given a graph G and a collection $\mathcal{C}$ of cycles in G, to find a minimum size set of vertices or arcs that meets all cycle in $\mathcal{C}$. More precisely the decision version of the problem is defined as follows.

> Feedback Vertex/Arc Set: Given a directed graph $D = (V, A)$, and a positive integer k, is there a subset of at most k vertices (or arcs) whose removal results in an acyclic graph?

Feedback Vertex/Arc Set (FVS/FAS) problem is one of the well known NP-complete problem in general directed graphs and is known to be complete even in special graph class like directed graphs with in-degree and out-degree at most 3. While the FVS problem was known to be NP-complete in tournaments [10], a NP-completeness proof for FAS problem in tournaments eluded us until recently. In 2005, three different groups independently showed FAS to be NP-complete in tournaments [2,3,4]. All the NP-completeness proofs for FAS in tournaments are very different from the usual NP-completeness proofs and use pseudorandom gadgets. The FAS problem has recently been shown to be NP-complete even in bipartite tournaments [7].

There is a long history of approximation algorithms for feedback arc set problem in general directed graphs. Leighton and Rao [8] designed an $O(\log^2 n)$-factor

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 354–361, 2007.
© Springer-Verlag Berlin Heidelberg 2007

approximation algorithms for feedback vertex set. Later, Seymour [9] gave an $O(\log \tau^* \log \log \tau^*)$ approximation algorithm for feedback arc set where τ^* is the size of a minimum feedback vertex set. In 2005, a factor 2.5 randomized approximation algorithm was developed for FAS in tournaments [1]. Later deterministic approximation algorithms with factor 5 and 3 was developed in [5] and [11] respectively.

Here we generalize the approximability results developed for FAS problem in tournaments to bipartite and multipartite tournaments by developing factor 4 deterministic and randomized approximation algorithms. Our results are inspired by the results presented in [1,11].

FAS problem in directed graphs, $D = (V, A)$ can also be phrased as, finding an ordering of vertices (V) such that the arcs from higher indexes to lower indexes, called *backward arcs*, are minimized. Arcs from lower indexes to higher indexes are called *forward arcs*. To see this note that a directed graph D has a topological ordering if and only if D is a directed acyclic graph. We will use this equivalent formulation of FAS problem in our algorithms and will always give an ordering of vertices as our solution. The FAS, F, of D can be formed by taking all backward arcs of this ordering. In proving the upper bounds on the approximation factor of our algorithms, we use as lower bound solutions obtained from linear programming (LP) formulation of some equivalent problems.

The paper is organized as follows. In Section 2 we give randomized factor 4 approximation algorithm for FAS problem in bipartite tournaments. Our deterministic approximation algorithm for FAS in bipartite tournaments is presented in Section 3. In this section we also suggest a simple modification to deterministic approximation algorithm for FAS in bipartite tournaments such that it can be made to work for any multipartite tournaments. Finally we conclude with some remarks in Section 4.

Given a graph $D = (V, A)$ or $T = (V, A)$, n represents the number of vertices, and m represents the number of arcs. For a subset $V' \subseteq V$, by $D[V']$ (or $T[V']$) we mean the subgraph of D (T) induced on V'.

2 Randomized Algorithm for FAS

In this section we give a factor 4 randomized approximation algorithm for FAS in bipartite tournaments. Our algorithm crucially uses the fact that a bipartite tournament has directed cycles if and only if it has a directed 4-cycle. Our randomized algorithm randomly selects an arc and partition the bipartite tournament into two smaller ones around this arc and then solves the problem on these two smaller bipartite tournaments recursively.

In any directed graph, the size of minimum feedback arc set (MFAS) is at least the number of maximum arc disjoint directed cycles in the directed graph (MADC). Since maximum number of arc disjoint 4-cycles (MAD4C) is upper bounded by the size of MADC, we have the following:

$$|MAD4C| \leq |MADC| \leq |MFAS|.$$

`Rand-MFASBT`$(T = (V = (V_1, V_2), A))$

Step 0: If T is a directed acyclic graph then do topological sorting on T and let X be the order of V returned by the topological sorting. **return**(X).
Step 1: Randomly select an arc $(i, j) \in A$, where $i \in V_1,\ j \in V_2$.
Step 2: Form the 2 sets V_L and V_R as follows.

$$V_L = \{u|(u, i) \in A \text{ or } (u, j) \in A\} \ \& \ V_R = \{v|(i, v) \in A \text{ or } (j, v) \in A\}$$

(It is clear that $T[V_L]$ and $T[V_R]$ are bipartite sub-tournaments of T.)
Step 3: **return**(`Rand-MFASBT`$(T[V_L])$,i,j, `Rand-MFASBT`$(T[V_R])$).

Fig. 1. Randomized Algorithm for Feedback Arc Set Problem in Bipartite Tournaments

We use the bound on MAD4C as a lower bound on MFAS in our algorithm and will show that the feedback arc set outputted by the algorithm is at most 4 times the size of MAD4C. This will prove the desired upper bound on the size of solution returned by the algorithm. The detailed description of our algorithm is presented in Figure 1.

Now we show that the algorithm `Rand-MFASBT` is a randomized factor 4 approximation algorithm for feedback arc set in bipartite tournaments.

Theorem 1. *Let $T = (V = (V_1, V_2), A)$ be a bipartite tournament. Then the expected size of the feedback arc set (or the number of backward arcs in the ordering) returned by the algorithm `Rand-MFASBT`(T) is at most 4 times the size of a minimum feedback arc set of T.*

Proof. Let OPT denote the optimal solution of feedback arc set problem for $T = (V_1 \cup V_2, A)$ and C_{PIV} denote the size of backward arcs or feedback arc set returned by `Rand-MFASBT` on T. To prove the theorem we show that:

$$E[C_{PIV}] \leq 4OPT.$$

Observe that an arc $(u, v) \in A$ becomes a backward arc if and only if $\exists (i, j) \in A$ such that (i, j, u, v) forms a directed 4-cycle in T and (i, j) was chosen as the pivot when all 4 were part of the same recursive call. Pivoting on (i, j) will then put v in V_L and u in V_R. Thus making (u, v) a backward arc.

Let R denote the set of directed 4 cycles in T. For a directed 4 cycle r, denote by A_r as the event that one of the arcs of r is chosen as a pivot when all four are part of the same recursive call. Let probability of happening of the event A_r be $Pr[A_r] = p_r$. Define a random variable $X_r = 1$ when A_r occurs and 0 otherwise. Then we get

$$E[C_{PIV}] = \sum_{r \in R} E[X_r] = \sum_{r \in R} p_r.$$

As observed earlier the size of the set of arc disjoint 4-cycles is a lower bound on OPT. This is also true 'fractionally'. More formally we formulate it in the following claim.

Claim. If $\{\beta_r\}$ is a system of non negative weights on 4-cycles in R such that

$$\forall a \in A, \quad \sum_{\{r \mid r \in R,\ a \in r\}} \beta_r \quad \leq 1,$$

then $OPT \geq \sum_{r \in R} \beta_r$.

Proof. Consider the following Covering LP :

$$\begin{array}{c} \min \quad \sum_{a \in A} x_a \\ \sum_{a \in r} x_a \geq 1 \text{ for } r \in R \text{ and} \\ x_a \geq 0,\ \forall a \in A \end{array}$$

This is a LP formulation of hitting all directed 4-cycles of T and hence a solution to it forms a lower bound for OPT which considers hitting all cycles. If we take dual of this linear programming formulation then we get following Packing LP:

$$\begin{array}{c} \max \quad \sum_{r \in R} y_r \\ \sum_{\{r \mid r \in R,\ a \in r\}} y_r \quad \leq 1,\ \forall a \in A \text{ and} \\ y_r \geq 0,\ \forall r \in R \end{array}$$

$\{\beta_r\}$ is a feasible solution to this dual LP and hence a lower bound for Covering LP and so a lower bound on the OPT. □

We find such a packing using the probabilities p_r. Let $r = (i, j, k, l)$ be a directed 4, cycle then

$$\Pr[a = (i, j) \text{ is a pivot of } r \mid A_r] = 1/4,$$

because we choose every edge as a pivot with equal probability. Let $a' = (k, l)$ and $B_{a'}$ be the event that a' becomes the backward arc. If A_r has occurred then a' becomes the backward arc if and only if (i, j) was chosen pivot among $(i, j), (j, k), (k, l), (l, i)$. So this gives us

$$\Pr[B_{a'} \mid A_r] = 1/4.$$

So, $\forall r \in R$ and $a \in r$ we have

$$\Pr[B_a \wedge A_r] = \Pr[B_a | A_r] \cdot \Pr[A_r] = \frac{1}{4} \Pr[A_r] = \frac{1}{4} p_r.$$

For 2 different 4 cycles r_1 and r_2, the events $B_a \wedge A_{r_1}$ and $B_a \wedge A_{r_2}$ are disjoint. Since the moment a becomes a backward arc for r_1 the endpoints of a become part of different recursive sets, so the next event $B_a \wedge A_{r_1}$ can never occur. Hence for any $a \in A$,

$$\sum_{\{r \mid a \in r\}} Pr[B_a \wedge A_r] \leq \max_{\{r \mid a \in r\}} \left(\frac{1}{4} \cdot p_r \right) \leq 1,$$

since only one of the events in the summation can occur.

So $\{\frac{1}{4} p_r\}_{r \in R}$ is a fractional packing of R and hence a lower bound on OPT. So finally we get,

$$OPT \geq \sum_{r \in R} \frac{1}{4} p_r = \frac{1}{4} E[C_{PIV}].$$

So $E[C_{PIV}] \leq 4 \cdot OPT$ is proved as desired. □

3 Deterministic Algorithm for FAS

We now give a deterministic factor 4 approximation algorithm for feedback arc set problem in bipartite tournaments. Our new algorithm is basically a derandomized version of the algorithm presented in the previous section. Here we replace the randomized step of `Rand-MFASBT` by a deterministic step based on a solution of a linear programming formulation of an auxiliary problem associated with feedback arc set problem in bipartite tournaments.

Given a bipartite tournament T, we call the problem of hitting all directed four cycles of T as 4-Cycle Hitting (4CH) problem. Notice that any feedback arc set of a bipartite tournament is also a solution to the 4CH problem. Given a bipartite tournament $T(V, A)$, we associate the following simple integer linear programming formulation to the 4CH problem. Let x_a be a variable for an arc $a \in A$, then

$$\begin{gathered}\min \textstyle\sum_{a \in A} x_a \\ \text{(subject to) } \textstyle\sum_{a \in r} x_a \geq 1 \text{ for all directed four cycles } r \text{ in } T, \\ x_a \in \{0, 1\} \quad \forall a \in A.\end{gathered}$$

For the relaxation of the above LP formulation for 4CH problem we allow $x_a \geq 0$ for all $a \in A$. Let x^* be an optimal solution of this relaxed LP. We will use x^* as a lower bound for minimum feedback arc set in the approximation factor analysis of the algorithm. Notice that when we select an arc (i, j) as pivot for partitioning in our algorithm then all the arcs going from V_R to V_L become backward and these remain the same until the end of the algorithm. Given an arc $q = (i, j)$ and a bipartite tournament T, let

$$R_q = \{(k, l) \mid (i, j, k, l) \text{ is a directed 4 cycle in } T \}.$$

We choose an arc q as pivot such that the size of R_q is 'minimized' and to do so we choose an arc such that:

$$\frac{|R_q|}{\sum_{a \in R_q} x^*_a}$$

is minimized. We present our deterministic algorithm with the above mentioned change in Figure 2.

We will show that the algorithm `Det-MFASBT` is a deterministic factor 4 approximation algorithm for feedback arc set in bipartite tournaments.

Theorem 2. *Let $T = (V = (V_1, V_2), A)$ be a bipartite tournament. Then the size of the feedback arc set (or the number of backward arcs in the ordering) returned by the algorithm `Det-MFASBT(T)` is at most 4 times the size of a minimum feedback arc set of T.*

Proof. Let X be the ordering returned by the algorithm `Det-MFASBT(T)` and let x^* be the optimal solution of the relaxed LP of 4CH problem. Also let B be the set of backward arcs with respect to the ordering X and OPT be the

`Det-MFASBT`$(T = (V = (V_1, V_2), A))$

Step 0: If T is a directed acyclic graph then do topological sorting on T and let X be the order of V returned by the topological sorting. **return**(X).
Step 1: Select an arc $(i, j) \in A$, such that

$$\frac{|R_q|}{\sum_{a \in R_q} x_a^*}$$

is minimized.
Step 2: Form the 2 sets V_L and V_R as follows.

$$V_L = \{u|(u, i) \in A \text{ or } (u, j) \in A\} \;\; \& \;\; V_R = \{v|(i, v) \in A \text{ or } (j, v) \in A\}$$

Step 3: **return**(`Det-MFASBT`$(T[V_L])$,i,j, `Det-MFASBT`$(T[V_R])$).

Fig. 2. Deterministic Algorithm for Feedback Arc Set Problem in Bipartite Tournaments

number of backward arcs in the ordering of T minimizing the backward arcs (size of minimum feedback arc set). Notice that

$$\sum_{a \in A} x_a^* \leq OPT.$$

Let $y_a = 1$ if $a \in B$ and $y_a = 0$ otherwise. Hence $\sum_{a \in A} y_a = \sum_{a \in B} y_a = |B|$. Now we show that

$$|B| = \sum_{a \in B} y_a \leq 4 \sum_{a \in B} x_a^* \leq 4 \sum_{a \in A} x_a^* \leq 4OPT.$$

If we take any $V' \subseteq V$ and consider the relaxed LP formulation of 4CH problem for $T[V']$ then x^* restricted to the arcs in $V[T']$ is a feasible solution to this LP. Notice that an arc (x, y) becomes a backward for the ordering X when we choose an arc (i, j) as pivot and $y \in V_L$ and $x \in V_R$ of the partition based on (i, j) and then remains backward after this as we recurse on the subproblems associated with V_L and V_R. In order to bound the size of backward arcs we show that there always exists a pivot $q = (i, j)$ such that

$$|R_q| \leq 4 \sum_{a \in R_q} x_a^*. \tag{1}$$

Notice that once we show this we are done as follows: we partition the arcs in B based on the pivot which has made it backward, let these partition be $B_{q1}, B_{q2} \cdots B_{ql}$ where qt, $1 \leq t \leq l$, are the arcs chosen as pivot in the algorithm. Then

$$|B| = \sum_{t=1}^{l} |B_t| \leq \sum_{t=1}^{l} 4 \left(\sum_{a \in R_{qt} = B_{qt}} x_a^* \right) = 4 \sum_{a \in B} x_a^* \leq 4OPT.$$

So now it all boils down to showing that we can choose a pivot q such that $|R_q| \leq 4 \sum_{a \in R_q} x^*_a$. Let R be the set of all directed four cycles in T, that is

$$R = \{((i,j),(j,k),(k,l),(l,i)) \text{ a directed 4 cycle in } T\}.$$

Define $x^*(r) = \sum_{a \in r} x_a$ for any $r \in R$. So we get the following :

$$\sum_{a \in A} \sum_{a' \in R_a} \mathbf{1} = \sum_{r \in R} \sum_{a'' \in r} \mathbf{1} = \sum_{r \in R} 4, \text{ and}$$
$$\sum_{a \in A} \sum_{a' \in R_a} x^*_{a'} = \sum_{r \in R} \sum_{a'' \in r} x^*_{a''} = \sum_{r \in R} x^*(r).$$

Notice that for every $r \in R$, we have a constraint in LP formulation of 4CH problem that $\sum_{a \in r} x_a \geq 1$ and hence $x^*(r) \geq 1$ for all $r \in R$. So we have

$$\sum_{r \in R} 4 \leq 4 \left(\sum_{r \in R} x^*(r) \right).$$

This implies the existence of a pivot satisfying the Equation 1. If not then

$$\sum_{r \in R} 4 = \sum_{a \in A} \sum_{a' \in R_a} \mathbf{1} > \sum_{a \in A} 4 \left(\sum_{a' \in R_a} x^*_{a'} \right) = 4 \left(\sum_{r \in R} x^*(r) \right),$$

which is a contradiction. This completes the proof of the theorem and shows that `Det-MFASBT(`T`)` is indeed a deterministic factor 4 approximation algorithm for FAS problem in bipartite tournaments. □

The algorithm `Det-MFASBT(`T`)` can be generalized to multipartite tournaments by doing a simple modification. When we pick an arc $q = (i,j)$ as pivot then we treat the partition containing i as one partition and union of all other parts as another one, making it 'similar' to a bipartite tournament. With this modification we create two smaller instances of multipartite tournaments in our algorithm and then recurse separately. All the analysis done for bipartite tournaments can be now carried over for multipartite tournaments. Without going into the details we state the following theorem.

Theorem 3. *Let $T(V,A)$ be a multipartite tournament. Then the modified `Det-MFASBT(`T`)` is a factor 4 approximation algorithm for FAS in T.*

4 Conclusion

In this paper we generalized the known approximation algorithms for FAS problem in tournaments to bipartite and multipartite tournaments. We gave factor 4 randomized and deterministic approximation algorithms for FAS problem in bipartite tournaments. There are only a few directed graph classes for which constant factor approximation algorithms for FAS problem are known. Here we

add multipartite tournaments to graph classes for which there exists a constant factor approximation algorithm.

Acknowledgment. I thank Dr Ramesh Krishnamurti for the course he taught on approximation algorithms. Couple of papers I read for that course led to this work.

References

1. N. Ailon, M. Charikar and A. Newman. *Aggregating Inconsistent Information: Ranking and Clustering*. In the Proceedings of 37th Annual ACM Symposium on Theory of Computing (STOC), (2005) 684-693.
2. N. Alon. *Ranking Tournaments*. Siam Journal on Discrete Mathematics, **20(1)**, (2006) 137-142.
3. P. Charbit, S.Thomassé and A.Yeo. *The minimum feedback arc set problem is NP-hard for Tournaments*. To appear in Combinatorics, Probability and Computing.
4. V. Contizer. *Computing Slater rankings using similarities among candidates*. Technical Report **RC23748**, IBM Thomas J Watson Research Centre, NY 2005.
5. D. Coppersmith, L. Fleischer and A. Rudra. *Ordering by weighted number of wins gives a good ranking for weighted tournaments*. In the Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), (2006) 776-782.
6. G. Even, J. Noar, B. Schieber and M. Sudan. *Approximating minimum feedback sets and multicuts in directed graphs*. Algorithmica, **20**, (1998) 151-174.
7. J. Guo, F. Hüffner, H. Moser . *Feedback Arc Set in Bipartite tournaments is NP-Complete*. To appear in Information Processing Letters (IPL).
8. T. Leighton and S. Rao. *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*. Journal of ACM, **46(6)**, (1999) 787-832.
9. P. D. Seymour. *Packing directed circuits fractionally*. Combinatorica, **15**, (1995) 281-288.
10. E. Speckenmeyer. *On Feedback Problems in Digraphs*. In the Proceedings of the 15th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'89), Lecture Notes in Computer Science, **411**, (1989) 218-231.
11. A. van Zuylen. *Deterministic approximation algorithm for clustering problems*. Technical Report **1431**, School of Operation Research and Industrial Engineering, Cornell University.

Studying on Economic-Inspired Mechanisms for Routing and Forwarding in Wireless Ad Hoc Network*

Yufeng Wang[1], Yoshiaki Hori[2], and Kouichi Sakurai[2]

[1] College of Telecommunications and Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[2] Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka 812-0053, Japan

Abstract. Considering the fact that there exist information asymmetry (hidden information) in routing phase, and moral hazard (hidden action) in forwarding phase in autonomous Ad hoc network, this paper argues that economic-based mechanisms play both a signaling and a sanctioning role, which reveal the node's true forwarding cost in routing phase while provide incentives to nodes to exert reasonable effort in forwarding phase, that is, the role of economic-inspired mechanisms in information asymmetry is to induce learning whereas the role of such mechanisms in moral hazard settings is to constrain behavior. Specifically, this paper conducts the following works: considering the mutually dependent link cost, we demonstrate that, for each participant, truth-telling is the risk dominant strategy in VCG-like routing mechanism based on analysis of extensive game form. Then, Individual rationality (IR) and Incentive Compatibility (IC) constraints are formally offered, which should be satisfied by any game theoretical routing and forwarding scheme. And different solution concepts are investigated to characterize the economic meanings of two kind forwarding approaches, that is, Nash equilibrium with no per-hop monitoring and dominant strategy equilibrium with per-hop monitoring.

1 Introduction

Traditional system design assumes that participants behave according to the intentions of the system architects. But wireless ad hoc network is composed by autonomous nodes determining their behaviors independently and shared by rational (selfish) users with different and competing interest. Obviously, Incentive mechanisms are needed to encourage nodes to participate in the responsibility of network infrastructure, like routing and forwarding packet. Ref. [1] illustrates the notion of elementary cooperation, and introduces the incentive pattern as tool to construct incentive schemes for specific application environment. According to remuneration type, incentive mechanisms can be classified into the following types: 1) Reputation-based schemes [2-4]. In such schemes, by keeping monitoring packet forwarding activities,

* Research supported by the NSFC Grants 60472067, JiangSu education bureau (5KJB510091) and State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 362–373, 2007.
© Springer-Verlag Berlin Heidelberg 2007

the misbehaving nodes may be detected and isolated from the rest of the network. The advantage of these schemes is that they do not require central management points, while the disadvantage is that these schemes usually cannot handle well the dynamically changing topology and asymmetric packet forwarding request demands, for example, a node with few packets to send has no incentive to forward all the packets for another node with a lot of packets to send; 2) credit-based schemes [5-7,11,16,17]. In those approaches, principal entity explicitly compensates agent entity for its offered services (In a service oriented perspective, the agent entity is the provider of a service like message forwarding etc, and the principal entity is the consumer). Compared with those reputation-based schemes, the advantage of the payment-based schemes lies in that they can work under various situations, such as the network with dynamically changing topology or asymmetric request demands, while the disadvantage is that they may require some management points to handle billing information. Another idea to model cooperative routing in ad hoc networks is to use the concept of repeated game [8-9]. The intuition in repeated game-based routing is that cooperative behavior emerges as a result of symmetric node interaction patterns (the threat to interact in future will drive nodes cooperate in current period), which is not always true for ad-hoc networks. Ref. [13-14] also discussed the usage of repeated game in network field, but their models are not suitable for multi-hop Ad hoc network.

In this paper, we focus on the economic-based mechanisms. There are several papers similar to our work. Ref. [10] adopts Mechanism Design (MD) to design Ad hoc-VCG routing scheme. But this paper ignores the unique challenge in wireless ad hoc networks that link cost is determined by two neighboring nodes together, namely mutually dependent link cost. Ref. [12] investigates incentive-compatible routing and forwarding protocols, Corsac, in wireless ad hoc network, which integrates VCG with a novel cryptographic technique to address the challenge of mutual-dependent link cost in wireless ad-hoc networks. But this paper provides several wrong arguments which will be discussed in this paper.

Generally, cooperation is compromised by information asymmetries (hidden information in routing phase) and moral hazard (in forwarding phase) in Ad hoc network. Hidden information means that nodes possess private information (e.g., costs for forwarding packet) which is not known to all of the parties involved. For example, in traditional routing protocol which selects low-cost routing path, there is no guarantee that any wireless node will reveal its cost truthfully unless it is convinced that is cannot do better by declaring a different cost. Economic-based mechanisms alleviate the issue of hidden information by acting as signaling devices, which will help the system designer learn the true forwarding cost of each node. In ad hoc network, moral hazard represents what made attractive by a lack of observation. For example, in the forwarding phase, without proper incentive mechanism, each participant in the low-cost path has inventive to drop packet. In brief, selfish wireless nodes may hinder the functioning of the network completely, thus an incentive mechanism is required to encourage the users to provide service to other nodes. Economic-based mechanisms can deter moral hazard by acting as sanctioning devices, which punishes dishonest nodes through specific way. If the present value of punishment exceeds the gains from cheating, then rational nodes have sufficient incentives to cooperate. The most important distinction between (pure) moral hazard

and (pure) adverse selection settings is that, in the former, all sellers are capable of the same type of behavior (e.g. cooperate, cheat), whereas in the latter case seller behavior is completely constrained by its innate "type." The role of economic-inspired mechanisms in pure moral hazard settings is to constrain behavior whereas the role of such mechanisms in pure adverse selection settings is to induce learning. In ac hoc network, economic-based mechanisms play both a sanctioning and a signaling role, which reveal the node's true forwarding cost in routing phase while provide incentives to nodes to exert reasonable effort.

The paper's contributions lie in the following aspects: Firstly, for mutually dependent link cost, based on analysis of extensive game form, we argue that truth-telling is nodes' risk dominant strategy, and repeated game can be used to achieve this equilibrium; Then, this paper formally describes the constraints of individual rationality (IR) and incentive compatibility (IC) in ad hoc routing and forwarding game. Finally, based on those two basic constraints, different solution concepts are used to characterize the features of ad hoc network game with per-hop monitoring and without per-hop monitoring.

The paper is organized as follows. In section 2, system models used in this paper are described, including the cost model of ad hoc nodes, and related constraints and equilibrium solutions in game theory and mechanism. In section 3, based on extensive game analysis, we argue that truth-telling is the risk dominant strategy of each participant in VCG-like routing mechanisms even facing the problem of mutual-dependent link cost. In section 4, the formal definitions of IR and IC in the loss link were offered, and different solution concepts are used to characterize the economic meanings of forwarding games with per-hop monitoring and without per-hop monitoring, that is, Nash equilibrium in no per-hop monitoring and dominant strategy equilibrium in per-hop monitoring. Finally, we briefly conclude the paper in section 5.

2 System Models

2.1 Cost Model

We assume ad hoc network to be an arbitrary, connected graph, $G=\{V,E,\omega\}$, of selfish ad hoc nodes $V=\{1,\ldots,N\}$ that represents mobile devices, a set $E \subseteq V * V$ of directed edges (i,j) that connect two nodes, and a weight function $c: E \to R$ for each edge (i,j) that indicates the cost of transmitting a data packet from node i to node j. Each node i has an individual parameter α_i indicating its cost-of-energy. If node i send a message using emission power P^{emit}, then it must receive at least remuneration amount $\alpha_i * P^{emit}$ to compensate for its energy cost.

An energy-efficient routing protocol ensures that a packet from a source to a destination gets routed along the most cost-efficient path possible via intermediate nodes. The total cost of a routing path is the sum on the energy cost incurred at the source and at each intermediate node. If a sender appends its emission power into the packet header, then the receiver can infer the required minimal power, and forward the value to the sender. That is:

$$P_{i,j}^{\min} = \frac{P_i^{emit}}{P_{i,j}^{rec}} P_{\min}^{rec} \tag{1}$$

P_i^{emit} denotes peer *i*'s emission power; $P_{i,j}^{rec}$ is the received power level by peer *j*; P_{min}^{rec} is the required power level to guarantee certain QoS level. Thus, the cost of transmitting packet from *i* to *j* is given as follows:

$$c(i,j) = \alpha_i P_{i,j}^{\min} \tag{2}$$

For the given path, we omit subscript *j*, let $c(i,j)=c_i$. (3)

It is easily seen that a node alone cannot determine transmission power level because it needs feedbacks from its neighbors, which is the unique challenge of mutually dependent link cost in Ad hoc network. The problem above may allow one node to cheat its neighbors in order to raise its own welfare. In next section, based on analysis of extensive game, we argue that truth-telling is the expected dominant strategy for each node in Ad hoc network.

2.2 Concepts in Mechanism Design

The goal of MD (also called "inverse game") is to define the strategic situation, or "rules of the game", so that the system as a whole exhibits good behavior in equilibrium when self-interested nodes pursue self-interested strategies (A more detailed introduction to mechanism design is provided in [15]). The standard setting of mechanism design is described as follows:

- There exist a set of agents $i \in I$ (with *N* agents altogether), and each agent *i* has private type $\theta_i \in \Theta_i$; A mechanism $m=(o,p)$ is composed two elements, outcome specification $o \in O$ which maps the set of private type to an allowed outcome based on the optimization of social selection function $f(\hat{\theta})$, which defines the system outcomes for each possible set of agent types. For example, the mechanism designer goal is to select the cost-effective path in wireless ad hoc network. $\hat{\theta} \in \Theta = (\Theta_1 \times \cdots \times \Theta_N)$ is vector of reported types by agents. We write $\hat{\theta}$ to emphasize that agents can misreport their true types. Mechanism also calculates *N*-tuple of payment $p = (p_1, \cdots, p_N)$ by (or to) each agent.
- Each agent *i* has utility $u_i(o;\theta_i)$, agent *i*'s utility for report $\hat{\theta}_i$, given others' reports $\hat{\theta}_{-i} = \{\hat{\theta}_1, \cdots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \hat{\theta}_N\}$, is $u_i(o(\hat{\theta}_i, \hat{\theta}_{-i}); \theta_i)$. Generally, utility is quasi-linear in value and payments, defined as $u_i(o;\theta_i) = p_i - c_i(o;\theta_i)$.

Note that, the equation (2) is the specific representation of the second term in the above equation; in routing phase, each node's private type can represent its

transmission cost (hidden information); in forwarding phase, each node's private type can represent the hidden action conducted by peer (like forwarding packet or not).

There are two types of constraints on the designer in building ad hoc routing and forwarding mechanism: individual rationality and incentive compatibility.

Individual Rationality (IR) Constraints

The first type of constraint is the following: the utility of each agent has to be at least as great as the agent's fallback utility, that is, the utility that the agent would receive if it did not participate in the mechanism. Otherwise that agent would not participate in the mechanism. This type of constraint is called an IR (also called participation constraint). There are three different possible IR constraints: ex ante, ex interim, and ex post, depending on what the agent knows about its own type and the others' types when deciding whether to participate in the mechanism [19]. Ex ante IR means that the agent would participate if it knew nothing at all (not even its own type). We will not study this concept in this paper. Ex interim IR means that the agent would always participate if it knew only its own type, but not those of the others. Ex post IR means that the agent would always participate even if it knew everybody's type. We will define Ex interim IR formally. First, we need to formalize the concept of the fallback outcome. We assume that each agent's fallback utility is zero for each one of its types. This is without loss of generality because we can add a constant term to an agent's utility function (for a given type), without affecting the decision-making behavior of that expected utility maximizing agent.

Definition 1. A mechanism is ex interim IR if for any agent *i* and any type of $\theta_i \in \Theta_i$, we have $E_{(\theta_1,\ldots,\theta_{i-1},\theta_{i+1},\ldots,\theta_N)|\theta_i}\left(p_i(\theta_1,\ldots,\theta_N) - c_i(\theta_i, o(\theta_1,\ldots,\theta_N))\right) \geq 0$. which states that the utility to an agent, given prior beliefs about the preferences of other agents, is at least the utility gained from not participating to the mechanism.

Incentive Compatibility (IC) Constraints

The second type of constraint states that the agents should never have an incentive to misreport their type. For this type of constraint, two most common variants (or solution concepts) are implementation in Nash equilibrium, and implementation in dominant strategies,.

Definition 2. A mechanism is said to implement its outcome and payment function in Nash equilibrium, If, given the truth-telling strategy in equilibrium, $(\theta_1^*,\ldots,\theta_i^*,\ldots,\theta_N^*)$, no agent has an incentive to deviate unilaterally from the strategy profile. Formally, for any agent *i*, any type $(\hat{\theta}_1,..,\hat{\theta}_i,..,\hat{\theta}_N) \in \Theta_1 \times \cdots \times \Theta_i \times \cdots \times \Theta_N$, any alternative type report $\hat{\theta}_i \in \Theta_i$ we have $p_i(\theta_1^*,\ldots,\theta_i^*,\ldots,\theta_N^*) - u_i(\theta_i, o(\theta_1^*,\ldots,\theta_i^*,\ldots,\theta_N^*)) \geq p_i(\theta_1^*,\ldots,\hat{\theta}_i,\ldots,\theta_N^*) - u_i(\theta_i, o(\theta_1^*,\ldots,\hat{\theta}_i,\ldots,\theta_N^*))$

Definition 3. A mechanism is said to implement its outcome and payment function in dominant strategies if truth-telling is always optimal even when the types reported by the other agent are already known. Formally, we have $p_i(\hat{\theta}_1,\ldots,\theta_i^*,\ldots,\hat{\theta}_N) - c_i(\theta_i, o(\hat{\theta}_1,\ldots,\theta_i^*,\ldots,\hat{\theta}_N)) \geq p_i(\hat{\theta}_1,\ldots,\hat{\theta}_i,\ldots,\hat{\theta}_N) - c_i(\theta_i, o(\hat{\theta}_1,\ldots,\hat{\theta}_i,\ldots,\hat{\theta}_N))$. The terms involving payments are left out if payment is not possible.

3 Hidden Information and Routing Mechanisms

Ad hoc-VCG mechanism works as follows [10]: based on each node's declaration of forwarding cost, assume $S,1,\ldots,i,j,\ldots,D$ denote the chosen cost-effective path. According to VCG mechanism, the remuneration amount received by node *i* is:

$$p_i := \left|SP^{-i}\right| - \left|SP \setminus (i,j)\right| = \left|SP^{-i}\right| - \left|SP\right| + c(i,j) \tag{4}$$

where $\left|SP^{-i}\right|$ represents the minimal path cost without node *i*; $\left|SP \setminus (i,j)\right|$ represents the minimal path cost, when the cost of link (i, j) equals zero.

Although the VCG mechanism has been applied to many networking problems in general and to routing protocols in particular, wireless ad-hoc networks poses unique challenges, that is, mutually dependent link cost. Ref. [12] adopts cryptography to solve the mutual-dependent link cost. We argue that, based on analysis of extensive game, it needs not to use considerable cryptography-based effort to guarantee the truth-telling property in VCG routing mechanism, even though there exist the mutually dependent link cost problem. We consider an extensive game model of ad hoc routing game. The basic component of the extensive form is the game tree which is a directed graph. The variable *T* is used to denote the game tree and the variables *V* and *E* are used to denote the collections of vertices and edges of the graph, i.e. $T = (V, E)$, the set of players is denoted by *N* and individual players are indexed using the variable *i*. The vertices are further divided into sets of terminal and non-terminal vertices. Terminal vertices have no successors, i.e. child vertices, and they define an outcome for the game, which can be associated with a payoff. Thus the path fully defines the outcome of the game. Each non-terminal vertex $v \in V$ is labeled using one of the players of the game and V_i is defined to be the collection of vertices that are labeled with *i*. At each non-terminal vertex $v \in V_i$, player *i* needs to perform an action. The sequential game played between neighboring node A and B (Assume node B is the successor of node A in the selected path) can be illustrated in Fig.1, each node has two choices: truthful revelation of related information and cheating. Furthermore, cheating behaviors include two types: make the related information larger or smaller.

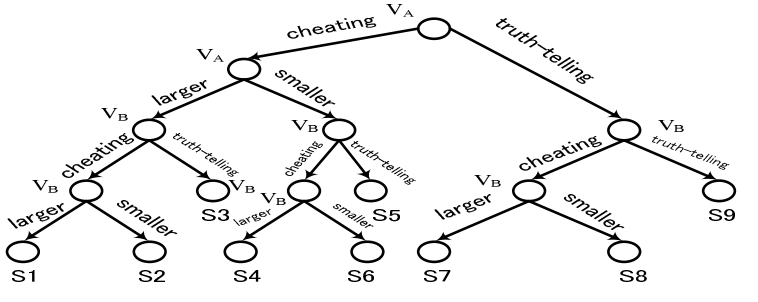


Fig. 1. Mutually dependent link cost decision illustrated in extensive game form

The detail analysis process to prove the truth-telling property in challenge of mutually dependent link cost is given as follows: if based on the truthful link cost, link AB is on the lowest cost path for source to destination, then,

- Scenario S1 (Both node A and B make the cost higher) can not appear, for after this joint cheating action, if peer A and peer B still in the lowest cost path, then the payment that peer A obtains, $SP^{-i} - SP \setminus (A,B)$, is unchanged; peer B's payment: $SP^{-i} - SP \setminus (B,B+1)$ is smaller, for SP becomes larger, where (B+1) denotes the successor of node B in shortest-cost path; if this joint cheating make the link AB leaves the lowest cost path, then node A and B can not get remuneration. For the similar reasons, Scenario S3, S7 can not appear;
- Scenario S5, S6 and S8 can not appear, for the simple reason that those cheating actions make node B can not receive the packet forwarded by node A. then, in this situation node A and B can not get any remuneration;
- In S2, S4 and S9, we can find that the best choice of each peer (A or B) is Tit for Tat. In our case, Tit for Tat means if node B believes that node A lie about the private information (making larger or smaller), then peer B also lies about the private information (make it smaller or larger correspondingly). If node A tells truth, then node B also tells truth. The same principle is also suitable for node A, which is so-called mutual-dependent strategy. So, we need investigate deeply whether the node A and B have the incentive to lie. Assume that, after the joint cheating actions, link AB still on the shortest path (otherwise, the joint cheating action is meaningless), Furthermore, we assume that if only one node cheats, then utility of A (or B) is zero respectively, for we already illustrated unilateral cheating brings no extra benefit for the cheating node. So, the payoff matrix of node A and B is given in Table 1.

Table 1. Payoff matrix of node A and B (in scenarios of S2, S4 and S9)

Node B Node A	Truthful	Defect (Smaller or larger)
Truthful	$(SP^{-i} - SP \setminus (A,B) - c)$, $(SP^{-i} - SP \setminus (B,B+1) - c)$	0,0
Defect (larger or smaller)	0,0	$(SP^{-i} - SP \setminus (A,B) - c)$, $(SP^{-i} - SP \setminus (B,B+1) - \varepsilon - c)$

Note, the best result of joint actions is to obtain the truth cost of link AB. Unfortunately, in most situations, the joint cheating actions make the cost of LCP is $SP+\varepsilon$, higher than the shortest cost *SP*. So, under joint cheating actions, node B's utility is: $u_B^{cheating} = (SP^{-i} - SP \setminus (B,B+1) - \varepsilon - c)$. And without losing generality, we assume truthful forwarding cost of node A and B is *c*.

Table 1 shows there are two equilibriums in this game. For the following reasons, we can infer that the strategy of (truthful, truthful) is the risk dominant strategy of participators (node A and B in our illustration).

- From the viewpoint of system designer, recall that the goal of designing mechanism is to select the shortest-cost forward path, this joint cheating action doesn't affect the selection outcome at all (for the best result of joint actions is to obtain the truth cost of link AB). Furthermore, if this joint cheating action still makes link AB in the shortest-cost path, then, with high probability, the total payment by the source (or system designer) decrease.
- The joint cheating actions need collusion of neighboring peers, but truthful revelation can be unilaterally determined by each participant to enforce. So, based on the assumption of rational nodes in Ad hoc network, truthful revelation is the risk dominant strategy for each node.
- Note, that even though, some nodes unintentionally offer the false feedback about the related information (with small probability), the Tit for Tat strategy will drive nodes to the dominant equilibrium strategy, truth-telling, which have been deeply discussed in [8][9].
- We also prove that, based on the uniformly probability of peer behaviors, the expected utility for peer to truthfully reveal private information is larger [18].

In brief, for the mutually dependent link cost in ad hoc network, truthful revelation of private information is the risk dominant strategy of each participant (that is, node A and B in Fig. 1).

If, based on the truthful link cost, link AB is not on the lowest cost path for source to destination, then, the unilateral cheating action either has not effect on routing selection (make the link cost higher), or make the node B can not receive packet which lead to obtaining no remuneration; just as described above, the best result of joint cheating actions is to make the truthful link cost emerge, which have also not effect on routing selection.

Then based on backward induction principle in extensive game, we can infer that truth-telling is the risk dominant choice of each participant.

Our result is interesting because the cryptographic approach taken in [12] represents considerable engineering effort to align the incentives. In fact, it is not necessary and worthwhile to spend the considerable cryptography-based effort at aligning the incentives in ad hoc routing game. Generally, the cost of cooperation inducing mechanism must be low and commensurate with its benefits. So, we think the cost to do so would vastly outweighs the benefit the protocol it enabled.

4 Hidden Actions in Message Forwarding

Mechanism in the above section for the ad hoc routing game assumes that once the transit cost have been obtained, and the shortest cost path has been determined, the nodes on the shortest cost path obediently forward all packets, and that there is no lost in the network, i.e, k=0, k is the per-hop loss rate. In fact, the wireless link faces relatively high loss rate. So, intuitively, each intermediate node should get higher remuneration to bear the threat of loss link than in lossless environment. Furthermore, actually, nodes' actions may be unobservable by entities that buy the routing and forwarding service from Ad hoc network, so-called principal. Principal may be the

source node, destination node or the authorized third party. In this section, we investigate the equilibrium properties of forwarding game in ad hoc network, under the condition of per-hop monitoring and no per-hop monitoring.

The simplified model is given as follows: multiple agents perform sequential hidden action, that is, each agent can choose to drop or forward packet; Principal can observe: the final outcome only (without per-hop monitoring); the per-hop outcome (with per-hop monitoring). Action $a_i \in \{0,1\}$, $a_i = 0$ means that node i drops packet, otherwise, $a_i = 1$; Outcomes: $R(a) = r \in \{r^G, r^B\}$, r^G means that packet reaches destination; r^B means that packet doesn't reaches destination; Utility of node i: $u_i(m_i, c_i, a_i) = m_i - a_i c_i$. Note that, in order to persuade node i to face the threat of loss link, obviously, m_i should large than p_i obtained in equation (4).

Given a per-hop loss rate of k, we can express the probability that a packet is successfully delivered from node i to its successor i+1 as: $Pr(r^G_{i\to i+1} | a_i) = (1-k)a_i$, where $r^G_{i\to i+1}$ denotes a successful transmission from node i to j.

Ref. [12] uses the scenario of no per-hop monitoring scheme to illustrate the Theorem 1 proposed in their paper, "There does not exist a forwarding-dominant protocol for ad hoc games". But, their proof procedure violates the basic constraint, IR, that is, in their situation, no node will join the game (Refer to [18] for detailed description).

Base on different solution concepts, this paper attempts to characterize the economic meanings of forwarding game with per-hop monitoring and without per-hop monitoring.

Theorem 1. In ad hoc forwarding game, let the payment scheme be:

$$0 \le m_i^B < \min_{j \in SP}\{p_j - c_j\};\ m_i^G = \frac{p_i}{(1-k)^{n-i+1}}$$

where m_i^G and m_i^B denote the payment to node i under the outcome of r^G and r^B. Then, forwarding game without per-hop monitoring can achieve Nash equilibrium.

Let the payment scheme be

$$0 \le m_i^B < \min_{j \in SP}\{p_j - c_j\};\ m_i^G = \frac{p_i}{(1-k)}$$

That is, payment to node i is m_i^G, if the packet has reached node i+1, and m_i^B otherwise. Then, per-hop monitoring mechanism provides the dominant strategy equilibrium.

Proof: ① **Nash equilibrium in no per-hop monitoring**

The IR and IC in Nash equilibrium can be formally described as follows:

IR:

$$\Pr(r^G_{S\to i} | a_{j<i} = 1)\left(\Pr(r^G | a_{j\ge i} = 1)m_i^G + \Pr(r^B | a_{j\ge i} = 1)m_i^B - c_i\right) + \Pr(r^B_{S\to i} | a_{j<i} = 1)m_i^B \ge 0$$

This constraint says that the expected utility from participating is greater than or equal to zero (reservation utility), if all other nodes forward.

IC:

$$Pr(r^G_{S\to i}\left|a_{j<i}=1\right.)\left(Pr(r^G\left|a_{j\ge i}=1\right.)m_i^G+Pr(r^B\left|a_{j\ge i}=1\right.)m_i^B-c_i\right)+Pr(r^B_{S\to i}\left|a_{j<i}=1\right.)m_i^B$$
$$\ge Pr(r^G_{S\to i}\left|a_{j<i}=1\right.)\left(Pr(r^G\left|a_i=0,a_{j>i}=1\right.)m_i^G+Pr(r^B\left|a_i=0,a_{j>i}=1\right.)m_i^B\right)+Pr(r^B_{S\to i}\left|a_{j<i}=1\right.)m_i^B$$

The constraint says that the expected utility from forwarding is greater than or equals to its expected utility from dropping, if all nodes forward as well.

Based on the loss link model, the above general equations can be represented as:

IC:

$$(1-k)^i\left((1-k)^{n-i+1}m_i^G+\left(1-(1-k)^{n-i+1}\right)m_i^B-c_i\right)+\left(1-(1-k)^i\right)m_i^B\ge 0$$

IR:

$$(1-k)^i\left((1-k)^{n-i+1}m_i^G+\left(1-(1-k)^{n-i+1}\right)m_i^B-c_i\right)+\left(1-(1-k)^i\right)m_i^B\ge$$
$$(1-k)^i m_i^B+\left(1-(1-k)^i\right)m_i^B$$

Base on the proposed payment scheme, it is easy to verify that IR and IC conditions can be satisfied, which implies that there exist Nash equilibrium.

② **Dominant strategy equilibrium in per-hop monitoring**

In this approach, the principal make the payment schedule contingent on whether the packet has reached the next hop or not. Under monitoring mechanism, the principal has to satisfy the following constraints:

IR: $$Pr(r^G_{S\to i}\left|a_{j<i}=1\right.)\left(Pr(r^G_{i\to i+1}\left|a_i=1\right.)m_i^G+Pr(r^B_{i\to i+1}\left|a_i=1\right.)m_i^B-c_i\right)\ge 0$$

IC:

$$Pr(r^G_{S\to i}\left|a_{j<i}=1\right.)\left(Pr(r^G_{i\to i+1}\left|a_i=1\right.)m_i^G+Pr(r^B_{i\to i+1}\left|a_i=1\right.)m_i^B-c_i\right)\ge$$
$$Pr(r^G_{S\to i}\left|a_{j<i}=1\right.)\left(Pr(r^G_{i\to i+1}\left|a_i=0\right.)m_i^G+Pr(r^B_{i\to i+1}\left|a_i=0\right.)m_i^B\right)$$

The above two conditions can also be represented as:

IR: $$(1-k)^i\left((1-k)m_i^G+km_i^B-c_i\right)\ge 0$$

IC: $$(1-k)^i\left((1-k)m_i^G+km_i^B-c_i\right)\ge(1-k)^i m_i^B$$

So, obviously, IR and IC conditions can be satisfied.

It is also easy to verify that the expected payment is identical in those two approaches, per-hop monitoring and no per hop monitoring. But the fundamental difference between two scenarios is the solution concepts used. If no per-hop monitoring is used, the strategy profile is Nash equilibrium, which means that no agent has an incentive to deviate unilaterally from the strategy profile. In contrast, per-hop monitoring implies that the best action chosen by each node is always to forward packet, irrespective of other nodes' behaviors. Therefore, per-hop monitoring provides us with dominant strategy equilibrium, so an argument for designing mechanism with dominant strategies is that it makes the game much easier for the agent to play (that is, participants need not worry about doing any research into how the agents are likely to play). Ref. [7] designs payment schemes, Sprite, to prevent several types of selfish behaviors, in which receipt is used as per-hop monitoring mechanism. But unfortunately, the procedure has weak points. That is, decision that peer *i* whether forward packet will depend on not only the actual forwarding behavior of peer *i,* but the behavior of the next-hop peer *i*+1 to report receipt, which is another kind of mutual decision scenarios. For this

receipt-based per-hop monitoring, we also investigate peers' incentive to provide receipt [18], for limited space in this paper, we omit this part.

5 Conclusion and Future Works

Recently, cross-disciplinary efforts involving economics and computer network have proliferated. One main reason is that open network architecture already made computing infrastructure possess characteristics of an economy-society entity as well as those of computer system. Specifically, wireless ad-hoc networks are often formed by rational and autonomous nodes determining their behaviors independently. Intuitively, those nodes do not cooperate unless they have incentive to do so. Thus, both routing and packet forwarding become games. This paper argues that economic-based mechanisms play both a sanctioning and a signaling role, that is, the role of economic-inspired mechanisms in pure information asymmetry (in routing phase) is to induce learning (signaling role) whereas the role of such mechanisms in pure moral hazard (forwarding phase) settings is to constrain behavior (sanctioning role). In this paper, based on economic models, we re-examine several proposals in related paper, and firstly demonstrate that, based on analysis of extensive game, the VCG-like routing mechanism is risk dominant strategy in the mutually dependent link cost environment, which means that the link cost is determined by two neighboring nodes together. Then, we formally describe the Individual Rationality (IR) and Incentive Compatibility (IC) constraints which should be satisfied by any routing and forwarding game in Ad hoc network, and investigate the economic implications of per-hop monitoring scheme and no per-hop monitoring scheme using different solution concepts. That is, with proper payment structure, no per-hop monitoring scheme provides Nash equilibrium, and per-hop monitoring scheme provides dominant strategy equilibrium.

As the most famous general technique for designing truthful mechanisms, there exist four major problems in VCG mechanism, First, the traditional VCG mechanism can be used only if we want to maximize the overall social welfare, e.g., minimize the cost to the agents; Second, the VCG mechanism may be computationally intractable because the underlying optimization problem is NP-hard. Third, the amount the mechanism must pay to the agents can be significantly higher. Fourth, VCG mechanism is significantly vulnerable to coalitions of agents. In this paper, we preliminarily investigate some problems in VCG-like routing and forwarding mechanism in Ad hoc network. In future work, we should further investigate how to address or alleviate the above problems, when VCG-like mechanisms are properly adapted to solve problems in open networks.

References

[1] P. Obreiter and J. Nimis, A taxonomy of incentive patterns - the design space of incentives for cooperation, In Proc. of the Second International Workshop on Agents and Peer-to-Peer Computing (AP2PC'03), LNCS 2872, Melbourne, Australia, 2003.

[2] P. Michiardi and R. Molva, Core: a COllaborative REputation mechanism to enforce node cooperation in mobile ad hoc networks, In Proc. of the IFIP - Communications and Multimedia Security Conference, 2002.

[3] S. Buchegger and J.Y.L. Boudec, Performance analysis of the CONFIDANT protocol: cooperation of nodes-fairness in distributed Ad-hoc networks, In proc. of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC), 2002.
[4] W. Yu and K. J. R. Liu, Attack-resistant cooperation stimulation in autonomous Ad Hoc networks, IEEE Journal on Selected Areas in Communications: Autonomic Communication Systems, 2005.
[5] J. Hubaux, T. Gross, J. Boudec and M. Vetterli, Toward self-organized mobile Ad hoc networks: the Terminodes project, IEEE Communication Magazine, Jan. 2001.
[6] L. Buttyan and J. Hubaux, Stimulating cooperation in self-organizing mobile ad hoc network, ACM/Kluwer Mobile Networks and Applications (MONET), Vol. 8, 2003.
[7] Sheng Zhong, Jiang Chen and Yang Richard Yang, Sprite: A simple, cheat-proof, credit-based system for mobile Ad-Hoc Networks, In Proc. of IEEE INFOCOM 2003.
[8] V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao, Cooperation in wireless ad hoc networks, in Proc. of IEEE INFOCOM, 2003.
[9] M´ark F´elegyh´azi, Jean-Pierre Hubaux and Levente Butty´an, Nash equilibria of packet Forwarding Strategies in Wireless Ad Hoc Networks, To appear in IEEE TRANSACTIONS ON MOBILE COMPUTING, 2006.
[10] L. Anderegg, S. Eidenbenz, Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile Ad hoc networks with selfish agents, In Proc. of ACM MobiCom'03, pp. 245-259.
[11] S.Eidenbenz, G.Resta P.Santi, COMMIT: A Sender-Centric Truthful and Energy-efficient routing protocol for Ad Hoc networks with selfish nodes, in Proc. IEEE Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN), 2005.
[12] Sheng Zhong, Li (Erran) Li, Yanbin Grace Liu and Yang Richard Yang, On Designing Incentive-Compatible Routing and Forwarding Protocols in Wireless Ad-Hoc Networks— An Integrated Approach Using Game Theoretical and Cryptographic Techniques, In Proc. of MobiCom, 2005.
[13] Steven J. Bauer, Peyman Faratin and Robert Beverly, Assessing the assumptions underlying mechanism design for the Internet, In Proc. of the First Workshop on the Economics of Networked Systems (NetEcon), 2006.
[14] M. Afergan, Using repeated games to design incentive-based routing systems, In Proc. of IEEE INFOCOM, 2006.
[15] N. Nisan and A. Ronen, Algorithmic mechanism design, Games and Economic Behavior, Vol. 11, No. 2, 2001.
[16] WANG Yu-feng, WANG Wen-dong, YUAN gang and CHENG Shi-duan, Study on cooperation incentive mechanism based on Vickrey auction in Ad hoc networks, Journal of Beijing University of Posts and Telecommunications (BUPT), Vol. 28, No. 4, 2005.
[17] M. Feldman, J. Chuang, I. Stoica, and S. Shenker, Hidden-action in multi-hop routing, In Proc. of the 6th ACM Conference on Electronic Commerce (EC), 2005.
[18] Yufeng Wang, Yoshiaki Hori and Kouichi SAKURAI, On incentive-compatible mechanisms in open networks, Technical Report, Kyushu University, 2006.
[19] Michiardi Pietro, A survey of computational economics applied to computer networks, Research Report - Institut Eurecom, August 2006 RR-06-173.

Enhancing Simulation for Checking Language Containment*

Jin Yi[1,2] and Wenhui Zhang[1]

[1] Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] Graduate University of the Chinese Academy of Sciences, Beijing, China
{yijin,zwh}@ios.ac.cn

Abstract. Many verification approaches based on automata theory are related to the language containment problem, which is PSPACE-complete for nondeterministic automata. To avoid such a complexity, one may use simulation as an approximation to language containment, since simulation implies language containment and computing simulation is a polynomial time problem. As it is an approximation, there exists a gap between simulation and language containment, therefore there has been an effort to develop methods to narrow the gap while keeping the computation in polynomial time. In this paper, we present such an approach by building a Büchi automaton based on partial marked subset construction to be used in the computation of simulation relation, such that the automaton preserves the original language and has a structure that helps identify more pairs of automata that are in language containment relation. This approach is an improvement to the fair-k-simulation method [3].

1 Introduction

Many verification approaches based on automata theory are related to the language containment problem, which is PSPACE-complete for nondeterministic automata [2]. Although we can compute this relationship by transforming the problem $L(A_1) \subseteq L(A_2)$ into checking the emptiness of $L(A_1) \cap L(\overline{A_2})$, it is not easy to implement the complementation process [12,18,13,8] efficiently. Simulation relation is a pre-order relation of states, which captures branching behavior containment. Computing simulation is sufficient for checking trace behavior containment (language containment). From the practical perspective, checking language containment by computing simulation has its advantages, since computing simulation is of polynomial complexity.

There are a few simulation notions for automata which can be used to identify language containment. In [15,2], direct simulation has been investigated. Direct

* Supported by the National Natural Science Foundation of China under Grant No. 60573012 and 60421001, and the National Grand Fundamental Research 973 Program of China under Grant No. 2002cb312200.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 374–385, 2007.
© Springer-Verlag Berlin Heidelberg 2007

simulation requires that every accepting state must be simulated by an accepting state in addition to the standard simulation definition. In [9,12], the authors proposed a new view of simulation for finite-state systems with weak or strong fairness and gave an algorithm for computing fair simulation by tree automata emptiness testing. In [5], the authors did research on simulation using a game-theoretic framework, and developed an efficient algorithm to compute fair and delayed simulation by parity game [10] in $O(mn^3)$ where n and m are the number of states and transitions,respectively. Among other related work, [1] compare the above notions of simulation, and [6] computes the simulation relation for alternating Büchi automata.

The methods for computing language containment of automata based on the above notions are incomplete. (i.e., Two states which are not connected via simulation can nevertheless be in language-containment relation.) There has been some research to narrow the gap between simulation and language-containment. [3] proposes k-simulation which allows the duplicator to use k pebbles instead of 1 (the normal definition) in response to the spoiler's move of a single pebble. We can regard this process as performing partial improved subset construction, i.e., restrict the size of subset by a given number k. Note that when k increases to n (the number of states), k-simulation of Büchi automata is still not equivalent to language containment. [11] consider trace inclusion (language containment) between two finite-state fair discrete systems (two nondeterministic Street automata), and showed that this problem can be solved by fair simulation if an appropriate *non-constraining* automaton is provided.

In our paper, we construct a Büchi automaton based on partial marked subset construction. This automaton preserves the original language and has a structure that helps to identify more pairs of the original automaton with language containment relation. We prove that this approach is an improvement to the fair-k-simulation method [3]. The organization of this paper is as follows: In Section 2, we present the background knowledge. In Section 3, we describe our approach with three steps, and we make a comparison between our approach and the fair-k-simulation method. In Section 4, we give conclusion remarks.

2 Preliminaries

Definition 1 (Büchi automata). *A Büchi automaton is a tuple* $A = \langle Q, \Sigma, q_0, \Delta, F\rangle$ *where* Q *is a finite set of states,* Σ *is the input alphabet,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of accepting states,* $\Delta \subseteq Q \times \Sigma \times Q$ *is the transition relation.*

Let $\delta : Q \times \Sigma \to 2^Q$ denote the transition function defined by: $q \in \delta(q', a)$ if $(q, a, q') \in \Delta$. A run of A on an infinite word $\alpha = \alpha(0)\alpha(1)\cdots$ is a sequence $r = r(0)r(1)\cdots$ such that $r(0) = q_0$, and for every $i \geq 0$, $r(i+1) \in \delta(r(i), \alpha(i))$. Let $inf(r)$ be the set of states that r visits infinitely often. If $inf(r) \cap F \neq \emptyset$, then the run r is *an accepting run* and α is in the language of A. The language of A is denoted by $L(A)$.

For convenience, we write $q \in A$ for q being a reachable state of A. We write Q^A, Δ^A, F^A for respectively the set of states of A, the set of transition relation of A, and the set of accepting states of A.

The usual subset construction [17] builds a new automaton with exponential number of states, each state is the subset of states of the original automaton. The states in the same subset can be reached on the same string. Breakpoint construction [14] is a simple variant of usual subset construction used to transform an alternating Büchi automaton to a nondeterministic Büchi automaton. [18] regards this method as an improved subset construction. While doing improved subset construction, we mark every accepting state and every state that has a marked processor. After reading the prefix of some infinite word, if all the states that can be reached from the initial state are marked, we identify a breakpoint and clear the markings from these states. In the following, we formally describe this method and call it *marked* subset construction as in [16]. Let $\delta_{sub}(X, a)$ denote the set $\cup_{x \in X} \delta(x, a)$.

Definition 2 (marked subset construction). *Let $A = \langle Q, \Sigma, q_0, \Delta, F\rangle$ be a Büchi automaton. Then $sub(A) = \langle Q', \Sigma, q'_0, \Delta', F'\rangle$ is the marked subset automaton of A with*

- $Q' = \{(X, f) \mid X \subseteq Q, f \subseteq X\}$
- $q'_0 = (\{q_0\}, \{q_0\})$
- $((X, f), a, (Y, g)) \in \Delta'$ *if*
  - $(X, f), (Y, g) \in Q'$, $Y = \delta_{sub}(X, a)$
  - *Let* $X = \{x_1, x_2 \cdots x_n\}$, $Y = \{y_1, y_2 \cdots y_m\}$
    $y_i \in g$ *if it satisfies one of following conditions:*
    1. $y_i \in F$
    2. $X \neq f$ *and* $\exists x_j \in f$ *such that* $(x_j, a, y_i) \in \Delta$
- $F' = \{(X, f) \in Q' \mid X = f\}$

Each state $q = (X, f) \in sub(A)$ is a pair of subsets X and f. f records the marked states. If $s \in X$ is in f, then s is marked in q, otherwise, s is unmarked. For convenience, we use $q.X$ to denote the set X of q.

Let A be a Büchi automaton and $w \in \Sigma^*$. We use $s_1 \xrightarrow{A,w} s_2$ to represent that there is a path in A from s_1 to s_2 consistent with the input w. The symbol A in $\xrightarrow{A,w}$ is usually omitted, if it is clear from the context.

We define simulation game-theoretically [5]. The simulation game on A and A' is played by two players: Spoiler and Duplicator in rounds. At round 0, a red and a blue pebble are placed on q_0 and q'_0, respectively. In each round, Spoiler may choose any transition (q_i, a, q_{i+1}) and move the red pebble to the position q_{i+1}. Then Duplicator must have a corresponding transition (q'_i, a, q'_{i+1}) such that he can move the blue pebble to q'_{i+1}. If Duplicator does not have such a transition, game halts and the spoiler wins. Otherwise the game produces two infinite runs: $\pi = q_0 a_0 q_1 a_1 q_2 \cdots$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \cdots$. The pair (π, π') is called an outcome of the game. The duplicator wins the game according to the following rules:

1. Direct simulation game $G^{fair}_{A,A'}(q_0, q'_0)$: the outcome (π, π') is winning for Duplicator iff, for all i, if $q_i \in F^A$, then $q'_i \in F^{A'}$
2. Fair simulation game $G^{direct}_{A,A'}(q_0, q'_0)$: the outcome (π, π') is winning for Duplicator iff, if there are infinitely many j such that $q_j \in F^{A'}$ or there are only finite many i such that $q_i \in F^A$

A strategy for Duplicator is a partial function $f : Q(Q'\Sigma Q)^* \to Q'$. It determines the next move of Duplicator according to the history of the play. That is, $f(q_0) = q'_0$ and $q'_j = f(q_0 q'_0 a_0 q_1 q'_1 a_1 \cdots a_{j-1} q_j)$ where $(q_i, a_i, q_{i+1}) \in \Delta^A$ and $(q'_i, a_i, q'_{i+1}) \in \Delta^{A'}$ for $i < j$. A strategy for Duplicator is a winning strategy if whenever $\pi = q_0 a_0 q_1 \cdots$ is a run of A and $\pi' = q'_0 a_0 q'_1 \cdots$ is a run defined by $q'_{i+1} = f(q_0 q'_0 a_0 q_1 q'_1 \cdots a_i q_{i+1})$, then (π, π') is winning for Duplicator.

Definition 3. *Let A and A' be Büchi automata, state $q' \in A'$ directly or fairly simulates a state $q \in A$, denoted by $q \leq^* q'$, if there is a winning strategy for the duplicator in direct or fair simulation game $G^*_{A,A'}(q, q')$, where $* = direct, fair$.*

If $A = \langle Q, \Sigma, q_0, \Delta, F\rangle$ is an automaton and $q \in Q$, then $A[q]$ denotes the modified automaton $\langle Q, \Sigma, q, \Delta, F\rangle$.

Proposition 1. *[5] Let A and A' be Büchi automata,*

1. *$\leq^*$ is a reflexive, transitive relation;*
2. *$\leq^{direct} \subseteq \leq^{fair}$;*
3. *if $q \leq^* q'$, then $L(A[q]) \subseteq L(A'[q'])$.*

[5] propose an efficient algorithm to compute fair simulation game, which is based on the lifting algorithm of [10]. The following proposition shows the complexity of this algorithm.

Proposition 2. *The lifting algorithm of [5] computes the fair simulation game in time $O(m'(n_1 + 1))$ and space $O(m')$, where m' is the number of edges in $G^{fair}_{A,A'}(q_0, q'_0)$ and $n_1 < |A||A'|$.*

Given a Büchi automaton A (let $A = A'$), the k-simulation game [3] is defined for a state $q_0 \in Q$ and a k-vector $\mathbf{q}'_0 = (q'_{0,1}, \cdots, q'_{0,k}) \in Q^k$. In each round, when the spoiler chooses a transition $(q_i, a, q_{i+1}) \in \Delta$ and the red pebble is placed in q_{i+1}, the duplicator must choose a transition that belongs to his transition relation Γ_A and move k pebbles. Given $\mathbf{q}_0 = (q_{0,1}, \cdots, q_{0,k})$ and $\mathbf{q}'_0 = (q'_{0,1}, \cdots, q'_{0,k})$, $(\mathbf{q}_0, a, \mathbf{q}'_0) \in \Gamma_A$ if for each $i \in [k]$ there is some $j \in [k]$ such that $(q_{0,j}, a, q'_{0,i}) \in \Delta$. Note that i and j need not be the same. We say that $\mathbf{q}'_m$ is *good* since round $m' \leq m$ if at round m, for all $q_{m,i} \in \mathbf{q}'_m$, there exists a path from $q'_{m',j}$ to $q'_{m,i}$ in A, which goes through an accepting state and m is the first round with this property. The meaning of a vector of states $\mathbf{q}'_m$ being good since some prior round, is analogous to a single state q being an accepting state. For fair-k-simulation game, let $\pi = q_0 a_0 q_1 a_1 q_2 \cdots$ and $\pi' = \mathbf{q}'_0 a_0 \mathbf{q}'_1 a_1 \mathbf{q}'_2 \cdots$, the duplicator wins if the following is true: if there are infinitely many i such that $q_i \in F$, then for each such i, there exists $j \geq i$ such that $\mathbf{q}'_j$ is *good* since round i.

In the rest of this paper, $A \leq^* B$ also means $q_0 \leq^* q'_0$, where q_0, q'_0 is the initial state of A, B, respectively.

3 Search Pairs of States for Language Containment

To understand the motivation of this work, we consider two Büchi automata A (left) and B (right) in Fig.1. It is obvious that $L(B) \subseteq L(A)$. However, we cannot prove this language containment relation by the fair-k-simulation method.

Consider playing the fair-k-simulation game between A and B. When the spoiler put the red pebble in q_1, the duplicator can put two blue pebbles in s_1 and s_2 respectively, for s_0 can reach s_1 and s_2 by two transitions with the same label "a". The transitions of the vector (s_1, s_2) can match all transitions of q_1, therefore, the duplicator can simulate the spoiler in q_1 by the vector (s_1, s_2). However, when the spoiler has an infinite play $\pi = q_0 a q_1 b q_1 \cdots$ on ab^ω, the duplicator has a corresponding play $\pi' = (s_0)a(s_1, s_2)b(s_1, s_2)\cdots$, since q_1 is an accepting state and appears infinitely, but the vector (s_1, s_2) does not become an "accepting" state in any round. By the rule of fair-k-simulation game, the duplicator loses the game. This shows that the fair-k-simulation method is not sufficient for proving $L(B) \subseteq L(A)$.

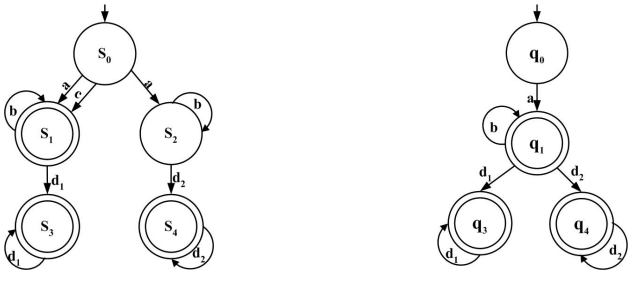


Fig. 1. A Motivating Example

We present an approach to prove $L(B) \subseteq L(A)$ by constructing a new Büchi automaton A^* such that $B \leq^{fair} A^*$. A^* preserves the language of A and has a structure that is more amenable to fair simulation.

The process to build the new Büchi automaton consists of three steps. (1) Build Büchi automaton A^1 whose state comprises an index and a state of A. The states in A^1 reached over the same string contain the same index. (2) Build Büchi automaton A^2 by adding accepting states into A^1. (3) Build Büchi automaton A^* by adding transitions into A^2. This construction of the Büchi automaton does not change the language of A, i.e. $L(A^1) = L(A^2) = L(A^*) = L(A)$. The details are explained in the following subsection.

3.1 Construct New Büchi Automaton

For constructing A^1, we want to know for each given string, the set of states in A that can be reached over that string. For this purpose, we may do subset construction on A. Since the $sub(A)$ has 2^n states ($|A| = n$), from the complexity aspect, we need restrict the number of states in subset (up to a $k \geq 2$), thus we

can have a polynomial number of states. This restriction is a trade-off, because it results in decreasing the opportunities to add transitions into A^2 at step 3.

Based on the idea of the paper [3], we introduce the marked k-subset construction. Let $Set_k(X) = \{Y \mid Y \subseteq X, 1 \leq |Y| \leq k\}$.

Definition 4 (marked k-subset construction). *Let $A = \langle Q, \Sigma, q_0, \Delta, F\rangle$ be a Büchi automaton. The marked k-subset automaton $sub^k(A) = \langle Q^k, \Sigma, q_0^k, \Delta^k, F^k\rangle$ is given as follows.*

- $Q^k = \{(X, f) \mid X \subseteq Q, |X| \leq k, f \subseteq X\}$
- $q_0^k = (\{q_0\}, \{q_0\})$
- $((X, f), a, (Y, g)) \in \Delta^k$ *if*
  - $(X, f), (Y, g) \in Q^k$, $Y \in Set_k(\delta_{sub}(X, a))$
  - *Let* $X = \{x_1, x_2 \cdots x_n\}$, $Y = \{y_1, y_2 \cdots y_m\}$, $m, n \leq k$ $y_i \in g$ *if it satisfies one of following conditions:*
    1. $y_i \in F$
    2. $X \neq f$ *and* $\exists x_j \in f$ *such that* $(x_j, a, y_i) \in \Delta$
- $F^k = \{(X, f) \in Q^k \mid X = f\}$

Note that while $sub(A)$ a deterministic automaton, $sub^k(A)$ may be non-deterministic. The following properties of $sub^k(A)$ are similar to those of $sub(A)$.

Proposition 3. *Given A and $sub^k(A)$, Let $(X, f), (Y, g) \in sub^k(A)$. If $(X, f) \xrightarrow{w} (Y, g)$ in $sub^k(A)$, then for each $y \in Y$, there exists $x \in X$ such that $x \xrightarrow{w} y$ in A.*

Proposition 4. $L(sub^k(A)) \subseteq L(A)$.

An example of marked k-subset construction $sub^2(A)$ is shown in Fig.2, where the state labelled by $\dot{\mathbf{s}}$ means that $s \in A$ is marked and s is a part of the state (an element of the subset of the states of A). Similarly, $\mathbf{s}$ means that s is unmarked and is a part of the state. For convenience, we assign each state of $sub^2(A)$ an identifying number.

In the following, we give the definition of A^1 which is constructed similar to the product $sub^k(A) \times A$ with accepting states according to A.

Definition 5 (A^1). *Let $A = \langle Q, \Sigma, q_0, \Delta, F\rangle$ and $sub^k(A) = \langle Q^k, \Sigma, q_0^k, \Delta^k, F^k\rangle$. $A^1 = \langle Q^1, \Sigma, q_0^1, \Delta^1, F^1\rangle$ is the Büchi automaton with*

- $Q^1 = \{(q, s) \mid q = (X, f) \in Q^k, s \in X \subseteq Q\}$
- $q_0^1 = (q_0^k, q_0)$
- $F^1 = \{(q, s) \in Q^1 \mid s \in F\}$
- $\Delta^1 = \{((q, s), a, (q', s')) \in Q^1 \times \Sigma \times Q^1 \mid (q, a, q') \in \Delta^k \wedge (s, a, s') \in \Delta\}$

We call q the index of (q, s). The index q contains information related to the path from the initial state to (q, s). For the first, if two states $u, u' \in A^1$ have the same index, then one can be reached over a string from the initial state iff the other can be reached over the same string. For example, the corresponding

A^1 of A and $sub^2(A)$ is shown in Fig.3 where the states $(3, s_1)$ and $(3, s_2)$ are both with index 3 (which represents the state $s_1 s_2$ in Fig.2). Then the states $(3, s_1)$ and $(3, s_2)$ can be reached over the same strings ab^*. Note that in Fig.3, A^1 has been simplified by deleting the state $(6, s_2)$(it is redundant according to the state $(3, s_2)$). For the second, the index contains information on accepting states that the path has passed. Since whether (q, s) belongs to F^{A^1} is decided by $s \in F^A$, thus whether the path from some accepting state of A^1 to (q, s) has passed an accepting state can be checked by whether s is marked or unmarked in q. This mark information decides whether we can add transitions into A^1 (at step 3). Therefore we choose marked subset construction, not the usual subset construction for this purpose.

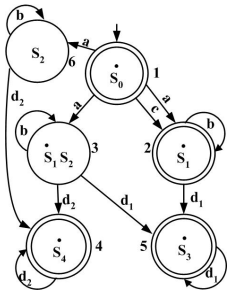


Fig. 2. $sub^2(A)$

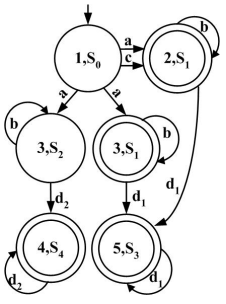


Fig. 3. A^1

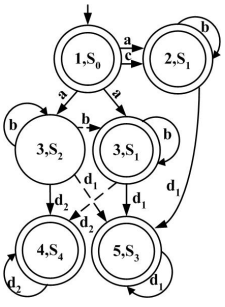


Fig. 4. A^*

Theorem 1. $L(A^1) = L(A)$.

Proof. We first prove $L(A) \subseteq L(A^1)$. Let $\alpha = \alpha_0\alpha_1 \cdots \in L(A)$ and $r = s_0 s_1 s_2 \cdots$ be an accepting run of A on α. By the definition of marked k-subset construction, $\exists r^{sub} = q_0 q_1 q_2 \cdots$ on α in $sub^k(A)$. Based on r and r^{sub}, we can construct a path $r^1 = (q_0, s_0)(q_1, s_1)(q_2, s_2) \cdots$ which is a run of A^1 on α, according to the construction of A^1. Since there is some $s_i \in inf(r) \cap F^A$, then there is some $(q_i, s_i) \in inf(r^1) \cap F^{A^1}$. Therefore r^1 is an accepting run and $\alpha \in L(A^1)$.

We then prove $L(A^1) \subseteq L(A)$. Let $\alpha = \alpha_0\alpha_1 \cdots \in L(A^1)$ and $r^1 = (q_0, s_0)$ $(q_1, s_1)(q_2, s_2) \cdots$ be an accepting run of A^1 on α. We can make a projection of r^1 onto A, and obtain a path $r = s_0 s_1 s_2 \cdots$ of A on α. Since there exists some $(q_i, s_i) \in inf(r^1) \cap F^{A^1}$, there also exists $s_i \in inf(r) \cap F^A$ according to the definition of F^{A^1}. Therefore r is an accepting run of A and $\alpha \in L(A)$. □

Then we define A^2 which is the same as A^1 except that the set of accepting states is enlarged.

Definition 6 (A^2). *Given $A^1 = \langle Q^1, \Sigma, q_0^1, \Delta^1, F^1 \rangle$ and $sub^k(A)$. Then $A^2 = \langle Q^1, \Sigma, q_0^1, \Delta^1, F^2 \rangle$ is a Büchi automaton where $F^2 = F^1 \cup \{(q, s) \mid q \in F^{sub^k(A)}\}$.*

Theorem 2. $L(A^2) = L(A^1)$.

Proof. $L(A^1) \subseteq L(A^2)$ is obvious by the definition of A^2.

Now we prove $L(A^2) \subseteq L(A^1)$. Let $r = (q_0, s_0)(q_1, s_1) \cdots$ be an accepting run of A^2 on α. If $inf(r) \cap F^1 \neq \emptyset$, then $\alpha \in L(A^1)$. Otherwise, Let $(q_i, s_i) \in inf(r)$ and $(q_i, s_i) \in F^2 \backslash F^1$, since r is also a run of A^1 and a run of A^1 can be projected into a run of A and a run of $sub^k(A)$, there exists a run $r' = q_0 q_1 \cdots$ of $sub^k(A)$ on α. For $(q_i, s_i) \in F^2 \backslash F^1$, by the definition of F^2, we know $q_i \in F^{sub^k(A)}$, therefore r' is an accepting run of $sub^k(A)$ and $\alpha \in L(sub^k(A)) \subseteq L(A) = L(A^1)$. □

The fact that two states $u_1, u_2 \in A^2$ with the same index can be reached over exactly the same set of strings, implies that if u_1 has some transitions that u_2 does not have, then adding corresponding transitions into u_2 does not affect the words of A^2 (including words do not satisfy the acceptance criteria). Such added transitions help the duplicator having more choices to correspond to the move of the spoiler, and increase the possibility to win the simulation game.

However, adding such transitions may change the language of A^2. So we need to define a kind of transition that can be added into A^2 safely. Since each accepting run of Büchi automata contains an SCC (strongly connected component of the graph), which has at least one accepting state. Therefore, we need to check whether adding transition forms an SCC in A^2. Moreover according to the definition of A^1, we do not need to checking the SCC in A^2 directly, but the corresponding SCC of $sub^k(A)$.

Now we give the conditions to add one transition and analysis that each of them guarantees the language of A^2 unchanged.

Let $u_1 = (q, s_1)$, $u_2 = (q, s_2)$, $u_3 = (q', s_1') \in A^2$. Given $(u_1, a, u_3) \in \Delta^2$ and $(u_2, a, u_3) \notin \Delta^2$. The tuple (u_2, a, u_3) is called **admissible-transition** if one of the following conditions holds:

1. (q, a, q') is not a part of any SCC of $sub^k(A)$.
2. every SCC of $sub^k(A)$ of which (q, a, q') is a part satisfies one of the following conditions:
   (a) the SCC has accepting states.
   (b) the SCC has no accepting states and s_2 is unmarked in q.

For convenience, we use scc_{A^2} to denote an SCC of A^2 formed by adding (u_2, a, u_3) and $scc_{sub(A)}$ to denote the SCC of $sub^k(A)$ which is the projection of scc_{A^2} onto $sub^k(A)$.

For the first, we consider the case when (q, a, q') is not a part of any SCC of $sub^k(A)$. In this case adding the admissible-transition (u_2, a, u_3) into A^2 does not change the language of A^2, because (u_2, a, u_3) cannot be a part of any scc_{A^2}. The reason is as follows. Suppose that (q, a, q') is not a part of any SCC of $sub^k(A)$, then there is no path from q' to q in $sub^k(A)$, thus there is no state $s' \in q'.X$ that can reach a state $s \in q.X$. By the definition of A^1 and A^2, there is no state of A^2 with index q' that can reach a state with index q, Therefore (u_2, a, u_3) is not a part of any scc_{A^2}.

For the second, we consider the case (condition (2a)) where (q, a, q') is a part of some $scc_{sub(A)}$ and the $scc_{sub(A)}$ has an accepting state. In this case, even adding (u_2, a, u_3) forms an scc_{A^2} and this scc_{A^2} has an accepting state, it still

does not change the language of A^2. The reason is as follows. Since $scc_{sub(A)}$ has an accepting state, by the definition of $sub^k(A)$ and A^2, there exists another SCC of A^2 with an accepting state, and the projection of this SCC onto $sub^k(A)$ is $scc_{sub(A)}$. Therefore, adding (u_2, a, u_3) does not change $L(A^2)$.

For the third, condition (2b) is based on the intuition that even adding a transition may form an scc_{A^2}, but if there is no accepting state in such scc_{A^2}, then adding such a transition does not affect $L(A^2)$. We explain this formally by the following lemma. We omit the proof due to space limitation.

Lemma 1. *Let (u_2, a, u_3) be an admissible-transition such that adding this transition forms an scc_{A^2}. Suppose that there is no accepting state in the corresponding $scc_{sub(A)}$. If s_2 is unmarked in q, then there is no accepting state in this scc_{A^2}.*

We define A^* by adding the transition relation of A^2 with the set of admissible-transitions.

Definition 7 (A^*). *Given $A^2 = \langle Q^1, \Sigma, q_0^1, \Delta^1, F^2\rangle$ and $sub^k(A)$. Let Θ be the set of admissible-transitions. A^* is defined to be the Büchi automaton $\langle Q^1, \Sigma, q_0^1, \Delta^1 \cup \Theta, F^2\rangle$.*

Fig.4 shows A^* where there are three admissible-transitions which are denoted by the dashed arrows. $((3, s_1), d_2, (4, s_4))$ and $((3, s_2), d_1, (5, s_3))$ satisfy condition (1), because $(3, d_2, 4)$ and $(3, d_1, 5)$ are not part of any SCC of $sub^2(A)$. $((3, s_2), b, (3, s_1))$ satisfies condition (2b), because s_2 is unmarked in 3. It is easy to see that $B \leq^{fair} A^*$.

Proposition 5. *Let $r = (q_0, s_0)(q_1, s_1)(q_2, s_2)\cdots$ be a path of A^* running on α. Then there exists a corresponding path $r' = q_0 q_1 q_2 \cdots$ of $sub^k(A)$ running on α.*

Note that for all i, if $r'(i) = q_i$, then q_i is the index of $r(i)$.

Theorem 3. $L(A^*) = L(A^2)$.

Proof. It is easy to prove that $L(A^2) \subseteq L(A^*)$. Now we prove $L(A^*) \subseteq L(A^2)$. Let r running on α be an accepting path of A^*. If it does not include any admissible-transition, then $\alpha \in L(A^2)$ is obvious. Otherwise, we consider the following cases:

1. Every admissible-transition of r only appears finitely.
   Let $u_i = (q_i, s_i)$, $u_{i+1} = (q_{i+1}, s_{i+1})$. Suppose (u_i, a, u_{i+1}) is the last admissible-transition appearing in r. We divide r into two parts r_0 and r_1, and α into α_0 and α_1 accordingly.

   Let $r_0 = (q_0, s_0)\cdots(q_i, s_i)(q_{i+1}, s_{i+1})$, r_1 be the rest of r with all the transitions belong to Δ^{A^2}. By Proposition 5, there exists a path $r'_0 = q_0 \cdots q_i q_{i+1}$ on α_0 in $sub^k(A)$.

   By the definition of admissible-transition, there exists $u'_i = (q_i, s'_i)$ such that $(u'_i, a, u_{i+1}) \in \Delta^{A^2}$. Since s'_i and s_i are both in q_i, moreover, q_0 can

reach q_i according to r'_0. By Proposition 3 and the definition of A^1, we can get a path $t = (q_0, s_0) \cdots (q_i, s'_i)$ of A^1, which is also a path of A^2.

Since $(u'_i, a, u_{i+1}) \in \Delta^{A^2}$, then we append state $u_{i+1} = (q_{i+1}, s_{i+1})$ to t and get the path $t' = (q_0, s_0) \cdots (q_i, s'_i)(q_{i+1}, s_{i+1})$ of A^2 on α_0. Since r_1 is the last part of r, it must be an accepting run. It is easy to see that $r'' = t'r_1$ is an accepting path of A^2 on α. Therefore $\alpha \in L(A^2)$.

2. Some admissible-transition of r appears infinitely.
According to the proof of item 1, we know that admissible-transitions of r appearing finitely do not affect $\alpha \in L(A^2)$. So in the following, we only consider admissible-transitions which appear infinitely.

Let $u_i = (q_i, s_i)$, $u_{i+1} = (q_{i+1}, s_{i+1})$, (u_i, a, u_{i+1}) be an admissible-transition. Let r' be the run of $sub^k(A)$ corresponding to r (cf. Proposition 5).

Suppose that (u_i, a, u_{i+1}) appears infinitely in r. Then it must be a part of some scc_{A^2}. Accordingly (q_i, a, q_{i+1}) is in $scc_{sub(A)}$. Therefore, by the definition of admissible-transition, condition (2) (in the definition of admissible-transition) must be satisfied. We first show that condition (2b) cannot be satisfied in this case, and then show that if it satisfies condition (2a), then $\alpha \in L(A^2)$.

- Suppose that the admissible-transition satisfies condition (2b), then s_i is unmarked in q_i and there is no accepting state in $scc_{sub(A)}$. Since $inf(r) \cap F^{A^2} \neq \emptyset$, then exists $(q_j, s_j) \in F^{A^2}$. Because there is no accepting state in $scc_{sub(A)}$, we have $(q_j, s_j) \in F^{A^1}$. By the definition of F^{A^1}, $s_j \in F^A$ and s_j is marked in q_j.

  Let $tr = ((q_j, s_j), a_j, (q_{j+1}, s_{j+1}))$ be a transition in the scc_{A^2}, since s_j is marked, then tr cannot be an admissible-transition, i.e. $tr \in \Delta^{A^2}$.

  Furthermore, since $\Delta^{A^1} = \Delta^{A^2}$, by the definition of Δ^{A^1}, we know that $(s_j, a_j, s_{j+1}) \in \Delta^A$. Because s_j is marked in q_j and $q_{j+1} \notin F^{sub^k(A)}$, then s_{j+1} must be marked in q_{j+1}.

  By the same method, we can prove that all transitions of the path $u_j \cdots u_i u_{i+1}$ cannot be admissible-transitions.

  This contradict with that (u_i, a, u_{i+1}) is an admissible-transition. Therefore condition (2b) cannot be satisfied.
- Suppose that the admissible-transition satisfies condition (2a), i.e., there exists an accepting state in the $scc_{sub(A)}$. Then r' is an accepting run of $sub^k(A)$, and $\alpha \in L(sub^k(A))$. By Proposition 4, Theorem 1 and Theorem 2, we have $L(sub^k(A)) \subseteq L(A) = L(A^1) = L(A^2)$. Therefore $\alpha \in L(A^2)$.

From above analysis, we know that if some admissible-transition of r appears infinitely, then it must satisfy condition (2a), and we have proved that $\alpha \in L(A^2)$ based on this condition. □

Now we analysis the complexity to compute the fair simulation relation between B and A^*. Given a Büchi automaton A with n states and m transitions, there are at most $n(2n)^k$ states and $n(2n)^k m$ transitions in A^*. For simplifying the

problem, we suppose B has the same number of states and transitions as that of A. Therefore, we can compute the game on B and A^* in time $O(n^4(2n)^{2k}m)$ and space $O(n^2(2n)^k m)$ (cf. Proposition 2). Thus, for each fixed k, checking $B \leq^{fair} A^*$ requires polynomial time $n^{O(k)}$.

3.2 Comparison

The following proposition shows that our approach may find more pairs of states with language containment than that of the fair-k-simulation method which can be regarded as doing marked k-subset construction.

Proposition 6. $sub^k(A) \leq^{direct} A^*$.

We omit the proof due to space limitation. Since the condition of direct simulation is stronger than fair simulation (cf. Proposition 1), so by the above proposition, we see $sub^k(A) \leq^{fair} A^*$. Therefore, if we prove $L(B) \subseteq L(A)$ by showing $B \leq^{fair} sub^k(A)$, we can also prove it by showing $B \leq^{fair} A^*$. On the other hand, it is possible that we have $B \leq^{fair} A^*$ but $B \not\leq^{fair} sub^k(A)$ (see the example in Fig.1).

The intuition of our method is to either preserve the original language or add more branches for each state. So we construct A^1 by product A and $sub^k(A)$, then A^1 preserves the original language of A, at the same time, it contains the marked information from $sub^k(A)$, using the information, we can add transitions into A^1 to increase its opportunities to fairly simulate B. So, there may exist some state (q, s) in A^* such that s can directly simulates q and $L(q) \subset L(s)$. For this kind of states, A^* fairly simulates B but $sub^k(A)$ does not. Therefore in a sense our method makes $L(sub^k(A))$ be more close to $L(A)$ than that of the fair-k-simulation method. However, like the fair-k-simulation method, even $k = n$(n is the number of states), our method can not make the fair simulation relation reach language containment.

4 Conclusion

We have presented an approach for computing pairs of states that are in language containment relationship based on fair simulation. The principle of this method is to build a new Büchi automaton A^* without changing the original language. In the construction process, we construct the automaton A^1 based on the marked k-subset construction, each state of A^1 contains an index. Then we add accepting states and transitions into A^1 according to the marking information in $sub^k(A)$. Adding transitions increases the choices for the duplicator in the fair simulation game on A^*, therefore we may find more pairs of states with language containment. In addition, we have compared our approach with fair-k-simulation method and showed the advantage.

Since [11] provide the method to compute fair simulation for generalized Büchi automata, we may also do fair simulation for this kind of automata.

Although our approach enhances the possibility to find pairs of states with language containment relation, our approach is still not complete. The direction of our future work is to continue research on how to efficiently make simulation relation even closer to language containment.

References

1. D. Bustan, O. Grumberg: Applicability of fair simulation. Information and computation, Vol. 194(1), pp. 1-18, 2004.
2. D. L. Dill, A. J. Hu, and H.Wong-Toi. Checking for language inclusion using simulation preorders. CAV'91, vol. 575 of LNCS, pp. 255-265, 1991.
3. K. Etessami. A Hierarchy of Polynomial-Time Computable Simulations for Automata. CONCUR'02, vol. 2421 of LNCS, pp. 131-144, 2002.
4. K. Etessami and G. Holzmann. Optimizing Büchi automata. CONCUR'00, vol. 1877 of LNCS, pp. 153-167, 2000.
5. K. Etessami, Th. Wilke, and R. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM Journal of Computing, 34(5): 1159–1175, 2005.
6. C. Fritz. Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büchi Automata. CIAA'03, vol. 2759 of LNCS, pp. 35-48, 2003.
7. S. Gurumurthy, R. Bloem, F. Somenzi. Fair Simulation Minimization. CAV'02, vol. 2404 of LNCS, pp. 610-624, 2002.
8. S. Gurumurthy, O. Kupferman, F. Somenzi, M. Y. Vardi. On Complementing Nondeterministic Büchi Automata. CHARME'03:96-110.
9. T. A. Henzinger, O. Kupferman, and S. K. Rajamani. Fair simulation. CONCUR'97, vol. 1243 of LNCS, pp. 273-287, 1997.
10. M. Jurdzinski. Small progress measures for solving parity games. STACS'00, vol. 1770 of LNCS, pp. 290-301, 2000.
11. Y. Kesten, N. Piterman and A. Pnueli. Bridging the Gap Between Fair Simulation and Trace Inclusion. Information and Computation, 200(1):35-61, 2005.
12. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. STOC'98:408-429.
13. A. Lindahl. Complementation of Büchi automata: A survey and implementation. master thesis, Linköpings university Department of computer and information science, 2004.
14. S. Miyano and T. Hayashi. Alternating Finite Automata on ω-Words, Theoretical Computer Science vol.32: 321-330, 1984.
15. R. Milner. Communication and Concurrency. Prentice-Hall, 1989.
16. M. Mukund. Finite-state Automata on Infinte Inputs, Sixth National Seminar on Theoretical Computer Science, Banasthali Vidyapith, Banasthali, Rajasthan, 1996.
17. M. O. Rabin and D. Scott. Finite automata and their decision problems IBM Journal of Research 3(2):115-125,1959.
18. S. Safra. Complexity of automata on infinite object. PhD thesis, 1989.
19. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. CAV'00, vol. 1855 of LNCS, pp. 248-263, 2000.

QBF-Based Symbolic Model Checking for Knowledge and Time

Conghua Zhou[1], Zhenyu Chen[2], and Zhihong Tao[3]

[1] School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013, Jiangsu, China
chzhou@mail.edu.cn
[2] School of Computer Science and Engineering, Southeast University, Nanjing 210096, Jiangsu, China
zychen@mail.edu.cn
[3] School of Software and Microelectronics, Peking University, Beijing 100871, China
tzh21001@ss.pku.edu.cn

Abstract. For temporal and epistemic property CTLK we propose a new symbolic model checking technique based on Quantified Boolean Formula(QBF). The verification approach is based on an adaption of the technique of bounded model checking. We decide the validity of a CTLK formula ψ in the finite reachable state space of a system, and reduce the validity to a QBF which is satisfiable if and only if ψ is validated. The new technique avoids the space blow up of BDDs, and sometimes speeds up the verification.

1 Introduction

Model checking[1] is an automatic verification technique which focussed predominantly on system specification expressed in temporal logic-linear temporal logic in the case of SPIN[2,3] and FORSPEC[4], branching temporal logic in the case of SMV[5] and its relatives. In 1991, Halpern and Vardi proposed the use of model checking as an alternative to the deduction for logics of knowledge. Since then many researchers focused on the model checking problem of multi-agent systems(MAS)[6,7,8]. In the model checking of MAS, properties are expressed with temporal logics of knowledge, and interpreted systems are used to describe behaviors of MAS. The state explosion problem is still main problem in the model checking of MAS.

Recently bounded model checking based on SAT[9] has been introduced as a complementary technique to BDD-based symbolic model checking[5]. Since bounded model checking was introduced researchers pay much attention to the bounded model checking technique for temporal logics of knowledge[10,11,12,13]. They proposed many new temporal logics of knowledge such as CTLK[10], CTL*K. Their researches showed that bounded model checking based on SAT can overcome the state explosion problem efficiently in the model checking of MAS. However, their verification properties are only universal fragments of temporal logics of knowledge. We proposed a new bounded model checking technique based on QBF[14,15] for the whole CTLK.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 386–397, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Our basic idea is to decide the validation of a CTLK formula ψ in the finite step reachable state space of a system, and reduce the validation to a QBF which is satisfiable if and only if ψ is validated. Like SAT evaluation, QBF evaluation can be search based and does not suffer from the potential space explosion problem of BDDs. Modern QBF solvers can handle QBF satisfiability problems with thousands of variables or more. Therefore our new symbolic technique is feasible.

The paper is organized as follows. In Section 2 we recall some basic concepts about QBF. In Section 3, we introduce Interpreted System Semantics. In Section 4, we define the bounded semantics of CTLK. In Section 5, the translation algorithm is presented. Section 6 concludes the paper.

2 Quantified Boolean Formula

The set of quantified boolean formulas(QBF) is defined inductively as follows:

Definition 1. *(Quantified Boolean Formula)*

(1) Every propositional formula is a quantified boolean formula.
(2) Let ϕ be a quantified boolean formula and let x(respectively y) be a propositional variable. Then both $\forall x\phi$ and $\exists y\phi$ are quantified boolean formulas.
(3) Only the formulas given by (1) and (2) above are quantified boolean formulas.

According to Definition 1, quantified boolean formulas are always in prenex form, i.e. they consist of a sequence of quantifiers, the *prefix*, followed by a quantifier free propositional formula, the so-called *matrix* of the formula: $\Phi = Q_1x_1 \ldots Q_nx_n\phi$ with $Q_i \in \{\exists, \forall\}$ and x_i a propositional variable for $1 \leq i \leq n$. The semantics of a QBF Φ can be defined recursively as follows. If the prefix is empty, then the satisfiability of Φ is defined according to the truth tables of propositional logic. If Φ is $\exists x\phi$ (resp. $\forall x\phi$), Φ is satisfiable if and only if Φ_x or (resp. and) Φ_{-x} are satisfiable. Here Φ_x is the QBF obtained from by substituting x with $True$, Φ_{-x} is the QBF obtained from by substituting x with $False$.

3 Interpreted System Semantics

Interpreted systems are mainstream semantics for temporal logics of knowledge. We assume that the modelling system composed of multiple agents, each of which is an independently operating process. Let $Ag = \{1, \ldots, n\}$ denote the set of agents. We assume that each agent $i \in Ag$ can be any of a set L_i of local states. An agent's local state contains all the information required to completely characterize the state of the agent: the value of each of its local variables, together with the value of its program counter. In particular, the information available to an agents is determined by its local state. The state of a system at any moment can be characterized by a tuple $(l_1, \ldots, l_n)$, where $l_i \in L_i$ is the local state of agent i at this moment. We let $G \subseteq L_1 \times \cdots \times L_n$ denote the global states

of the system. Notice that we have not explicitly introduced environments. For simplicity we assume that an environment can be modelled as an agent in the system.

Definition 2. *(Models) Given a set of atomic propositions AP and a set of agents* $Ag = \{1, \ldots, n\}$*, a temporal knowledge model(simply a model) over AP and Ag is a pair* $M = (\mathcal{K}, L)$ *with* $\mathcal{K} = (G, S, R, s_0, \sim_1, \ldots, \sim_n)$*, where G is the finite set of the global states for the system(simply states);* $R \subseteq G \times G$ *is a total binary relation on G; S is a set of reachable global states from* s_0*, i.e.,* $S = \{s \in G | (s_0, s) \in R^*\}$*(*$R^*$ *denotes the transitive closure of R);* $s_0 \in S$ *is the initial state;* $\sim_i \subseteq G \times G (i \in Ag)$ *is a knowledge accessibility relation for each agent* $i \in Ag$ *defined* $s \sim_i s'$ *iff* $l_i(s') = l_i(s)$*, where the function* $l_i : G \to L_i$ *returns the local state of agent i from a global state s;* $L : G \to 2^{AP}$ *is a function which labels each state with a subset of the atomic propositions set AP.*

Knowledge relations. It is obvious that in the system model M, the relation $\sim_i$ is an equivalence relation. Let $\Gamma \subseteq Ag$. Given the knowledge relations for the agents in Γ, the union of $\Gamma's$ accessibility relations defines the knowledge relation corresponding to the modality of everybody knows: $\sim_\Gamma^E = \bigcup_{i \in \Gamma} \sim_i$. $\sim_\Gamma^C$ denotes the transitive closure of $\sim_\Gamma^E$, and corresponds to the relation used to interpret the modality of common knowledge. The intersection of $\Gamma's$ accessibility relations defines the knowledge relation corresponding to the modality of distributed knowledge: $\sim_\Gamma^D = \bigcap_{i \in \Gamma} \sim_i$.

A path in system model M is an infinite sequence of states $\pi = s_0, s_1, \ldots$ such that $(s_i, s_{i+1}) \in R$ for each $i \geq 0$. A k-path is an finite sequence of states $\pi = s_0, s_1, \ldots, s_k$ such that $(s_i, s_{i+1}) \in R$ for each $0 \leq i \leq k-1$. $Path_k$ is a set of all k-paths. For a path $\pi = s_0, s_1, \ldots$ or a k-path $\pi = s_0, s_1, \ldots, s_k$, let $\pi(k) = s_k$ and π^k denote the suffix of π starting from the kth state.

3.1 Computation Tree Logic of Knowledge-CTLK

Definition 3. *(Syntax of CTLK) Let AP be a set of propositional variables containing the symbol true. The set of CTLK formulas is defined inductively as follows:*

- *if* $p \in AP$*, then p is a CTLK formula.*
- *if* α *and* β *are CTLK formulas, then so are* $\neg\alpha, \alpha \wedge \beta$ *and* $\alpha \vee \beta$*.*
- *if* α *and* β *are CTLK formulas, then so are* $EX\alpha, EG\alpha, E\alpha U\beta$*, and* $E\alpha R\beta$ *are CTLK formulas.*
- *if* α *is a CTLK formula, then so are* $\overline{K}_\Gamma\alpha$*, for* $i \in Ag$*.*
- *if* α *is a CTLK formula, then so are* $\overline{D}_\Gamma\alpha$*,* $\overline{C}_\Gamma\alpha$*,* $\overline{E}_\Gamma\alpha$*, for* $\Gamma \subseteq Ag$*.*

The basic modalities are defined by derivation as follows: $F\alpha := true U \alpha$, $AX\alpha := \neg EX\neg\alpha$, $AG\alpha := \neg EF\neg\alpha$, $A\alpha R\beta := \neg E(\neg\alpha U \neg\beta)$, $A\alpha U\beta := \neg E(\neg\alpha R \neg\beta)$, $D_\Gamma\alpha := \neg\overline{D}_\Gamma\neg\alpha, C_\Gamma\alpha := \neg\overline{C}_\Gamma\neg\alpha, E_\Gamma\alpha := \neg\overline{E}_\Gamma\neg\alpha, K_i\alpha := \neg\overline{K}_i\neg\alpha$.

Definition 4. *(Semantics of CTLK) Let M be a model, s be a state,* π *be path, and* α, β *be formulas of CTLK.* $M, s \models \alpha$ *denotes that* α *is true at the state s*

in the model M. M is omitted, if it is implicitly understood. The relation $\models$ is defined inductively as follows:

- $s \models p$ *iff* $p \in L(s)$,
- $s \models \neg\alpha$ *iff* $s \not\models \alpha$,
- $s \models \alpha \vee \beta$ *iff* $s \models \alpha$ *or* $s \models \beta$,
- $s \models \alpha \wedge \beta$ *iff* $s \models \alpha$ *and* $s \models \beta$,
- $s \models EX\alpha$ *iff there exists a path* π *starting from* s *such that* $\pi(1) \models \alpha$.
- $s \models EG\alpha$ *iff there exists a path* π *starting from* s *such that for all* $i \geq 0$, $\pi(i) \models \alpha$.
- $s \models E\alpha U\beta$ *iff there exists a path* π *starting from* s *such that for some* $i \geq 0$,$[\pi(i) \models \beta$ *and for all* $j < i, \pi(i) \models \alpha]$.
- $s \models \overline{K}_i\alpha$ *iff* $\exists s' \in S(s \sim_i s'$ *and* $s' \models \alpha)$.
- $s \models \overline{D}_\Gamma\alpha$ *iff* $\exists s' \in S(s \sim_\Gamma^D s'$ *and* $s' \models \alpha)$.
- $s \models \overline{E}_\Gamma\alpha$ *iff* $\exists s' \in S(s \sim_\Gamma^E s'$ *and* $s' \models \alpha)$.
- $s \models \overline{C}_\Gamma\alpha$ *iff* $\exists s' \in S(s \sim_\Gamma^C s'$ *and* $s' \models \alpha)$.

Definition 5. *(Validity) A CTLK formula ϕ is valid on a model M, denoted $M \models \phi$, iff $M, s_0 \models \phi$, i.e., ϕ is true at the initial state of the model M.*

4 Bounded Semantics of CTLK

The basic idea of bounded model checking for CTLK is to consider only a finite reachable state space of a system that may be a solution to the model checking problem. We restrict the length of the prefix by a certain bound k. In practice we progressively increase the bound, looking for longer and longer possible witness. In the following we give a bounded semantics of CTLK that is an approximation to the unbounded semantics of CTLK . It allows us to define the bounded model checking problem.

Definition 6. *Let M be an interpreted system, π be a k-path. A function loop : $Path_k \to 2^N$ is defined as :*

$$loop(\pi) = \{l | l \leq k \text{ and } (\pi(k), \pi(l)) \in R\}$$

Definition 7. *(Reachable in k-Step) We call a state s' reachable in k-step from state s if there is a finite path between s and s' whose length is no more than k.*

Without loss of expressibility we assume that all CTLK formulas are negation normal formulas.

Definition 8. *(Bounded Semantics for CTLK) Let M be an interpreted system, f, g be CTLK formulas. $M, s \models_k f$ denotes that f is true at the state s in bounded semantics. M is omitted if it is implicitly understood. The relation $\models_k$ is defined inductively as follows:*

- $s \models_k p$ *iff* $p \in L(s)$, $\quad s \models_k f \vee g$ *iff* $s \models_k f$ *or* $s \models_k g$,
- $s \models_k \neg p$ *iff* $p \notin L(s)$, $\quad s \models_k f \wedge g$ *iff* $s \models_k f$ *and* $s \models_k g$.

- $s \models_k EXf$ *iff there is a k-path* π *such that* $\pi(0) = s$ *and* $\pi(1) \models_k f$.
- $s \models_k AXf$ *for each k-path* π *if* $\pi(0) = s$ *then* $\pi(1) \models_k f$.
- $s \models_k EFf$ *there is a k-path* π *such that* $\pi(0) = s$ *and* $\bigvee_{i=0}^{k} \pi(i) \models_k f$.
- $s \models_k AFf$ *iff for each k-path* π *if* $\pi(0) = s$ *then* $\bigvee_{i=0}^{k} \pi(i) \models_k f$.
- $s \models_k EGf$ *iff there is a k-path such that* $\pi(0) = s$, $\bigwedge_{j=0}^{k} \pi(j) \models_k f$, *and* $loop(\pi) \neq \emptyset$.
- $s \models_k AGf$ *iff for each k-path* π *if* $\pi(0) = s$ *then* $\bigwedge_{j=0}^{k} \pi(j) \models_k f$, *and for each state* s' *if* s' *satisfies* $R(\pi(k), s')$ *then* s' *is reachable from* s *in k-step.*
- $s \models_k EfUg$ *iff there is a k-path* π *such that* $\pi(0) = s$ *and* $\exists 0 \leq j \leq k(\pi(j) \models_k g$ *and* $\bigwedge_{i=0}^{j-1} \pi(i) \models_k f)$.
- $s \models_k AfUg$ *iff for each k-path* π *if* $\pi(0) = s$ *then*$\exists 0 \leq j \leq k(\pi(j) \models_k g$ *and* $\bigwedge_{i=0}^{j-1} \pi(i) \models_k f))$
- $s \models_k EfRg$ *iff there is a k-path* π *such that* $\pi(0) = s$ *and one of the following conditions holds: (1)*$loop(\pi) \neq \emptyset \wedge \bigwedge_{j=0}^{k} \pi(j) \models_k g$.*(2)* $\exists 0 \leq j \leq k(\pi(j) \models_k f \wedge \bigwedge_{i=0}^{j} \pi(i) \models_k g)$.
- $s \models_k AfRg$ *iff for each k-path* π *if* $\pi(0) = s$ *then one of the following conditions holds:(1)* $\bigwedge_{j=0}^{k} \pi(j) \models_k g$ *and for each state* s' *if* $R(\pi(k), s')$ *holds then* s' *is reachable from* s *in k-step.(2)* $\exists 0 \leq j \leq k(\pi(j) \models_k f \wedge \bigwedge_{i=0}^{j} \pi(i) \models_k g)$.
- $s \models_k \overline{K}_i f$ *iff there is a k-path* π *starting from the initial state* s_0 *such that for some* $0 \leq j \leq k$,$s \sim_i \pi(j)$ *and* $\pi(j) \models_k f$.
- $s \models_k K_i f$ *iff for each k-path* π *starting from the initial state* s_0, *the following two conditions hold: (1) for all* $0 \leq j \leq k$, *if* $s \sim_i \pi(j)$ *then* $\pi(j) \models_k f$; *(2) for each state* s', *if* s' *satisfies* $R(\pi(k), s')$ *then* s' *is reachable from* s_0 *in k-step.*
- $s \models_k \overline{E}_\Gamma f$ *iff there is a k-path* π *starting from the initial state* s_0 *such that for some* $0 \leq j \leq k$,$s \sim_\Gamma^E \pi(j)$ *and* $\pi(j) \models_k f$.
- $s \models_k E_\Gamma f$ *iff for each k-path* π *starting from the initial state* s_0, *for all* $0 \leq j \leq k$, *the following two conditions hold: (1) if* $s \sim_\Gamma^E \pi(j)$ *then* $\pi(j) \models_k f$; *(2) for each state* s' *if* s' *satisfies* $R(\pi(k), s')$ *then* s' *is reachable from* s_0 *in k-step.*
- $s \models_k \overline{D}_\Gamma f$ *iff there is a k-path* π *starting from the initial state* s_0 *such that for some* $0 \leq j \leq k$,$s \sim_\Gamma^D \pi(j)$ *and* $\pi(j) \models_k f$.
- $s \models_k D_\Gamma f$ *iff for each k-path* π *starting from the initial state* s_0, *for all* $0 \leq j \leq k$, *the following two conditions hold: (1) if* $s \sim_\Gamma^D \pi(j)$ *then* $\pi(j) \models_k f$;

(2) for each state s' if s' satisfies $R(\pi(k), s')$ then s' is reachable from s_0 in k-step.

- $s \models_k \overline{C}_\Gamma f$ *iff there is a k-path π starting from the initial state s_0 such that for some $0 \leq j \leq k$,$(s, \pi(j)) \in {E_\Gamma}^k$ and $\pi(j) \models_k f$.*
- $s \models_k C_\Gamma f$ *iff for each k-path π starting from the initial state s_0, the following two conditions hold: (1) for all $0 \leq j \leq k$, if $(s, \pi(j)) \in {E_\Gamma}^k$ then $\pi(j) \models_k f$; (2) for each state s' if s' satisfies $R(\pi(k), s')$ then s' is reachable from s_0 in k-step.*

Now we describe the model checking problem ($M, s \models f$) can be reduced to the bounded model checking problem ($M, s \models_k f$).

Theorem 1. *(Soundness) Let M be an interpreted system structure, k be a positive integer, s be a state of M, and f be a CTLK formula. Then $M, s \models_k f$ implies $M, s \models f$.*

Proof. By induction on f. The theorem follows directly for the propositional variables and their negations. Next assume that the hypothesis holds for all the proper subformulas of f. If f is equal to either $\alpha \vee \beta$ or $\alpha \wedge \beta$, then it is easy to check that the theorem holds. Consider f to be of the following forms:

- Let $f = EX\alpha$. By the definition of bounded semantics, there is a k-path π such that $\pi(0) = s$ and $M, \pi(1) \models_k \alpha$. By the induction $M, \pi(1) \models \alpha$. By the definition of the unbounded semantics $M, s \models f$.
- Let $f = AX\alpha$. By the definition of bounded semantics, for any k-path π if $\pi(0) = s$ then $M, \pi(1) \models_k \alpha$. By the induction for any k-path π if $\pi(0) = s$ then $M, \pi(1) \models \alpha$. By the definition of the unbounded semantics $M, s \models f$.
- Let $f = EG\alpha$. By the definition of bounded semantics, there exists a k-pathπ, such that $loop(\pi) \neq \emptyset$ and $\bigwedge_{j=0}^{k} M, \pi(j) \models_k \alpha$. By the induction $\bigwedge_{j=0}^{k} M, \pi(j) \models \alpha$. Without loss of generality we assume that $l \in loop(\pi)$, then for the infinite path $\pi' = \pi(0), \ldots, \pi(l-1), (\pi(l), \ldots, \pi(k))^\omega$, for each $i \geq 0$, $M, \pi'(i) \models \alpha$. By the definition of the unbounded semantics $M, s \models f$.
- Let $f = AG\alpha$. We assume that $M, s \not\models AG\alpha$, that is $M, s \models EF\neg\alpha$. By the definition of the unbounded semantics there is a state s' such that s' is reachable from s and $M, s' \models \neg\alpha$. Let d be the shortest distance between s and s'. If $d \leq k$, then by the definition of $M, s \models_k AG\alpha$, $M, s' \models_k \alpha$. by the induction $M, s' \models \alpha$. Therefore $d > k$. Assume that the shortest path between s and s' is π'. By the bounded semantics of $M, s \models_k AG\alpha$, $\pi'(k+1)$ is reachable from s in k-step. Therefore there is a path connecting s and s' which's length is $d-1$. So our assume $M, s \not\models AG\alpha$ is infeasible. That is $M, s \models AG\alpha$.
- Let $f = E\alpha U\beta$. By the definition of bounded semantics, there exists a k-path π, an integer j with $0 \leq j \leq k$, such that $M, \pi(j) \models_k \beta$ and $\bigwedge_{i=0}^{j-1} M, \pi(i) \models_k \alpha$. By the induction $M, \pi(j) \models \beta$ and $\bigwedge_{i=0}^{j-1} M, \pi(i) \models \alpha$. By the definition of the unbounded semantics $M, s \models f$.

- Let $f = A\alpha U\beta$. By the definition of bounded semantics, for each k-path π with $\pi(0) = s$, there is an integer j with $0 \leq j \leq k$, such that $M, \pi(j) \models_k \beta$ and $\bigwedge_{i=0}^{j-1} M, \pi(i) \models_k \alpha$. By the induction $M, \pi(j) \models_k \beta$ and $\bigwedge_{i=0}^{j-1} M, \pi(i) \models_k \alpha$. By the definition of the unbounded semantics $M, s \models f$.
- Let $f = E\alpha R\beta$. By the definition of bounded semantics, there exists a k-path π starting from s satisfying one of the following conditions: (1)there is an integer j with $0 \leq j \leq k$, such that $M, \pi(j) \models_k \alpha$ and $\bigwedge_{i=0}^{j} M, \pi(i) \models_k \beta$. (2) for each $i \geq 0$, $loop(\pi) \neq \emptyset$ and $M, s \models_k \beta$. For the case 1, the proof is the same with $f = E\alpha U\beta$. For the case 2, the proof is the same with $f = EG\alpha$.
- Let $f = A\alpha R\beta$. By the definition of bounded semantics, $M, s \models_k f$ iff for each k-path starting from s one of the following condition holds:

  1. $\bigwedge_{j=0}^{k} \pi(j) \models_k g$ and for each state s' if $R(\pi(k), s')$ holds then s' is reachable from s in k-step.

  2. $\exists 0 \leq j \leq k(\pi(j) \models_k f \wedge \bigwedge_{i=0}^{j} \pi(i) \models_k g)$.

  For the case 1, the proof is the same with $f = AG\alpha$. For the case 2, the proof is the same with $f = A\alpha U\beta$.
- Let $f = \overline{K}_i\alpha$. By the definition of bounded semantics, there is a state s' reachable from s_0 in k-step with $s \sim_i s'$ and $s' \models_k \alpha$. By the induction, $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models \overline{K}_i\alpha$.
- Let $f = K_i\alpha$. By the definition of bounded semantics, if a state s' reachable from s_0, then s' reachable from s_0 in k-step. By the induction $s' \models_k \alpha$ implies $s' \models \alpha$. Therefore for each state s' reachable from s_0, if $s \sim_i s'$ then $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models K_i\alpha$.
- Let $f = \overline{E}_\Gamma\alpha$. By the definition of bounded semantics, there is a state s' reachable from s_0 in k-step with $s \sim_\Gamma^E s'$ and $s' \models_k \alpha$. By the induction, $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models E_\Gamma\alpha$.
- Let $f = E_\Gamma\alpha$. By the definition of bounded semantics, if a state s' reachable from s_0, then s' reachable from s_0 in k-step. By the induction $s' \models_k \alpha$ implies $s' \models \alpha$. Therefore for each state s' reachable from s_0, if $s \sim_\Gamma^E s'$ then $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models E_\Gamma\alpha$.
- Let $f = \overline{D}_\Gamma\alpha$. By the definition of bounded semantics, there is a state s' reachable from s_0 in k-step with $s \sim_\Gamma^D s'$ and $s' \models_k \alpha$. By the induction, $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models D_\Gamma\alpha$.
- Let $f = D_\Gamma\alpha$. By the definition of bounded semantics, if a state s' reachable from s_0, then s' reachable from s_0 in k-step. By the induction $s' \models_k \alpha$ implies $s' \models \alpha$. Therefore for each state s' reachable from s_0, if $s \sim_\Gamma^D s'$ then $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models D_\Gamma\alpha$.
- Let $f = \overline{C}_\Gamma\alpha$. By the definition of bounded semantics, there is a state s' reachable from s_0 in k-step with $(s, s') \in {E_\Gamma}^k$ and $s' \models_k \alpha$. $(s, s') \in {E_\Gamma}^k$ implies $s \sim_\Gamma^C s'$. And by the induction, $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models C_\Gamma\alpha$.

– Let $f = C_\Gamma\alpha$. By the definition of bounded semantics, if a state s' reachable from s, then s' reachable from s_0 in k-step. By the induction $s' \models_k \alpha$ implies $s' \models \alpha$, and $(s, s') \in {E_\Gamma}^k$ implies $s \sim_\Gamma^C s'$. Therefore for each state s' reachable from s_0, if $s \sim_\Gamma^C s'$ then $s' \models \alpha$. By the definition of the unbounded semantics $M, s \models C_\Gamma\alpha$.

We recall what is the recurrence diameter of a system. Given a system model M, its recurrence diameter, denoted as $rd(M)$, is the minimal number $d \in N$ with the following property. For every sequence of states $s_0, \ldots s_{d+1}$ with $(s_i, s_{i+1}) \in R$ for $i \leq d$, there exists $j \leq d$ such that $s_{d+1} = s_j$.

Theorem 2. *(Completeness) Let M be a system model, s be a state of M, f be a CTLK formula without knowledge operators $\overline{C}_\Gamma, C_\Gamma$, and $k = rd(M) + 1$. If $M, s \models f$ then $M, s \models_k f$.*

Theorem 2 can be directly proved by induction on f. Limited by space we omit the proof's details.

Theorem 3. *Let M be a system model, s be a state of M, f be a CTLK formula and $k = |M|$. If $M, s \models f$ then $M, s \models_k f$.*

Proof. We only need to consider $f = C_\Gamma\alpha$. It is clear that $s \sim_\Gamma^C s'$ if and only if there is an integer k with $k \leq |M|$ such that $(s, s') \in {E_\Gamma}^k$. By the unbounded semantics, $s \models C_\Gamma\alpha$ iff for each state s' reachable from s_0, if $s \sim_\Gamma^C s'$ then $s' \models \alpha$. By the induction $s' \models_k \alpha$. It is clear that s' is reachable from s_0 in $|M|-$step. Therefore by the definition of bounded semantics $M, s \models_k C_\Gamma\alpha$.

5 The BMC Algorithm for CTLK

In the previous subsection, we defined the semantics for bounded model checking. We now reduce bounded model checking for CTLK to QBF problem. First we assume that each state can be represented by a vector of state variables $s = (s[1], s[2], \ldots, s[n])$, where $s[i]$ for $i = 1, \ldots, n$ are propositional variables. In the following we use the notation $s^{[m][n]}$ to represent a state, where $[m]$ is used distinguish the different paths , [n] used to represent the current position. For notational convenience we give the following definitions:

– $p(s^{[m][n]}) := p \in L(s^{[m][n]}); I(s^{[m][n]}) := (s^{[m][n]} \leftrightarrow s_0)$.
– $\exists S_k^{[m]} := \exists s^{[m][0]} \ldots \exists s^{[m][k]}$. $\forall S^{[m]} := \forall s^{[m][0]} \ldots \forall s^{[m][k]}$, where k represents the length of a finite path.
– $Path_k^{[m]} := \forall w, v(\bigvee_{i=0}^{k-1}(w \leftrightarrow s^{[m][i]} \wedge v \leftrightarrow s^{[m][i+1]}) \rightarrow R(w, v)$, where k represents the length of a finite path. (The satisfiability of $Path_k^{[m]}$ means that the state sequence $s^{[m][0]}, ..., s^{[m][k]}$ is k-path.)
– $Loop_k^{[m]} := \exists w(\bigvee_{i=0}^{k}(w \leftrightarrow s^{[m][i]}) \rightarrow R(s^{[m][k]}, w))$, where k represents the length of a finite path. (The satisfiability of $Loop_k^{[m]}$ means that $loop(s^{[m][0]}, \ldots, s^{[m][k]}) \neq \emptyset$).

- $K_i(s^{[m][n]}, s^{[h][l]}) := (l_i(s^{[m][n]}) \leftrightarrow l_i(s^{[h][l]}))$
- $E_\Gamma(s^{[m][n]}, s^{[h][l]}) := \bigvee_{i \in \Gamma} K_i(s^{[m][n]}, s^{[h][l]})$.
- $D_\Gamma(s^{[m][n]}, s^{[h][l]}) := \bigwedge_{i \in \Gamma} K_i(s^{[m][n]}, s^{[h][l]})$.
- $E_\Gamma^k(s^{[m][n]}, s^{[h][l]}) := \exists S_k^{[m+1]}(\bigwedge_{i=0}^{k-1} E_\Gamma(s^{[m+1][i]}, s^{[m+1][i+1]}) \wedge \bigvee_{i=0}^{k-1}(s^{[m+1][i]} \leftrightarrow s^{[m][n]} \wedge s^{[m+1][i]} \leftrightarrow s^{[h][l]}))$.
- $\delta_k^{[m][n]} := \forall s^{[m+1][k+1]}(R(s^{[m+1][k]}, s^{[m+1][k+1]}) \rightarrow \exists S_k^{[m+2]}(s^{[m+2][0]} \leftrightarrow s^{[m][n]} \wedge Path_k^{[m+2]} \wedge \bigvee_{i=0}^{k}(s^{[m+2][i]} \leftrightarrow s^{[m+1][k+1]})))$. (The satisfiability of $\delta_k^{[m][n]}$ means that for the k-path $s^{[m][0]}, ..., s^{[m][k]}$, any successor of $s^{[m][k]}$ is reachable from $s^{[m][0]}$ in k steps).
- $\Delta_k^{[m][n]} := \forall s^{[m+1][k+1]}(R(s^{[m+1][k]}, s^{[m+1][k+1]}) \rightarrow \exists S_k^{[m+2]}(I(s^{[m+2][0]}) \wedge Path_k^{[m+2]} \wedge \bigvee_{i=0}^{k}(s^{[m+2][i]} \leftrightarrow s^{[m+1][k+1]})))$. (The satisfiability of $\Delta_k^{[m][n]}$ means that for the k-path $s^{[m][0]}, ..., s^{[m][k]}$, any successor of $s^{[m][k]}$ is reachable from the initial state in k steps).

Algorithm 1. (Translating Bounded Model Checking for CTLK into QBF)

- $[M, p]_k^{[m][n]} := p(s^{[m][n]})$. $[M, \neg p]_k^{[m][n]} := \neg p(s^{[m][n]})$
- $[M, \alpha \vee \beta]_k^{[m][n]} := [M, \alpha]_k^{[m][n]} \vee [M, \beta]_k^{[m][n]}$. $[M, \alpha \wedge \beta]_k^{[m][n]} := [M, \alpha]_k^{[m][n]} \wedge [M, \beta]_k^{[m][n]}$.
- $[M, EX\alpha]_k^{[m][n]} := \exists s^{[m+1][1]}(R(s^{[m][n]}, s^{[m+1][1]}) \wedge [M, \alpha]^{[m+1][1]})$.
- $[M, AX\alpha]_k^{[m][n]} := \forall s^{[m+1][1]}(R(s^{[m][n]}, s^{[m+1][1]}) \rightarrow [M, \alpha]_k^{[m+1][1]})$.
- $[M, EF\alpha]_k^{[m][n]} := \exists S_k^{[m+1]}((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]} \wedge \bigvee_{i=0}^{k}[M, \alpha]_k^{[m+1][i]})$.
- $[M, AF\alpha]_k^{[m][n]} := \forall S_k^{[m+1]}(((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]}) \rightarrow \bigvee_{i=0}^{k}[M, \alpha]_k^{[m+1][i]})$.
- $[M, EG\alpha]_k^{[m][n]} := \exists S_k^{[m+1]}((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]} \wedge Loop_k^{[m+1]} \wedge \bigwedge_{i=0}^{k}[M, \alpha]_k^{[m+1][i]})$
- $[M, AG\alpha]_k^{[m][n]} := \forall S_k^{[m+1]}(((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]}) \rightarrow (\bigwedge_{i=0}^{k}[M, \alpha]_k^{[m+1][i]} \wedge \delta_k^{[m][n]}))$
- $[M, E\alpha U\beta]_k^{[m][n]} := \exists S_k^{[m+1]}((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]} \wedge \bigvee_{i=0}^{k}([M, \beta]_k^{[m+1][i]} \wedge \bigwedge_{j=0}^{i-1}[M, \alpha]_k^{[m+1][j]}))$
- $[M, A\alpha U\beta]_k^{[m][n]} := \forall S_k^{[m+1]}(((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]}) \rightarrow (\bigvee_{i=0}^{k}([M, \beta]_k^{[m+1][i]} \wedge \bigwedge_{j=0}^{i-1}[M, \alpha]_k^{[m+1][j]})))$

- $[M, E\alpha R\beta]_k^{[m][n]} := \exists S_k^{[m+1]}((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]} \wedge ((\bigvee_{i=0}^{k}([M,\alpha]_k^{[m+1][i]} \wedge \bigwedge_{j=0}^{i-1}[M,\beta]_k^{[m+1][j]})) \vee (Loop_k^{[m+1]} \wedge \bigwedge_{i=0}^{k}[M,\alpha]_k^{[m+1][i]})))$
- $[M, A\alpha R\beta]_k^{[m][n]} := \forall S_k^{[m+1]}(((s^{[m+1][0]} \leftrightarrow s^{[m][n]}) \wedge Path_k^{[m+1]}) \rightarrow ((\bigvee_{i=0}^{k}([M,\alpha]_k^{[m+1][i]} \wedge \bigwedge_{j=0}^{i-1}[M,\beta]_k^{[m+1][j]})) \vee (\bigwedge_{i=0}^{k}[M,\alpha]_k^{[m+1][i]} \wedge \delta_k^{[m][n]}))$
- $[M, \overline{K}_i\alpha]_k^{[m][n]} := \exists S_k^{[m+1]}(I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]} \wedge \bigvee_{j=0}^{k}(K_i(s^{[m][n]}, s^{[m+1][j]}) \wedge [\alpha]_k^{[m+1][j]}))$.
- $[M, K_i\alpha]_k^{[m][n]} := \forall S_k^{[m+1]}(((I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]}) \rightarrow \bigwedge_{j=0}^{k}(K_i(s^{[m][n]}, s^{[m+1][j]}) \rightarrow [\alpha]_k^{[m+1][j]})) \wedge \Delta_k^{[m][n]})$.
- $[M, \overline{E}_\Gamma\alpha]_k^{[m][n]} := \exists S_k^{[m+1]}(I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]} \wedge \bigvee_{j=0}^{k}(E_\Gamma(s^{[m][n]}, s^{[m+1][j]}) \wedge [\alpha]_k^{[m+1][j]}))$.
- $[M, E_\Gamma\alpha]_k^{[m][n]} := \forall S_k^{[m+1]}(((I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]}) \rightarrow \bigwedge_{j=0}^{k}(E_\Gamma(s^{[m][n]}, s^{[m+1][j]}) \rightarrow [\alpha]_k^{[m+1][j]})) \wedge \Delta_k^{[m][n]})$.
- $[M, \overline{D}_\Gamma\alpha]_k^{[m][n]} := \exists S_k^{[m+1]}(I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]} \wedge \bigvee_{j=0}^{k}(D_\Gamma(s^{[m][n]}, s^{[m+1][j]}) \wedge [\alpha]_k^{[m+1][j]}))$.
- $[M, D_\Gamma\alpha]_k^{[m][n]} := \forall S_k^{[m+1]}(((I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]}) \rightarrow \bigwedge_{j=0}^{k}(D_\Gamma(s^{[m][n]}, s^{[m+1][j]}) \rightarrow [\alpha]_k^{[m+1][j]})) \wedge \Delta_k^{[m][n]})$.
- $[M, \overline{C}_\Gamma\alpha]_k^{[m][n]} := \exists S_k^{[m+1]}(I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]} \wedge \bigvee_{j=0}^{k}(E_\Gamma^k(s^{[m][n]}, s^{[m+1][j]}) \wedge [\alpha]_k^{[m+1][j]}))$.
- $[M, C_\Gamma\alpha]_k^{[m][n]} := \forall S_k^{[m+1]}(((I(s_k^{[m+1][0]}) \wedge Path_k^{[m+1]}) \rightarrow \bigwedge_{j=0}^{k}(E_\Gamma^k(s^{[m][n]}, s^{[m+1][j]}) \rightarrow [\alpha]_k^{[m+1][j]})) \wedge \Delta_k^{[m][n]})$.

Definition 9. *(General Translation)* $[M^{f,s}]_k := ((s \leftrightarrow s^{[0][0]}) \wedge [M, f]_k^{[0][0]})$.

5.1 The Correctness of the Translation

For the convenience of the proof note that $(s \leftrightarrow s^{[0][0]} \wedge [M, f]_k^{[0][0]})$ is satisfiable if and only if for any positive integers i, j, $(s \leftrightarrow s^{[i][j]} \wedge [M, f]_k^{[i][j]})$ is satisfiable.

Theorem 4. *$[M^{f,s}]_k$ is satisfiable if and only if $M, s \models_k f$.*

Theorem 4 can be directly proved by induction on f. Limited by space we omit the proof's details.

Corollary 1. *Let M be a system model, s be a state of M, and f be a CTLK formula, $k \in N$ be a natural number. If $[M^{f,s}]_k$ is satisfiable then $M, s \models f$.*

Corollary 2. *Let M be a system model, s be a state of M, f be a CTLK formula without knowledge operators $C_\Gamma, \overline{C}_\Gamma$, and $k = rd(M) + 1$. $M, s \models f$ if and only if $[M^{f,s}]_k$ is satisfiable.*

Corollary 3. *Let M be a system model, s be a state of M, f be a CTLK formula, and $k = |M|$. $M, s \models f$ if and only if $[M^{f,s}]_k$ is satisfiable.*

6 Summary

Model checking has had a great impact as an efficient method for algorithmic verification of finite state systems. A limiting factor in its application is the state space explosion problem. In this paper we presented that how to apply QBF procedures to model checking CTLK. We believe that our technique not only in theory is feasible, but also has the potential to handle much larger designs than what is currently possible. In the future we need to show that our theory works in practice. And optimization techniques in generating quantified Boolean formulas need to be further investigated. In addition we would like to investigate how to use domain knowledge to guide the search in QBF procedures.

References

1. E. M. Clarke, O. Grumberg, and D. Peled, Model checking (MIT Press, 2000).
2. G. Holzmann. Design and Validation of Computer Protocols. Prentice Hall International: Hemel Hempstead England, 1991.
3. G. Holzmann. The Spin model checker. IEEE Transaction on Software Engineering, 23(5):279-295, May 1997.
4. M. Y. Vardi. Branching vs. linear time: Final showdown. In T. Margaria and W. Yi, editors, Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001 pages 1-22. Springer-Verlag: Berlin, Germany, April 2001.
5. K. L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers: Boston, MA, 1993.
6. J. Y. Halpern, M. Y. Vardi. Model checking vs. theorem proving: a manifesto. In V. Lifschitz, editor. Artificial Intelligence and Mathematical Theory of Computation, pages 151-176. Academic Press, San Diego Calif 1991.433.
7. R. Fagin J. Y. Halpern, Y. Moses, M. Y. Vardi. Resaoning about Knowledge. MIT Press, 1995.
8. M. Y. Vardi. Implementing knowledge-based programs. In Proc. of the Conf. on Theoretical Aspects of Rationality and Knowledge pages 15-30, 1996.
9. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, Symbolic model checking without BDDs. in Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS99). LNCS,1579,193-207 (Springer-Verlag,1999).
10. W. Penczek, A. Lomuscio, Verifying Epistemic Properties of Multi-agent Systems via Bounded Model Checking. Fundamenta Informaticae, 55(2):167-185, 2003

11. Xiangyu Luo, Kaile Su, Abdul Sattar, Qingliang Chen, Guanfeng Lv, Bounded Model Checking Knowledge and Branching Time in Synchronous Multi-agent Systems. AAMAS'05.
12. B. Wozna, A. Lomuscio, W. Penczek, Bounded Model Checking for Knowledge and Real Time. AAMAS'05.
13. B. Wozna, A. Lomuscio, W. Penczek, Bounded model checking for deontic interpreted systems. In Proc. of LCMAS'04, volume 126 of ENTCS, 93-114. Elsevier, 2004.
14. H. Kleine Buning, Xishun Zhao, On Models for Quantified Boolean Formulas, In Lecture Notes in Computer Science 3075, 18-32, Springer-Verlag, 2004.
15. H. Kleine Buning, K. Subramani, Xishun Zhao, On Boolean Models for Quantified Boolean Horn Formulas, In Lecture Notes in Computer Scence 2919, 93-104, Springer-Verlag, 2004.

A Characterization of the Language Classes Learnable with Correction Queries*

Cristina Tîrnăucă[1] and Satoshi Kobayashi[2]

[1] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tàrraco 1, Tarragona 43005, Spain
cristina.bibire@estudiants.urv.cat
[2] Department of Computer Science, University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
satoshi@cs.uec.ac.jp

Abstract. Formal language learning models have been widely investigated in the last four decades. But it was not until recently that the model of learning from corrections was introduced. The aim of this paper is to make a further step towards the understanding of the classes of languages learnable with correction queries. We characterize these classes in terms of triples of definite finite tell-tales. This result allowed us to show that learning with correction queries is strictly more powerful than learning with membership queries, but weaker than the model of learning in the limit from positive data.

Keywords: correction query, query learning, Gold-style learning.

1 Introduction

The field of learning formal languages was practically introduced by E.M. Gold [1] in 1967, in an attempt to construct a precise model for the notion of "being able to speak a language". Gold imagined language learning as an infinite process in which the learner has access to a growing sequence of examples (*learning from text*), or both positive and negative information (*learning from informant*), and is supposed to make guesses. At some point his conjecture should be the target language and he should never change his mind afterwards.

In the same paper Gold also introduces the notion of *finite identification* (from text and informant). The main difference between this model and *learning in the limit model* is that the learner has to stop the presentation of information at some finite time when he "feels" that he has received enough, and state the identity of the target language.

In [2] D. Angluin gives several necessary and sufficient conditions for a class of languages to be learnable in the limit from positive data. Twelve years later, Y. Mukouchi [3] describes the class of languages finitely identifiable from text

* This work was possible thanks to the FPU Fellowship AP2004-6968 from the Spanish Ministry of Education and Science.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 398–407, 2007.
© Springer-Verlag Berlin Heidelberg 2007

(informant) in terms of definite finite tell-tales (pairs of definite finite tell-tales, respectively).

All the models mentioned so far are also known in the literature as Gold-style learning. A totally different language learning model is the *query learning* model, introduced by Angluin in 1987 [4]. In this setting the learner has access to a truthfully oracle which is allowed to answer specific kind of queries. In [4] a polynomial time query learning algorithm for the class of minimal complete deterministic finite automata (DFAs) is given, in which the learner can ask membership queries (MQs) and equivalence queries (EQs). There are though other types of possible queries: subset queries, superset queries, etc.

Although these two learning models seem to be quite different at a first glance, S. Lange and S. Zilles showed that in fact there is a strong correlation between them [5]. They proved that the class of languages learnable from MQs only coincides with the class of languages finitely identifiable from an informant, and that learning from EQs is equally powerful as learning in the limit from an informant.

As previously mentioned, the study of formal languages learning has its origins in the desire of a better understanding of how children learn so effortlessly their native language. Still, none of these models accurately describes the process of human language learning. Moreover, even the presence of negative information in the process of children language acquisition is subject to a long and still unsolved debate. Clearly, children are not explicitly provided negative examples (words that are not in the language or ungrammatical sentences). Yet, they are corrected when they make a mistake, and this can be thought of as negative information. Actually, these ideas can be found in Gold's paper [1]. Although he points out that "those working in the field generally agree that most children are rarely informed when they make grammatical errors, and those that are informed take little heed", he suggests that maybe "the child receives negative instances by being corrected in a way we do not recognize".

Motivated by these aspects of human language acquisition, L. Becerra-Bonache, A.H. Dediu and C. Tîrnăucă introduced in [6] a new type of query, namely correction query (CQ). A CQ is a slightly modified type of MQ: instead of a 'yes'/'no' answer, the learner receives a correcting string (given s in Σ^*, the correcting string of s with respect to the language L is the smallest strings s' such that ss' belongs to L, if such string exists, and a special symbol otherwise). The same article presents a polynomial time algorithm which infers minimal complete DFAs using CQs and EQs.

In this paper we characterize the language classes learnable with CQs for which the teacher can be effectively implemented (the answers to CQs are computable) by means of triples of definite finite tell-tales (Section 3). We consider only classes of recursive languages, and neglect time complexity issues. Preliminary notions and results are presented in Section 2. In Section 4, using this characterization, we show some relations between our learning model (learning with CQs) and other well-known learning models (like the model of learning

with MQs, or the model of learning in the limit from positive data). Concluding remarks and future work ideas are presented in Section 5.

2 Preliminaries

We assume that the reader is familiar with basic notions from formal language theory. A wealth of further information about this area can be found in [7].

Let Σ be a finite alphabet of symbols. By Σ^* we denote the set of all finite strings of symbols from Σ. A *language* is any set of strings over Σ. The length of a string w is denoted by $|w|$, and the concatenation of two strings u and v by uv or $u \cdot v$. The empty string (i.e., the unique string of length 0) is denoted by λ. If $w = uv$ for some $u, v \in \Sigma^*$, we say that u is a prefix of w and v is a suffix of w. By $Pref(L)$ we denote the set $\{w \in \Sigma^* \mid \exists w' \in \Sigma^* \text{ such that } ww' \in L\}$.

Assume that Σ is a totally ordered set and let $\prec_L$ be the lexicographical order on Σ^*. Then, the *lex-length order* $\prec$ on Σ^* is defined by: $u \prec v$ if either $|u| < |v|$, or else $|u| = |v|$ and $u \prec_L v$. In other words, strings are compared first according to length and then lexicographically.

Let $\mathcal{C}$ be a class of recursive languages over Σ^*. We say that $\mathcal{C}$ is an *indexable class* if there is an effective enumeration $(L_i)_{i\geq 1}$ of all and only the languages in $\mathcal{C}$ such that membership is uniformly decidable, i.e., there is a computable function that, for any $w \in \Sigma^*$ and $i \geq 1$, returns 1 if $w \in L_i$, and 0 otherwise. Such an enumeration will subsequently be called an *indexing* of $\mathcal{C}$.

In the sequel we might say that $\mathcal{C} = (L_i)_{i\geq 1}$ is an indexable class and understand that $\mathcal{C}$ is an indexable class and $(L_i)_{i\geq 1}$ is an indexing of $\mathcal{C}$.

2.1 Query Learning

In the query learning model a learner has access to an oracle that truthfully answers queries of a specified kind. A query learner M is an algorithmic device that, depending on the reply on the previous queries, either computes a new query, or returns a hypothesis and halts.

More formally, let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class, let $L \in \mathcal{C}$ and let M be a query learner. We say that M learns L using some type of queries if it eventually halts and its only hypothesis, say i, correctly describes L, i.e., $L_i = L$. So, M returns its unique and correct guess i after only finitely many queries. Moreover, M learns $\mathcal{C}$ using some type of queries if it learns every $L \in \mathcal{C}$ using queries of the specified type. Below we consider:

Membership queries. The input is a string w and the answer is 'yes' or 'no', depending on whether or not w belongs to the target language L.

Correction queries. The input is a string w and the answer is the smallest string (in lex-length order) w' such that ww' belongs to the target language L if $w \in Pref(L)$, and the special symbol $\theta \notin \Sigma$ otherwise. We denote the correction of a string w with respect to the language L by $C_L(w)$.

Equivalence queries. The input is an index j of some language $L_j \in \mathcal{C}$. If $L = L_j$, the answer is 'yes'. Otherwise together with the answer 'no', a counterexample from $(L_j \backslash L) \cup (L \backslash L_j)$ is supplied.

The collections of all indexable classes $\mathcal{C}$ for which there is a query learner M such that M learns $\mathcal{C}$ using membership, correction, and equivalence queries are denoted by $MemQ$, $CorQ$ and $EquQ$, respectively.

In this paper we focus on classes of languages for which $Pref(L_i)$ is recursive for all $i \geq 1$. More precisely, we consider indexable classes $\mathcal{C}$ which have the following property (A): there exists a recursive function $f : \mathbb{N}_+ \times \Sigma^* \to \Sigma^* \cup \{\theta\}$ such that $f(i, w) = v$ if and only if $C_{L_i}(w) = v$ for any w in Σ^* and L_i in $\mathcal{C}$.

For this purpose, we denote by $CorQ^{(A)}$ the collection of classes of languages in $CorQ$ for which condition (A) is satisfied. Similarly, $MemQ^{(A)}$ is defined. Clearly, for the language classes in $CorQ^{(A)}$ the answers to the correction queries can be effectively computed. That is why in this case we speak about a teacher instead of an oracle.

2.2 Gold-Style Learning

In order to present the Gold-style learning models we need some further notions, briefly explained below (for details, see [1,2,8]).

Let L be a non-empty language. A *text for* L is an infinite sequence $\sigma = w_1, w_1, w_3, \ldots$ such that $\{w_i \mid i \geq 1\} = L$. An *informant for* L is an infinite sequence $\sigma = (w_1, b_1), (w_2, b_2), (w_3, b_3), \ldots$ with $b_i \in \{0, 1\}$, $\{w_i \mid i \geq 1 \text{ and } b_i = 1\} = L$, and $\{w_i \mid i \geq 1 \text{ and } b_i = 0\} = \Sigma^* \backslash L$.

Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class. An inductive inference machine (IIM) is an algorithmic device that reads longer and longer initial segments σ of a text (informant) and outputs numbers as its hypotheses. An IIM returning some i is construed to hypothesize the language L_i. Given a text (an informant) σ for a language $L \in \mathcal{C}$, M identifies L from σ if the sequence of hypotheses output by M, when fed σ, stabilizes on a number i (i.e., past some point M always outputs the hypothesis i) with $L_i = L$. We say that M identifies $\mathcal{C}$ from text (informant) if it identifies every $L \in \mathcal{C}$ from every corresponding text (informant).

A slightly modified version is the so called model of conservative learning (see [9,10] for more details). A conservative IIM is only allowed to change its mind in case its actual guess contradicts the data seen so far.

As above, *LimTxt* (*LimInf*) denotes the collection of all indexable classes $\mathcal{C}$ for which there is an IIM M such that M identifies $\mathcal{C}$ from text (informant). One can similarly define *ConsvTxt* and *ConsvInf*, for which the inference machines should be conservative IIMs.

Although an IIM is allowed to change its mind finitely many times before returning its final and correct hypothesis, in general it is not decidable whether or not it has already output its final hypothesis. In case that for a given indexable class $\mathcal{C}$, there exists an IIM M such that given any language $L \in \mathcal{C}$ and any text (or informant) for L, the first hypothesis i output by M is already correct (i.e., $L_i = L$), we say that M finitely identifies $\mathcal{C}$ (see [1]). The corresponding models *FinTxt* and *FinInf* are defined as above.

In the sequel we present some characterizations for the classes *FinInf* and *ConsvTxt* in terms of pairs of definite finite tell-tales and finite tell-tales, respectively. Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class.

Definition 1 (Angluin, [2]). *A set T_i is a finite tell-tale of L_i if*
(1) T_i is a finite subset of L_i, and
(2) for all $j \geq 1$, if $T_i \subseteq L_j$ then L_j is not a proper subset of L_i.

Theorem 1 (Lange and Zeugmann, [11]). *An indexable class $\mathcal{C} = (L_i)_{i\geq 1}$ belongs to ConsvTxt if and only if a finite tell-tale of L_i is uniformly computable for any index i, that is, there exists an effective procedure which on any input $i \geq 1$ enumerates a finite tell-tale of L_i and halts.*

Definition 2 (Mukouchi, [3]). *A language L is consistent with a pair of sets $\langle T, F\rangle$ if $T \subseteq L$ and $F \subseteq \Sigma^* \backslash L$. The pair $\langle T, F\rangle$ is said to be a pair of definite finite tell-tales of L_i if:*
(1) T_i is a finite subset of L_i, F_i is a finite subset of $\Sigma^* \backslash L_i$, and
(2) for all $j \geq 1$, if L_j is consistent with the pair $\langle T, F\rangle$, then $L_j = L_i$.

Theorem 2 (Mukouchi, [3]). *An indexable class $\mathcal{C} = (L_i)_{i\geq 1}$ belongs to FinInf if and only if a pair of definite finite tell-tales of L_i is uniformly computable for any index i.*

Moreover, there is a strong relation between query learning models and Gold-style learning models. The following strict hierarchy can be found in [5]: $FinTxt \subset FinInf = MemQ \subset ConsvTxt \subset LimTxt \subset LimInf = EquQ$.

3 Characterization of the Class $CorQ^{(A)}$

In this section we show that an indexable class with property (A) is learnable from CQs if and only if each language of that class is uniquely characterized by a triple of finite sets. For this, we need some further definitions and notations.

We say that a language L is *consistent with a triple of sets* $\langle T, F, U\rangle$ if $T \subseteq L$, $F \subseteq \Sigma^* \backslash L$ and $U \subseteq \Sigma^* \backslash Pref(L)$.

The triple $\langle T_i, F_i, U_i\rangle$ is a *triple of definite finite tell-tales of* L_i w.r.t. $\mathcal{C} = (L_i)_{i\geq 1}$ if :
(1) T_i, F_i and U_i are finite,
(2) L_i is consistent with $\langle T_i, F_i, U_i\rangle$, and
(3) for all $j \geq 1$, if L_j is consistent with $\langle T_i, F_i, U_i\rangle$, then $L_i = L_j$.

Theorem 3. *Let $\mathcal{C} = (L_i)_{1\geq 1}$ be an indexable class with property (A). Then $\mathcal{C}$ belongs to CorQ if and only if a triple of definite finite tell-tales of L_i is uniformly computable for any index i.*

The theorem is a direct consequence of the following two propositions.

Proposition 1 (Sufficient condition). *Let $\mathcal{C} = (L_i)_{1\geq 1}$ be an indexable class. If a triple of definite finite tell-tales of L_i is uniformly computable for any index i, then $\mathcal{C}$ is in CorQ.*

Proof. Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class for which a triple of definite finite tell-tales $\langle T_i, F_i, U_i\rangle$ is uniformly computable for any index i, and let $w_1, w_2, \ldots$ be the lex-length enumeration of all words in Σ^*. If L is the target language, then the following query learning algorithm identifies L using CQs.

Algorithm 1. A correction query algorithm for the language L in $\mathcal{C}$

```
1: T := ∅, F := ∅, U := ∅, j := 1
2: while TRUE do
3:    get from the oracle the answer to C_L(w_j)
4:    if (C_L(w_j) = θ) then
5:       U := U ∪ {w_j}
6:       F := F ∪ {w_j}
7:    else
8:       T := T ∪ {w_j · C_L(w_j)}
9:       if C_L(w_j) ≠ λ then
10:         F := F ∪ {w_j}
11:      end if
12:   end if
13:   for i := 1 to j do
14:      if (T_i ⊆ T, F_i ⊆ F and U_i ⊆ U) then
15:         output i and halt
16:      end if
17:   end for
18:   j := j + 1
19: end while
```

It is not very difficult to see that if our algorithm outputs an hypothesis, then it is the correct one. Since we constructed T, F and U such that $T \subseteq L, F \subseteq \Sigma^* \backslash L$ and $U \subseteq \Sigma^* \backslash Pref(L)$, it is clear that as soon as we have $T_i \subseteq T$, $F_i \subseteq F$ and $U_i \subseteq U$, the target language L will be consistent with the triple $\langle T_i, F_i, U_i \rangle$, and hence the algorithm outputs i such that $L_i = L$.

Now, let us prove that after asking a finite number of queries, the sets T, F and U will be large enough to include T_i, F_i and U_i, respectively, where i is the smallest index such that $L_i = L$. Let k_1, k_2, k_3 and k be such that $k_1 = \max\{j \mid w_j \in T_i\}$, $k_2 = \max\{j \mid w_j \in F_i\}$, $k_3 = \max\{j \mid w_j \in U_i\}$ and $k = \max\{k_1, k_2, k_3, i\}$.

Consider the sets T, F, U constructed after receiving the corrections for the strings $w_1, s_2, \ldots, s_k$.

1. If $w \in T_i$, then $w \preceq w_k$ and $C_L(w) = \lambda$. Hence, $w \in T$.
2. If $w \in U_i$, then $w \preceq w_k$ and $C_L(w) = \theta$. Hence, $w \in U$.
3. If $w \in F_i$, then $w \preceq w_k$ and $C_L(w) \neq \lambda$. We distinguish two cases. Either $C_L(w) \in \Sigma^+$ and then w is added to F at line 10 of the algorithm, or $C_L(w) = \theta$ and w is added to F at line 6 of the algorithm. In both of the cases, $w \in F$.

We have seen that after reading corrections of at most k strings, $T_i \subseteq T$, $F_i \subseteq F$ and $U_i \subseteq U$, and since i is smaller than or equal to k, the algorithm outputs the (correct) hypothesis i. □

Proposition 2 (Necessary condition). *If $\mathcal{C} = (L_i)_{1 \geq 1}$ is in $CorQ^{(A)}$ then a triple of definite finite tell-tales of L_i is uniformly computable for any index i.*

Proof. Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class in $CorQ^{(A)}$, and take M to be a query learning algorithm which learns $\mathcal{C}$ using CQs. The following procedure computes a triple of definite finite tell-tales of L_i for any $i \geq 1$.

Algorithm 2. Computing a triple of definite finite tell-tales

1: Input: the target language L_i
2: run M on L_i, and collect the sequence of queries and answers in QA_i
3: $T_i := \{wv \mid (w,v) \in QA_i, v \neq \theta\}$
4: $F_i := \{wv' \mid (w,v) \in QA_i, v \neq \theta, v' \prec v\}$
5: $U_i := \{w \mid (w,\theta) \in QA_i\}$
6: output $\langle T_i, F_i, U_i\rangle$ and halt.

Clearly, T_i, F_i and U_i are all finite. We show that $T_i \subseteq L_i$, $F_i \subseteq \Sigma^* \backslash L_i$ and $U_i \subseteq \Sigma^* \backslash Pref(L_i)$. If $u \in T_i$, then there exist w, v in Σ^* such that $u = wv$ and $v = C_{L_i}(w)$. Hence, $u = wv \in L_i$. If $u \in F_i$, then there exist w, v, v' in Σ^* such that $v' \prec v$, $u = wv'$ and $v = C_{L_i}(w)$. Hence, $u = wv' \notin L_i$. If $u \in U_i$, then $C_{L_i}(u) = \theta$, and hence $u \in \Sigma^* \backslash Pref(L_i)$.

Let us now take j such that L_j is consistent with the triple $\langle T_i, F_i, U_i\rangle$. We compute $C_{L_j}(w)$ for each pair (w,v) in QA_i. If $v = \theta$, then $w \in U_i$. But $U_i \subseteq \Sigma^* \backslash Pref(L_j)$ implies $w \notin Pref(L_j)$, and hence $C_{L_j}(w) = \theta$. If $v \in \Sigma^* \backslash \{\theta\}$, then $wv \in T_i$ and $wv' \in F_i$ for all $v' \prec v$. But $T_i \subseteq L_j$ and $F_i \subseteq \Sigma^* \backslash L_j$ implies $wv \in L_j$ and $wv' \notin L_j$ for all $v' \prec v$. Hence, $C_{L_j}(w) = v$.

We have shown that for all $(w,v) \in QA_i$, $C_{L_j}(w) = v = C_{L_i}(w)$. Since the algorithm M is assumed to identify a unique language from the class $\mathcal{C}$, we obtain $L_i = L_j$. This makes $\langle T_i, F_i, U_i\rangle$ a triple of definite finite tell-tales of L_i. □

4 Relations to Other Learning Models

Using the results presented in the previous section, we show the relations between correction query learning models and other learning models.

4.1 A Model Included in *CorQ*

Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class. We have the following theorem.

Theorem 4. *If $\mathcal{C}$ is in FinInf, then $\mathcal{C}$ is in CorQ.*

Proof. Assume that $\mathcal{C}$ is in *FinInf*. Then cf. Theorem 2, a pair of definite finite tell-tales $\langle T_i, F_i\rangle$ of L_i is uniformly computable for any index i. We show that $\langle T_i, F_i, \emptyset\rangle$ is a triple of definite finite tell-tales for L_i. Clearly, T_i is a finite subset of L_i, F_i is a finite subset of $\Sigma^* \backslash L_i$ and the empty set is a finite subset of $\Sigma^* \backslash Pref(L_i)$. Let us now take j such that L_j is consistent with the triple $\langle T_i, F_i, \emptyset\rangle$. Because $\langle T_i, F_i\rangle$ is a pair of definite finite tell-tales for L_i, $T_i \subseteq L_j$ and $F_i \subseteq \Sigma^* \backslash L_j$, we obtain $L_j = L_i$, and hence $\langle T_i, F_i, \emptyset\rangle$ is a triple of definite finite tell-tales for L_i. Using Proposition 1, we immediately get that $\mathcal{C}$ is in *CorQ*. □

Let us now show that the inclusion is strict. Take $K_1, K_2, K_3, \ldots$ to be the collection of all finite non-empty sets of positive integers (indexable somehow). Take $\Sigma = \{a\}$, and define $L_i = \{a^n \mid n \in K_i\}$ for all $i \geq 1$. Clearly, $\mathcal{C}^{CorQ}_{FinInf} = (L_i)_{i\geq 1}$ is an indexable class.

Lemma 1. *$\mathcal{C}^{CorQ}_{FinInf}$ is in CorQ.*

Proof. We show that $\langle T_i, F_i, U_i\rangle$ is a triple of definite finite tell-tales of L_i for any index i, where $T_i = L_i$, $l = \max\{n \mid n \in K_i\}$, $F_i = \{a^n \mid n \in \{1,\ldots,l\}\backslash K_i\}$ and $U_i = \{a^{l+1}\}$. Indeed, it is easy to see that T_i, F_i, U_i are finite, and that $T_i \subseteq L_i$, $F_i \subseteq \Sigma^*\backslash L_i$ and $U_i \subseteq \Sigma^*\backslash Pref(L_i)$. Let us take j such that L_j is consistent with the triple $\langle T_i, F_i, U_i\rangle$. Then, $F_i \subseteq \Sigma^*\backslash L_j$ implies $(\{1,\ldots,l\}\backslash K_i) \cap K_j = \emptyset$, and $U_i \subseteq \Sigma^*\backslash Pref(L_j)$ implies $K_j \subseteq \{1,\ldots,l\}$. Putting together these last two results we obtain $K_j \subseteq K_i$, and hence $L_j \subseteq L_i$. But $T_i \subseteq L_j$ implies $L_i \subseteq L_j$. So, $L_j = L_i$ which concludes the proof. □

Lemma 2. *$\mathcal{C}^{CorQ}_{FinInf}$ is not in FinInf.*

Proof. Now, assume that $\mathcal{C}^{CorQ}_{FinInf}$ is in *FinInf*. Cf. Theorem 2, this implies that a pair of definite finite tell-tales $\langle T_i, F_i\rangle$ of L_i is uniformly computable for any index i. Let us fix i, take $l = \max\{i \mid a^i \in F_i\}$, and set j to be the index for which $K_j = K_i \cup \{l+1\}$. Then, L_j is also consistent with the pair $\langle T_i, F_i\rangle$ since $T_i \subseteq L_i \subset L_j$ and $F_i \subseteq \Sigma^*\backslash L_j$ ($F_i \subseteq \Sigma^*\backslash L_i$ and $a^{l+1} \notin F_i$), and hence $L_j = L_i$. We reach a contradiction since $a^{l+1} \in L_j\backslash L_i$. □

This result can be extended to any alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ if we set L_i to be $\{a_1a_2\ldots a_{n-1}a_n^j \mid j \in K_i\}$ for any index i.

As a direct consequence, we obtain that the class *MemQ* is strictly included in *CorQ*. So, CQs are strictly more powerful than MQs, and they cannot be simulated by a finite number of MQs.

4.2 A Model Which Includes $CorQ^{(A)}$

Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class. We have the following theorem.

Theorem 5. *If $\mathcal{C}$ is in $CorQ^{(A)}$, then $\mathcal{C}$ is in ConsvTxt.*

Proof. If $\mathcal{C} = (L_i)_{i\geq 1}$ is in $CorQ^{(A)}$ then, cf. Proposition 2, a triple of definite finite tell-tales $\langle T_i, F_i, U_i\rangle$ of L_i is uniformly computable for any index i.

We show that T_i is a finite tell-tale for L_i. Clearly, T_i is a finite subset of L_i. Let us now take j such that $T_i \subseteq L_j$. We need to prove that L_j is not a proper subset of L_i. Assume by contrary that it is. Then, $L_j \subset L_i$ implies $Pref(L_j) \subseteq Pref(L_i)$, and hence $\Sigma^*\backslash Pref(L_j) \supseteq \Sigma^*\backslash Pref(L_i)$. Keeping in mind that $U_i \subseteq \Sigma^*\backslash Pref(L_i)$ we obtain that $U_i \subseteq \Sigma^*\backslash Pref(L_j)$. Moreover, $F_i \subseteq \Sigma^*\backslash L_i$ and $\Sigma^*\backslash L_j \supseteq \Sigma^*\backslash L_i$ imply $F_i \subseteq \Sigma^*\backslash L_j$. Since L_j is consistent with the triple $\langle T_i, F_i, U_i\rangle$, we have $L_i = L_j$ which contradicts our assumption. So given any index i, a finite tell-tale of L_i is uniformly computable. □

Let us now show that the inclusion is strict. For this, we denote by $I(n)$ the set of all positive integral multiples of n. Let the collection of all finite non-empty sets of prime positive integers be $P_1, P_2, P_3, \ldots$ indexable, for example, in order of increasing $\prod_{p\in P_i} p$. Then, take $\Sigma = \{a\}$, $R_i = \cup_{p\in P_i} I(p)$ and $L_i = \{a^n \mid n \in R_i\}$. Clearly, $\mathcal{C}^{ConsvTxt}_{CorQ} = (L_i)_{i\geq 1}$ is an indexable class.

Lemma 3. $\mathcal{C}_{CorQ}^{ConsvTxt}$ *is in* $ConsvTxt$.

Proof. Let us first notice that $T_i = \{a^p \mid p \in P_i\}$ is a finite tell-tale for L_i. Clearly, T_i is a finite subset of L_i. If we take j such that $L_j \supseteq T_i$, we have $R_j \supseteq P_i$, and furthermore $R_j \supseteq R_i$ and $L_j \supseteq L_i$. Hence, L_j is not a proper subset of L_i. Moreover, P_i is uniformly computable for any index i, and so is T_i. □

Lemma 4. $\mathcal{C}_{CorQ}^{ConsvTxt}$ *is not in* $CorQ^{(A)}$.

Proof. Assume by contrary that $\mathcal{C}_{CorQ}^{ConsvTxt}$ is in $CorQ^{(A)}$. Cf. Proposition 2, a triple of definite finite tell-tales $\langle T_i, F_i, U_i\rangle$ of L_i is uniformly computable, for any index i.

We introduce the following notation: $Num(S) = \{|w| \mid w \in S\}$ for any set $S \subseteq \Sigma^*$. Then $T_i \subseteq L_i$ is equivalent to $Num(T_i) \subseteq R_i$, and $F_i \subseteq \Sigma^* \backslash L_i$ is equivalent to $Num(F_i) \cap R_i = \emptyset$. Finally, $U_i \subseteq \Sigma^* \backslash Pref(L_i)$ implies $U_i = \emptyset$.

Let us now choose a prime number p such that $I(p) \cap Num(F_i) = \emptyset$ and $p \notin P_i$, and take j such that $P_j = P_i \cup \{p\}$. Clearly, $L_i \subset L_j$. We show that L_j is consistent with $\langle T_i, F_i, U_i\rangle$.

Indeed, $T_i \subseteq L_j$ because $T_i \subseteq L_i$ and $L_i \subset L_j$. Also, $Num(F_i) \cap R_j = \emptyset$ because $Num(F_i) \cap R_i = \emptyset$, $Num(F_i) \cap I(p) = \emptyset$ and $R_j = R_i \cup I(p)$. Hence, $F_i \subseteq \Sigma^* \backslash L_j$. The empty set is trivially included in any set, and hence $U_i \subseteq \Sigma^* \backslash Pref(L_j)$.

We found an index j such that L_j is consistent with $\langle T_i, F_i, U_i\rangle$ and $L_j \neq L_i$ which is a contradiction. □

5 Concluding Remarks

Learning formal languages has been a preoccupation of many researchers during time. With every new learning model introduced, many research possibilities were created. Since very recently a new model in the query learning theory has been proposed, namely learning with CQs, we considered that it is worth investigating the particularities of the classes of languages learnable within this setting.

We showed that there exists a method of characterizing these classes using some finite triples, called triples of definite finite tell-tales. With the help of this characterization, we managed to position the class $CorQ$ in the hierarchy formed by other well-known learning models (both Gold-style learning and query learning models).

As we already mentioned, our work was focused on the type of correction introduced in [6]. In the future we would like to consider other types of corrections and to answer to the following question: how is $CorQ$ influenced by the type of correction used? What about combining CQs and a limited number of EQs, or CQs and positive examples? What happens if we restrict to polynomial time learning? Can we construct a language class which is in $CorQ$ and not in $CorQ^{(A)}$? We believe that these are several question which deserve further investigation.

References

1. Gold, E.M.: Language identification in the limit. Information and Control **10** (1967) 447–474
2. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control **45** (1980) 117–135
3. Mukouchi, Y.: Characterization of finite identification. In Jantke, K.P., ed.: Proc. 3rd International Workshop on Analogical and Inductive Inference (AII '92). Volume 642 of Lecture Notes in Computer Science., London, UK, Springer-Verlag (1992) 260–267
4. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75** (1987) 87–106
5. Lange, S., Zilles, S.: Formal language identification: query learning vs. Gold-style learning. Information Processing Letters **91** (2004) 285–292
6. Beccera-Bonache, L., Dediu, A.H., Tîrnăucă, C.: Learning DFA from correction and equivalence queries. In Sakaibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E., eds.: Grammatical Inference: Algorithms and Applications, Proceedings of ICGI '06. Volume 4201 of Lecture Notes in Artificial Intelligence., Berlin, Heidelberg, Springer-Verlag (2006) 281–292
7. Martín-Vide, C., Mitrana, V., Păun, G., eds.: Formal Languages and Applications. Studies in Fuzzyness and Soft Computing 148. Springer-Verlag, Berlin, Heidelberg (2004)
8. Zeugmann, T., Lange, S.: A guided tour across the boundaries of learning recursive languages. In Jantke, K.P., Lange, S., eds.: Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report. Volume 961 of Lecture Notes in Computer Science., London, UK, Springer-Verlag (1995) 190–258
9. Zeugmann, T., Lange, S., Kapur, S.: Characterizations of monotonic and dual monotonic language learning. Information and Computation **120** (1995) 155–173
10. Zeugmann, T.: Inductive inference and language learning. In Cai, J., Cooper, S.B., Li, A., eds.: Proc. 3rd International Conference on Theory and Applications of Models of Computation (TAMC '06). Volume 3959 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (2006) 464–473
11. Lange, S., Zeugmann, T.: Types of monotonic language learning and their characterization. In: Proc. 5th Annual Workshop on Computational Learning Theory (COLT '92), New York, ACM Press (1992) 377–390

Learnable Algorithm on the Continuum

Zhimin Li[1,2] and Xiang Li[1]

[1] Institute of Computer Science, Guizhou University, Guiyang550025, China
[2] School of Mathematics and Computer Science, Guizhou University for Nationalities, Guiyang550025, China

Abstract. Based on limiting recursive function proposed by Gold [2], learnable algorithm on the continuum are defined. We discuss the class of learnable real numbers and learnable real sequence in various ways. In this paper we summarize some attempts to classify the learnably approximable real numbers by the convergence rates of the corresponding computable(or learnable) sequences of rational numbers.

1 Introduction

The theory of limiting recursion has enjoyed a recent resurgence of interest. This stems partly from a wider program of exploring alternative approaches to computation, such as learning theory and proof animation [3]; partly as an idealization of numerical algorithms where functions can be discontinuous [5]; and partly from a desire to use the tools of computation theory to better classify the variety of real recursive functions [7].

In most recent work on the hierarchy of Δ_2^0 real numbers, e.g. [6], is learnable in the sense of computational learning theory since the characteristic function of Δ_2^0-sets are limiting recursive functions. Nevertheless, many authors do not introduce the notion of learnability. Just as in standard computable analysis, a real number is computable if it is an effective limit of computable sequence of rational numbers. Analogy to this, we define learnable real numbers as a limit of a computable sequence of rational numbers $\{r_k\}$, the modulus of convergence is limiting recursive functions. In this paper, we will compare the class of learnable real number with the hierarchy of real numbers proposed by Weihrauch and Zheng[6]. Also, we research learnable real sequences in the framework of computable analysis.

The term "learnable" in this paper means only that there is some limit-recursive bound on the speed of convergence.

2 Limiting Recursion

Let ψ be a partial function on natural numbers. Then $\lim_{n\to\infty} \psi(n)$ is define and equals x if and only if $\exists m \forall n \geq m(\psi(n) \simeq x)$. A partial function $\varphi : N \to N$ is called limiting recursive if there is a total recursive function $F : N^2 \to N$ such that $\varphi(x) \simeq \lim_{n\to\infty} F(x, n)$. Obviously, the class of partial recursive functions is included in the class of limiting recursive functions. As Gold [2] shows, the limiting recursive functions are precisely the functions whose graph is Δ_2^0 in the arithmetical hierarchy, or,

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 408–415, 2007.
© Springer-Verlag Berlin Heidelberg 2007

equivalently, the functions recursive in $\varnothing'$. Clearly, the limiting recursive functions are closed under composition. If a function φ defined by $\varphi(x) \simeq \lim_{n\to\infty} \psi(x,n)$ is total, then φ is limiting recursive whenever ψ is partial recursive. When there is a total limiting recursive function φ such that $y_i = \varphi(i)$, we say that y_i is computable in the limit from i.

Lemma 1 (Waiting Lemma). *Let $a : N \to N$ be a one to one recursive function generating a recursively enumerable nonrecursive set A. let $\omega(n) = \max\{m : a(m) \le n\}$ denote the "waiting time", then function $\omega(n)$ is (partial) limiting recursive and not recursive.*

Proof. Due to [4], there is no recursive function c such that $\omega(n) \le c(n)$. To show $\omega(n)$ be a limiting recursive function, we construct a guess function F as following.

$$F(0,n) = 0;$$
$$F(m+1,n) = F(m,n), \text{ if } a(m+1) > n\ ;$$
$$F(m+1,n) = m+1, \text{ if } a(m+1) \le n\ .$$

$\omega(n) = \lim_{m\to\infty} F(m,n)$. This means we firstly guess $\omega(n) = 0$. Afterwards, if we find $m = 1$ such that $a(1) > n$, we do not change mind. Otherwise, put $\omega(n) = 1$, and so on. If we change mind infinite times, then $\omega(n)$ is undefined.

Definition 1. *[Computable real numbers and real functions]*

- *A sequence $\{r_k\}$ of rational numbers is recursive if there exist three recursive functions α,β,γ from N to N such that $\gamma(k) \neq 0$ for all k and $r_k = (-1)^{\alpha(k)}\frac{\beta(k)}{\gamma(k)}$.*
- *A sequence $\{x_k\}$ of real numbers effectively converges to x if there exists a recursive functions δ such that for all m:*

$$k > \delta(m) \Rightarrow |x_k - x| < 2^{-m}.$$

  δ is called modulus of convergence.
- *A sequence of real number $\{x_n\}$ is computable if there is a computable double sequence of rationals $\{r_{nk}\}$ such that $r_{nk} \to x_n$ as $k \to \infty$, effectively in k and n, e.g. there is a recursive function $e : N \times N \to N$ such that for all n,N:*

$$k \ge e(n,N) \text{ implies } |r_{nk} - x_n| \le 2^{-N}.$$

- *A real number $x \in R$ is computable if there is a recursive sequence of rational numbers which effectively converges to x.*
  Denote this class EC.
- *$f : [a,b] \to R$ is computable if*
  (i) f is sequentially computable, i.e, if $\{x_n\}$ is a computable sequence of real numbers, then $\{f(x_n)\}$ is a computable sequence of reals;
  (ii) f is effectively uniformly continuous, i.e, there is a recursive functions δ such that, for all x, y $\in [a,b]$ and all $k \in N$,

$$|x - y| \le \tfrac{1}{\delta(k)} \Rightarrow |f(x) - f(y)| \le \tfrac{1}{2^k}.$$

  Denote this class CTF.

In effective analysis[4], it is well known that there is general no effective test for deciding computable real number $x = 0$. The below proposition indicate that we can do this by means of a learnable procedure.

Proposition 1. *Let x be a computable real number. The ordered relation of x and 0 can be decided by learnable algorithm.*

Proof. Since x is a computable real number, x is the limit of a computable sequence $\{r_n\}$, with a modulus of convergence δ. Define function

$$\sigma(x) = \begin{cases} 1 & x \in (0, \infty); \\ 0 & x = 0; \\ -1 & x \in (-\infty, 0). \end{cases}$$

Put $\{M_P\}$, where

$$Mp = \begin{cases} 1, & r_{\delta(p)} > 1/2^p; \\ 0, & |r_{\delta(p)}| \leq 1/2^p; \\ -1, & r_{\delta(p)} < -1/2^p. \end{cases}$$

M_p is a recursive sequence. $\sigma(x) = \lim_p M_p$ will be stability at -1, 0 or 1.

3 Learnable Real Numbers

Let us first recall some hierarchies of real numbers proposed by Zheng et al[6]. x_A denotes a real number with a binary expansion A, i.e. there is a set $A \subseteq N$ such that $x = x_A = \sum_{i \in A} 2^{-i}$.

Definition 2. *[Zheng's hierarchy of real numbers[6]]*

- *A real number x_A is called strongly r.e.(sre for short), if A is a r.e. set. Denote this class SRE.*
- *A real number x is called r.e.(re for short), if it is a computable increasing sequence of rational numbers converging to x. Denote LC.*
- *A real number x is called weakly computable iff it is a computable sequence of rational numbers converging to x weakly effectively(i.e. there is a constant c such that $\sum_{s \in N} |x_s - x_{s+1}| \leq c$. Denote WC.*
- *A real number x is called divergence bounded computable if there is a recursive function h and a computable sequence $\{x_s\}$ converging to x, the n-convergence of the sequence $\{x_s\}$ bounded by $h(n)$ for any n. The n-convergence of the sequence $\{x_s\}$ is the maximal natural number m such that $|x_{i_s} - x_{j_s}| \geq 2^{-n}$ for any $s \leq m$. The class of all divergence bounded computable real numbers is denoted by DBC.*
- *A real number $x \in R$ is recursively approximable if it is the limit of computable sequence of rational numbers. Denote this class RA.*

Proposition 2. *[6] $EC \subsetneq SRE \subsetneq LC \subsetneq WC \subsetneq DBC$.*

Proposition 3. *[6] $DBC = CTF(LC) = \{f(x) : x \in LC, f \in CTF\}$, where CTF is a class of computable total functions.*

Next, we propose the definition of Learnable real numbers based on computable information(recursive sequence of rational numbers and computable sequence of real numbers).

Definition 3. *[Learnable real numbers]*

- *A sequence $\{x_k\}$ of real numbers is learnable convergence if there exists a limiting recursive functions δ such that for all m:*

$$k > \delta(m) \Rightarrow |x_k - x| < 2^{-m}.$$

- *A real number $x \in R$ is q-learnable if there is computable sequence of rational numbers which converges learnably.*
  Denote this class C_{qle}.
- *A real number $x \in R$ is r-learnable if there is a computable sequence of real numbers which converges learnably. .*
  Denote this class C_{rle}.

Proposition 4. *The class of Computable real numbers EC $\subsetneq C_{qle}$.*

Proof (sketch). Let $a : N \to N$ be a one to one recursive function generating a recursively enumerable nonrecursive set A. Consider $s_k = \sum_{m=0}^{k} 2^{-a(m)}$. And let $x = \lim_{k\to\infty} s_k$. In [4], we know that $\{s_k\}$ is computable and converges noneffectively, take limiting recursive function $\omega(n)$ as modulus of convergence. Thus x is learnable but not computable.

Proposition 5. *The C_{qle} forms a field.*

Proof (sketch). For any $x, y \in C_{qle}$, there are recursive sequences $\{x_s\}$ and $\{y_s\}$ which converge to x and y respectively, and there limiting recursive functions d_1 and d_2 which are modulus functions of $\{x_s\}$ and $\{y_s\}$ respectively.

$\{x_s + y_s\}$ is recursive sequences which converge to $x + y$, and $\max(d_1, d_2)$ which is modulus functions. Similar we can show that $x - y$, $xy \in C_{qle}$.

Suppose now that $y \neq 0$ and assume that $\forall s \in N(y_s \neq 0)$. Choose a $k \in N$ such that, for all $s \in N$, $|y_s| \geq 2^{-k}$ and $\max\{|y_s|, |y_s|\} \leq 2^k$. Then $|x_s/y_s - x/y| \leq 2^{-(n-1-3k)}$. We take $\max(d_{n+1+3k}, d_{n+1+3k})$ as modulus of convergence.

Corollary 1. *The WC $\subseteq C_{qle}$.*

Proposition 6. *LC $\subsetneq C_{qle}$.*

Proof (sketch). Let $x \in LC$, there is a increasing sequence of rational numbers $\{x_i\}$, $i \in N$, such that $\forall \varepsilon > 0$, $\exists M \in N$, such that $(n, m > M) \Rightarrow |x_n - x_m| < \epsilon$.

For given 2^{-p} , we can choose a limiting recursive function $\delta(p) = \min\{m : \forall n > m \in N, |x_n - x_m| < 2^{-p}\}$ as modulus of converging. That is, We first guess $\delta(p) = 1$, if $x_2 - x_1 < 1/2^{-p}$, we do not change mind; else if $x_2 - x_1 \geq 1/2$, we change the previous guess and put $\delta(p) = 2$(since $\{x_i\}$ is increasing, for all n, $x_n - x_1 \geq 1/2$). we do similar to this step by step. Obviously, this is an effective procedure. Due to the definition of $x \in LC$, $\delta(p)$ must be stability from one points. Therefore, we have $\forall n > m > \delta(p)$, $|x_n - x_m| < 2^{-p})$. So $LC \subseteq C_{qle}$.

Let $A, B \subseteq N$ be r.e.sets such that $A \not\leq_T B$ and $B \not\leq_T A$, then the real number $x_{A \bigoplus \bar{B}}$ is not in LC [1]. Put $A = B \setminus C$ of two r.e.sets B and C. The real number x_A can also be represented as a difference $x_{B \cup C} - x_C = x_A \in C_{qle}$ of two SRE real numbers. Thus, x_A is a witness to this proposition $LC \subsetneq C_{qle}$.

Proposition 7. *$DBC = CTF(LC) \subseteq C_{qle}$.*

Proof. Let $y \in LC$, by proposition 8, we can choose $\{y_s\}$ be an increasing computable sequence of rational numbers from the interval $[0, 1]$ which converges to y and the limiting recursive modulus function $\delta(N)$ such that

$$s > \delta(m) \Rightarrow |y_s - y| < 2^{-m}.$$

Let $f : [0, 1] \to [0, 1]$ be a computable real function with modulus function $e : N \to N$, e.g.for all $x, y \in [0, 1]$ and all $k \in N$,

$$|x - y| \leq 1/e(k) \Rightarrow |f(x) - f(y)| \leq 2^{-k}.$$

By definition 1, the sequence $\{x_s\}$ defined by $x_s = f(y_s)$, is a computable sequence of real numbers.Then there is a computable double sequence $\{r_{st}\}$ of rational number such that $(\forall s, t \in N)(|x_s - r_{st}| \leq 2^{-t})$. Let $r_s := r_{s(s+1)}$. Then $\lim r_s = \lim x_s = \lim f(y_s) = f(y)$. Therefore, there is a recursive sequence of rational number $\{r_s\}$, and a limiting recursive function $\delta(e(s + 1))$ such that

$$k > \delta(e(s + 1)) \Rightarrow |r_k - f(y)| < 2^{-s}.$$

Proposition 8. *$DBC \subsetneq C_{qle}$.*

Proof. We can construct a recursive sequence $\{x_s\}$ of rational numbers which converging learnable to x . The limit x satisfies further, for all $e :=< i, j >\in N$, the following requirements R_e:

$$(1)dom(\varphi_i) = N \text{ and } dom(\alpha_j) = N, \lim_{s\to\infty} \varphi_i(s) = y_i \text{ exits, and}$$
$$(2)\forall n(d(n, \{\varphi_i(s)\}_{s\in N}) \leq \alpha_j(3n + 1)).$$
$$\text{Then } x \neq y_i.$$

Here (φ_i) and (α_j) are effective enumeration of all computable functions $\varphi_i :\subseteq N \to Q$ and $\alpha_j :\subseteq N \to N$, respectively and $d(n, \varphi_i(s))_{s\in N})$ is the n-divergence of the sequence $\{\varphi_i(s)\}_{s\in N})$ Therefore, all requirements R_e together guarantee that x is not in DBC.

The construct is similar to [6]. In fact, we obtain a recursive sequence $\{x_s\}$ of rational numbers which converging h-monotonically to x, e.g. $(\forall s, t \in N)(t > s \Rightarrow h(s)|x - x_s| \geq |x - x_t|)$. Define limiting recursive function $\delta(n) = max\{s : |x - x_s| \geq 1/(2^n h(s))\}$. Therefore, $\{x_s\}$ converges learnably to x. Thus, $x \in C_{qle}$.

Summarizing above propositions , we have the following theorem:

Theorem 1. *$EC \subsetneq LC \subsetneq WC \subsetneq DBC \subsetneq C_{qle}$.*

4 Learnable Sequences of Real Numbers

The idea of sequential computability plays a key role in computable analysis.

M.Yasugi, Y.Tsujii, T.Mori investigated some functions which map a computable sequence of real numbers to a non-computable one which is learnable sequence in the sense of this paper [5].

With this section, we turn to study learnable sequences of real numbers.

Definition 4. *Let $\{x_{nk}\}$ be a double sequence of reals and $\{x_n\}$ a sequence of reals such that, as $k \to \infty$, $x_{nk} \to x_n$ for each n. We say that $x_{nk} \to x_n$ learnably in k and n if there is a limiting recursive function $e : N \times N \to N$ such that for all n, N:*

$$k \geq e(n, N) \Rightarrow |x_{nk} - x_n| \leq 2^{-N}.$$

Definition 5. *A sequence of real numbers $\{x_n\}$ is learnable(as a sequence) if there is a computable double sequence of rationals $\{r_{nk}\}$ such that $r_{nk} \to x_n$ as $k \to \infty$, learnably in k and n.*

Denote the set of all computable sequence of rational numbers *QCS* (Definition 1), real computable sequence *RCS* (Definition 1), learnable sequence *LCS* (Definition 5).

Proposition 9. *$QCS \subsetneq RCS \subsetneq LCS$.*

Proof. Let $a : N \to N$ be a one to one recursive function generating a recursively enumerable nonrecursive set A. Consider the computable double sequence $\{x_{nk}\}$ defined by $x_{nk} = 2^{-m}$, if $n = a(m)$ for some $m \leq k$; and 0, otherwise. Then as $k \to \infty$, $x_{nk} \to x_n$ where $x_n = 2^{-m}$, if $n = a(m)$ for some m; and 0, otherwise. $\{x_n\}$ is in *RCS* but not in *QCS* [4].

Let us take up the integer part function $[x]$, $\{x_n\} \in RCS$ as above. Then $\{y_n\} = \{1 - x_n\} \in RCS$. We have $[y_n] = 0$, if $n = a(m)$ for some m; and 1, otherwise. In [5], $\{[y_n]\}$ acts an example for $RCS \subsetneq LCS$.

Definition 6. *A sequence $\{r_k\}$ of rational numbers is learnable if there exist three limiting recursive functions α, β, γ from N to N such that $\gamma(k) \neq 0$ for all k and $(-1)^{\alpha(k)}\frac{\beta(k)}{\gamma(k)}$. Denote this class LQC.*

Proposition 10. *[6] For any $n \geq 1$ and any real number $x \in R$.*

- $x \in \Delta_{n+1} \Leftrightarrow (\exists f \in \Gamma_Q^{\emptyset^{(n)}})(x = \lim_{i\to\infty} f(i)$ *effectively).*
- $x \in \Delta_{n+2} \Leftrightarrow (\exists f \in \Gamma_Q^{\emptyset^{(n)}})(x = \lim_{i\to\infty} f(i))$.

Where $\Gamma_Q^{\emptyset^{(n)}}$ means the set of all total $\emptyset^{(n)}$-computable functions from N^n(for some $n \in N$)to Q(the set of all rational numbers).

Corollary 2. *$RA = \{x = \lim_{i\to\infty} f(i)$effectively $: f(i) \in LQC\}$.*

Corollary 3. *Define $C_{wle} = \{x = \lim_{i\to\infty} f(i)$learnably $: f(i) \in LQC\}$. Then $RA \subsetneq C_{wle}$.*

Corollary 4. *$\Delta_3^0 = \{x = \lim_{i\to\infty} f(i) : f(i) \in LQC\}$.*

Problem 1. Dose the formula $C_{wle} = \Delta_3^0$ hold? This means that for a sequence in *LQC* whether or not the learnable converging is equivalent to unlimited converging.

However, regarding to a computable real sequence $\{x_n\}$, the next proposition shows that unlimited convergence of a sequence is equivalent to learnable convergence.

Proposition 11. $C_{qle} = C_{rle} = RA$.

Proof. Indeed, it is trivial that C_{qle} is a subset of C_{rle} because any computable sequence of rational numbers is also a computable sequence of real numbers. And it is also trivial that C_{qle} is a subset of *RA*. That *RA* is a subset of C_{qle} is a consequence of the following Lemma *A*.

Lemma *A*: If a computable sequence of rational numbers converges, then it has a modulus of convergence which is limiting recursive.

Proof of the lemma *A*: Let $\{x_n\}$ be a computable sequence of rational numbers that converges to a real number x. We define a computable function $F\colon N \times N \to N$ by:

$$F(n,0) := 0 \text{ for all } n, \text{ and}$$
$$F(n,k+1) := F(n,k) \text{ if } |x_{F(n,k)} - x_{k+1}| \leq 2^{-(n+1)},$$
$$F(n,k+1) := k+1 \text{ otherwise.}$$

Then the limit $\lim_{k\to\infty} F(n,k)$ exists for every n, and the function δ defined by $\delta(n) := \lim_{k\to\infty} F(n,k)$ is limiting recursive and a modulus of convergence of the sequence $\{x_n\}$. This is the end of proof of lemma A.

Finally, we show that C_{rle} is a subset of *RA* and hence of C_{qle}. The following fact is well known and very easy to see.

Lemma B: If $\{x_n\}$ is a computable sequence of real numbers, then there is a computable sequence $\{q_n\}$ of rational numbers with $|x_n - q_n| \leq 2^{-n}$ for all n.

Proof sketch of lemma: Such a sequence $\{q_n\}$ can easily be obtained from a computable double sequence of rational numbers converging effectively to $\{x_n\}$. This s end of proof.

Now we show that C_{rle} is a subset of *RA* and hence of C_{qle}. Let x be an element of C_rle and $\{x_n\}$ be a computable sequence of real numbers converging to x. By Lemma B there is a computable sequence $\{q_n\}$ of rational numbers with $|x_n - q_n| \leq 2^{-n}$ for all n. This sequence converges to x as well. Hence, x is in *RA*.

Lemma *A* can be strengthened easily using Lemma *B* to the following:

Corollary 5. *If a computable sequence of real numbers converges, then it has a modulus of convergence which is limiting recursive.*

Corollary 6. $C_{rle} = CTF(C_{rle})$.

The learnable sequence of real numbers would play a important role in computability of discontinuous functions[5]. However, many questions are left open. Next we list parts of them.

Problem 2. – In proposition 9, we can see that Gauss function [x] map computable sequence of real numbers to a learnable one(we call all functions with this property—L-computability of function). How can we characterize L-computability of function [5], [7].

- Can we give hierarchies of (weakly) learnable real numbers?
- Let $\{f_n\}$ on [0, 1] be any computable sequence of functions such that $f_n(0) < 0$ and $f_n(1) > 0$ for all n. Is there a (learnable) sequence of points c_n in [0,1] with $f_n(c_n) = 0$ for all n? We know that there exists sequence of functions $\{f_n\}$, there is no such computable sequence $\{c_n\}$ [4].
- Let $f \in C^1$ on [0, 1] be any computable functions, $\{x_n\}$ be computable sequence. Is sequence of points $f(x_n)$ (weakly) learnable? We know that there exists functions f, $\{f(x_n)\}$ may not be computable [4].
- Let $f \in C^\infty$ on [0, 1] be any computable functions.
  Is all sequence $\{f^{(n)}(x)\}$ (weakly)learnable? We know that there exists functions $f \in C^\infty$, there is an noncomputable sequence $\{f^{(n)}(0)\}$ [4].

Acknowledgment. The authors would like to express their gratitude to anonymous referees for useful comments which helped improving the paper.

References

1. K.Ambos-Spies, K.Weihrauch and X.Zheng. Weakly computable real numbers. *Journal of complexity.* 16(2000),676-690.
2. E.M.Gold Limiting recursive. *Journal of symolic logic* vol30,(1965),28-48.
3. S.Hayashi, K.Nakata. Towards limit computable mathematics, LNCS 2277(2002)125-144.
4. M.B. Pour-El and J. I. Richards. *Computability in Analysis and Physics*. Springer, 1989.
5. M.Yasugi et.al.Computability aspects of some discont inuous functions, SCMJ, vol 5(2001).
6. X. Zheng, Recursive approximability of real numbers. *Math. Log. Quart.* (2002)473-485.
7. Z. Li, M.Yasugi. Computability of continuous and discontinuous real recursive functions. *Lecture Series on Computer and Computational Sciences(Ieccs2006)*(To appear).

Online Deadline Scheduling with Bounded Energy Efficiency

Joseph Wun-Tat Chan[1], Tak-Wah Lam[⋆,2], Kin-Sum Mak[2], and Prudence W.H. Wong[⋆⋆,3]

[1] Department of Computer Science, King's College London, UK
joseph.chan@kcl.ac.uk
[2] Department of Computer Science, The University of Hong Kong, Hong Kong
{twlam, ksmak}@cs.hku.hk
[3] Department of Computer Science, University of Liverpool, UK
pwong@csc.liv.ac.uk

Abstract. Existing work on scheduling with energy concern has focused on minimizing the energy for completing all jobs or achieving maximum throughput [19,2,7,13,14]. That is, energy usage is a secondary concern when compared to throughput and the schedules targeted may be very poor in energy efficiency. In this paper, we attempt to put energy efficiency as the primary concern and study how to maximize throughput subject to a user-defined threshold of energy efficiency. We first show that all deterministic online algorithms have a competitive ratio at least Δ, where Δ is the max-min ratio of job size. Nevertheless, allowing the online algorithm to have a slightly poorer energy efficiency leads to constant (i.e., independent of Δ) competitive online algorithm. On the other hand, using randomization, we can reduce the competitive ratio to $O(\log \Delta)$ without relaxing the efficiency threshold. Finally we consider a special case where no jobs are "demanding" and give a deterministic online algorithm with constant competitive ratio for this case.

1 Introduction

Processor scheduling is a classical optimization problem. As the proliferation of mobile computing devices, energy usage has become an important performance measure for processor scheduling. *Dynamic voltage scaling* [9,15,18] is a technology that enables the reduction in energy usage. It allows a processor to run in variable speed; the rate of energy consumption of a processor running at speed s is believed to be s^α where $\alpha \geq 2$ [5]. Note that a processor running at speed s can do s units of work in one unit of time. To process a job with w units of work at speed s, a processor consumes $s^\alpha w/s = s^{\alpha-1}w$ units of energy. In other words, it is more energy efficient to schedule a job at a low speed whenever possible.

In the literature, the study of energy-efficient scheduling was mainly in the context of deadline scheduling [19,2,7,13,14]. Given a processor where jobs arrive

⋆ The research is supported partly by RGC Grant HKU-7140/06E.
⋆⋆ The research is supported partly by EPSRC Grant EP/E028276/1.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 416–427, 2007.
© Springer-Verlag Berlin Heidelberg 2007

at unpredictable times with arbitrary work and deadline requirement, the aim is to design an (online) schedule that maximizes the throughput, which is the total work of the jobs completed by their deadlines. An algorithm A is said to be c-competitive, where $c \geq 1$, if for any job sequence, A produces a schedule with a throughput at least $1/c$ of the best offline schedule.

Existing work on scheduling with energy concern has focused on minimizing the energy for completing all jobs or achieving maximum throughput [19,2,7,13, 14]. That is, energy usage is a secondary concern when compared to throughput and the schedules targeted may be very poor in energy efficiency. In this paper, we attempt to put energy efficiency as the primary concern and study how to maximize throughput subject to a user-defined threshold of energy efficiency. We define the *energy efficiency* of a schedule to be the total amount of work completed in time divided by the total energy usage. The range of efficiency is $[0, \infty)$. For example, assuming the processor completes every job (on time) it works on, using unit-speed gives an efficiency of 1; 0.5-speed gives an efficiency of 4 (assuming $\alpha = 3$); 2-speed gives an efficiency of 0.25; running the processor at high speed would give an efficiency approaching zero. We refer to an efficiency of 1 the "ideal" efficiency. We assume that the user has a preference on energy efficiency and would not accept schedules with energy efficiency below a certain threshold E. Given a job sequence and an *efficiency threshold* E, we aim at finding an energy-efficient schedule (i.e., with energy efficiency at least E) that maximizes the throughput. In this paper we investigate online algorithms that produce energy-efficient schedules with throughput competitive to the optimal schedule which has the maximum throughput while maintaining an energy efficiency at least E.

Previous work. There has been a number of work when throughput is the primary concern and energy usage the secondary. Yao et al. [19] considered the case where a processor can run at any speed $s \geq 0$. They gave two online algorithms AVR and OA, and showed that AVR is $2^\alpha \alpha^\alpha$-competitive on energy usage (against the optimal offline schedule that uses the minimum amount of energy to complete all jobs). OA was later proved, by Bansal et al. [2], to be α^α-competitive. Bansal et al. [2] further improved the result with a new algorithm that is $2(\alpha/(\alpha-1))^\alpha e^\alpha$-competitive. Chan et al. [7] considered the case where the speed of a processor is upper bounded by a constant T, i.e., the processor can run at any speed $s \in [0, T]$. The optimal schedule is the one that maximizes the throughput and minimizes the energy usage among all schedules with the maximum throughput. They proposed an online algorithm FSA(OAT) that is 14-competitive on throughput and $(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive on energy with the optimal schedule. On the other hand, Li et al. [13] have considered structured jobs and shown that AVR has a better performance. Very recently, scheduling on a processor with a fixed number of discrete speed levels has also attracted some attention [12, 14, 7].

Our contributions. Assume that a processor can run at any speed $s \geq 0$, we study how to maximize throughput subject to a user defined threshold of energy

efficiency. We first show that no deterministic online algorithm can achieve a constant competitive ratio on throughput while guarantee an energy efficiency at least E. Precisely, let Δ be the max-min ratio of work of the jobs. All deterministic online algorithms have a competitive ratio at least Δ. Then, we study the problem in three different directions.

1. **Relaxed energy efficiency threshold:** Assume that the energy efficiency threshold of the online algorithm is relaxed to $E/(1+\epsilon)$ for some constant $\epsilon > 0$, while that for the optimal offline algorithm remains E. We give a deterministic online algorithm with competitive ratio $2 + \frac{3}{(1+\epsilon)^{\frac{1}{\alpha-1}}-1}$. For example if $\alpha = 2$, the algorithm is $2 + 3/\epsilon$-competitive. Note that we have a lower bound that no deterministic online algorithm can achieve a competitive ratio of 1 even with relaxed energy efficiency threshold. However with limited pages, the details are omitted.
2. **Randomized algorithm:** We devise a randomized online algorithm with a competitive ratio of $O(\log \Delta)$. Our algorithm is adapted from the randomized algorithm of Goldman et al. [8]. They showed that their algorithm is $6(\log \Delta + 1)$-competitive for a fixed-speed processor. The adapted algorithm works for a variable speed processor, which produces schedules with energy efficiency at least E and is still $O(\log \Delta)$-competitive.
3. **No demanding jobs:** We consider a special case that no job is *demanding*. A job is demanding if it could not be completed (by its deadline) by a processor running at unit speed. For this special case and with $E < 1$, we give a deterministic online algorithm with competitive ratio $2 + \frac{3}{1-E^{\frac{1}{\alpha-1}}}$. We can see that a smaller E would give a smaller competitive ratio. For $E \geq 1$, the lower bound of Δ for the general input also applies to this case.

Remarks. The literature also contains results on other interesting aspects of scheduling with energy concern [11]. Irani et al. [10] extended the result on AVR [19] to a setting where the processor has a sleep state, and showed that the extension increases the competitive ratio on energy by only a constant factor. On the other hand, Pruhs et al. [16] have studied the offline problem of minimizing total flow time subject to a fixed amount of energy, while Albers and Fujiwara [1] and Bansal et al. [4] have studied the online problem of minimizing a cost consisting of the energy usage and the total flow time. Furthermore, the offline problem of minimizing the makespan subject to a fixed amount of energy has been studied in [6, 17]. Another practical concern is the maximum temperature of the processor as the temperature is related to energy usage. Several interesting results have been reported in [2, 3].

Organization of paper. The rest of the paper is organized as follows. In Section 2, we give the problem definition and a general lower bound. In Section 3, we show that allowing the online algorithm to have a slightly poorer energy efficiency leads to a constant competitive online algorithm. In Section 4, we present the randomized algorithm and analyze its performance. Finally, in Section 5, we study the special case with no demanding jobs.

2 Preliminaries

For any job J, we denote the release time, work and deadline of J as $r(J)$, $w(J)$, and $d(J)$, respectively. We assume that all jobs satisfy the property $w(J) \leq (d(J) - r(J))/E^{\frac{1}{\alpha-1}}$, otherwise no algorithm can complete this individual job with energy efficiency at least E. The span of J, denoted as $\rho(J)$, is the time interval $[r(J), d(J)]$. For any set of jobs L, let $w(L)$ denote the total work of all jobs in L. To ease our discussion, we assume that an algorithm will never process a job after missing its deadline, and whenever we say that a job is completed, it is always meant to be completed by the deadline. We assume preemption is allowed, and a preempted job can be resumed at the point of preemption. Given an efficiency threshold E, the optimal schedule is one whose energy efficiency is at least E and the throughput is the maximum among these schedules. An online algorithm is said to be c-competitive if for any job sequence, the schedule produced has an energy efficiency at least E and throughput at least $1/c$ of the optimal schedule.

Next we give a lower bound on the competitive ratio on throughput.

Theorem 1. *To maintain a user defined energy efficiency E, any deterministic online algorithm is at least Δ-competitive on throughput, where Δ is the max-min ratio of the work of jobs.*

Proof. Consider an example with two jobs. The first job J_1 is released at time 0 with $d(J_1) = 2$ and $w(J_1) = 2/E^{\frac{1}{\alpha-1}}$. Any online algorithm must schedule this job immediately at its arrival with speed $1/E^{\frac{1}{\alpha-1}}$ in order to complete the job in time and satisfy the efficiency constraint. Otherwise, the adversary would stop releasing jobs. Then, at time 1, J_2 is released, with $d(J_2) = 2\Delta + 1$ and $w(J_2) = 2\Delta/E^{\frac{1}{\alpha-1}}$. The online algorithm cannot further schedule J_2 due to the efficiency constraint. However, the optimal schedule only schedules J_2, and hence the competitive ratio is Δ. □

3 Relaxed Energy Efficiency Threshold

In this section we study the case where the energy efficiency threshold of the online algorithm is relaxed to $E/(1+\epsilon)$ for some constant $\epsilon > 0$, while that for the optimal offline algorithm remains E. We first present an algorithm named Efficiency, and show some properties of the optimal offline algorithm. Then we analyze the performance of this online algorithm.

3.1 The Online Algorithm

Efficiency$_E$ takes a parameter E and schedules jobs with speed $1/E^{\frac{1}{\alpha-1}}$. The selection of jobs to run depends on a notion called $wait(\cdot)$ defined as follows. For any job J, we denote by $lst(J)$ the latest start time of J such that J can still be finished by Efficiency$_E$, i.e., $lst(J) = d(J) - w(J)E^{\frac{1}{\alpha-1}}$. At any time t, we define the allowed waiting time of the job J, $wait(J, t) = lst(J) - t$.

EFFICIENCY$_E$ works as follows: Whenever the processor is idle, EFFICIENCY$_E$ schedules the job with the smallest $wait(J,t) \geq 0$ with speed $1/E^{\frac{1}{\alpha-1}}$.

Lemma 1. *The energy efficiency of the schedule given by* EFFICIENCY$_E$ *is* E.

Proof. EFFICIENCY$_E$ always runs at speed $1/E^{\frac{1}{\alpha-1}}$ and it completes a job whenever the job is scheduled. Let T be the total time EFFICIENCY$_E$ works on I. Then the resulting efficiency is $\frac{T/E^{\frac{1}{\alpha-1}}}{T/E^{\frac{\alpha}{\alpha-1}}} = E$. □

We analyze the relation between the throughput of EFFICIENCY$_E$ and the optimal offline algorithm OPT. Consider any job sequence I. Let $A \subseteq I$ be the set of jobs scheduled by EFFICIENCY$_E$ and let $N = I - A$. Consider the union of spans of all jobs in N, i.e., $\bigcup_{X \in N} \rho(X)$. Let $\ell = |\bigcup_{X \in N} \rho(X)|$. We show below that the competitive ratio of EFFICIENCY$_E$ depends on the ratio $\ell/w(A)$. Consider the optimal schedule for I. We denote by S, M, F the amount of work finished by OPT using slow, medium and fast speed, respectively. Precisely, S, M, F denote the amount of work OPT finishes with speed $\leq 1/E^{\frac{1}{\alpha-1}}$, $\leq 2/E^{\frac{1}{\alpha-1}}$, and $> 2/E^{\frac{1}{\alpha-1}}$, respectively. Note that the total work finished by OPT equals $M+F$. The following lemma gives the relation between S, M, F and $w(A)$.

Lemma 2. *(i)* $F \leq S$*; (ii)* $M \leq w(A) + 2\ell/E^{\frac{1}{\alpha-1}}$*; (iii)* $S \leq w(A) + \ell/E^{\frac{1}{\alpha-1}}$.

Proof. (i) Suppose the periods OPT running with speed $> 2/E^{\frac{1}{\alpha-1}}$ have durations $t_1, t_2, \ldots, t_n$ and speed $2k_1/E^{\frac{1}{\alpha-1}}, 2k_2/E^{\frac{1}{\alpha-1}}, \ldots, 2k_n/E^{\frac{1}{\alpha-1}}$, respectively, for some $k_i > 1$. Since the efficiency of the resulting schedule of OPT must be at least E, we have

$$\frac{S + \sum_{1 \leq i \leq n} 2k_i t_i / E^{\frac{1}{\alpha-1}}}{\sum_{1 \leq i \leq n} 2^\alpha k_i^\alpha t_i / E^{\frac{\alpha}{\alpha-1}}} \geq \text{ Energy efficiency of OPT } \geq E\,.$$

Thus, we have $\sum_{1 \leq i \leq n} (2^{\alpha-1} k_i^{\alpha-1} - 1)(2k_i t_i / E^{\frac{1}{\alpha-1}}) \leq S$. Since $(2^{\alpha-1} k_i^{\alpha-1} - 1) \geq 1$, we have $F = \sum_{1 \leq i \leq n} (2k_i t_i / E^{\frac{1}{\alpha-1}}) \leq S$.

(ii) Using speed at most $2/E^{\frac{1}{\alpha-1}}$, OPT can at most complete all jobs in A and a work of $2\ell/E^{\frac{1}{\alpha-1}}$ during the period of length ℓ. Therefore, $M \leq w(A) + 2\ell/E^{\frac{1}{\alpha-1}}$.

(iii) Similarly, using speed at most $1/E^{\frac{1}{\alpha-1}}$, OPT can at most complete all jobs in A and a work of $\ell/E^{\frac{1}{\alpha-1}}$ during the period of length ℓ. Therefore, $S \leq w(A) + \ell/E^{\frac{1}{\alpha-1}}$. □

Lemma 3. $\frac{OPT}{w(A)} \leq 2 + \frac{3\ell}{w(A)} \cdot \frac{1}{E^{\frac{1}{\alpha-1}}}$.

Proof. $OPT = M + F \leq M + S$, the inequality is due to Lemma 2 (i). Together with Lemma 2 (ii) and (iii), the lemma follows. □

3.2 Performance of EFFICIENCY$_{E/(1+\epsilon)}$

We consider the algorithm EFFICIENCY$_{E/(1+\epsilon)}$ and bound the ratio $\frac{\ell}{w(A)}$, thereby, show that EFFICIENCY$_{E/(1+\epsilon)}$ is constant competitive.

For any $X \in N$, X is not scheduled by EFFICIENCY, implying that EFFICIENCY has scheduled some other jobs during the span $\rho(X)$. The following lemma gives a lower bound on $w(r(X), b)$, which denotes the amount of work finished by EFFICIENCY during the interval $[r(X), b] \subseteq \rho(X)$.

Lemma 4. *For any $X \in N$, $w(r(X), t) > ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)(t - r(X))/E^{\frac{1}{\alpha-1}}$, for any $r(X) < t \leq d(X)$.*

Proof. From $r(X)$ to $lst(X)$, EFFICIENCY must not be idle. Since the processor runs at a speed $((1+\epsilon)/E)^{\frac{1}{\alpha-1}}$, at any time $r(X) < t \leq lst(X)$, it has completed a work of $((1+\epsilon)/E)^{\frac{1}{\alpha-1}}(t - r(X)) > ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)(t - r(X))/E^{\frac{1}{\alpha-1}}$.

Furthermore, we have

$$\begin{aligned} w(r(X), lst(X)) &= ((1+\epsilon)/E)^{\frac{1}{\alpha-1}}(lst(X) - r(X)) \\ &= ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)|\rho(X)|/E^{\frac{1}{\alpha-1}} + |\rho(X)|/E^{\frac{1}{\alpha-1}} \\ &\quad -(d(X) - lst(X))((1+\epsilon)/E)^{\frac{1}{\alpha-1}} \\ &= ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)|\rho(X)|/E^{\frac{1}{\alpha-1}} + |\rho(X)|/E^{\frac{1}{\alpha-1}} - w(X). \end{aligned}$$

By the assumption of the work of a job, we have $w(X) \leq |\rho(X)|/E^{\frac{1}{\alpha-1}}$. Hence, $w(r(X), lst(X)) \geq ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)|\rho(X)|/E^{\frac{1}{\alpha-1}} \geq ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)(t - r(X))/E^{\frac{1}{\alpha-1}}$ for $r(X) < t \leq d(X)$. Since X is not scheduled by EFFICIENCY, another job must be scheduled at $lst(X)$ and last until a time after $lst(X)$ and by EFFICIENCY it must complete, and thus $w(r(X), t) > w(r(X), lst(X))$ for any $t > lst(X)$. Thus, we have $w(r(X), t) > ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)(t - r(X))/E^{\frac{1}{\alpha-1}}$ for $lst(X) < t \leq d(X)$. This completes the proof of the lemma. □

Based on Lemma 4, we can bound $w(A)/\ell$ as follows.

Lemma 5. $w(A) > ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)\ell/E^{\frac{1}{\alpha-1}}$.

Proof. Let M be a minimal subset of N such that $\bigcup_{X \in M} \rho(X) = \bigcup_{X \in N} \rho(X)$, i.e., M induces the same union of spans as N does. Note that no job in M has its span being a sub-interval of any job in M and no three jobs in M have their spans overlapping at a common time. A consequence is that we can arrange the jobs in M such that the arrival times, as well as the deadlines, of the jobs are strictly increasing. Furthermore, the span of the jobs may form several disjoint continuous intervals $\rho_1, \rho_2, \cdots, \rho_k$, for some k. To prove the lemma, it suffices to show that $w(\rho_i) > ((1+\epsilon)^{\frac{1}{\alpha-1}} - 1)|\rho_i|/E^{\frac{1}{\alpha-1}}$ for all $1 \leq i \leq k$.

Suppose the subset of jobs in M with span in ρ_i is $\{X_1, X_2, \cdots, X_m\}$ such that $r(X_j) < r(X_{j+1}) < d(X_j) < d(X_{j+1})$ for all $1 \leq j \leq m-1$. We are going to show by induction on j that, for any $r(X_1) < t \leq d(X_j)$, $w(r(X_1), t) >$

$((1+\epsilon)^{\frac{1}{\alpha-1}}-1)(t-r(X_1))/E^{\frac{1}{\alpha-1}}$. By Lemma 4, the base case is true. Assume it is true up to some j. Consider X_{j+1}. By induction and $r(X_{j+1}) \le d(X_j)$, we have $w(r(X_1),t) > ((1+\epsilon)^{\frac{1}{\alpha-1}}-1)(t-r(X_1))/E^{\frac{1}{\alpha-1}}$ for $r(X_1) < t \le r(X_{j+1})$. Using Lemma 4 for X_{j+1} and $r(X_{j+1}) < t \le d(X_{j+1})$, we have, for $r(X_1) < t \le d(X_{j+1})$, $w(r(X_1),t) > ((1+\epsilon)^{\frac{1}{\alpha-1}}-1)(t-r(X_1))/E^{\frac{1}{\alpha-1}}$. Therefore, the hypothesis is true for $j+1$ and the lemma follows. □

With Lemmas 3 and 5 , we have a constant competitive ratio for $\text{EFFICIENCY}_{E/(1+\epsilon)}$.

Theorem 2. $\text{EFFICIENCY}_{E/(1+\epsilon)}$ *is* $2+\frac{3}{(1+\epsilon)^{\frac{1}{\alpha-1}}-1}$*-competitive on throughput when the energy efficiency threshold is relaxed to* $E/(1+\epsilon)$.

4 Randomized Algorithm

As we have seen in Theorem 1, any deterministic online algorithm is at least Δ-competitive on throughput, where Δ is the max-min ratio of the work of jobs. In this section we show that randomization helps to overcome this barrier. By adapting the randomized algorithm of Goldman et al. [8], we can yield a competitive ratio of $O(\lceil \log \Delta \rceil)$ on throughput while maintaining an energy efficiency of E.

This section proceeds as follows. We first present the adapted randomized algorithm called RAN and state the results of Goldman et al. which also apply to RAN when the processor speed is fixed. Then, in Section 4.1, we analyze the competitive ratio of RAN when the processor speed can vary. More interestingly, we compare RAN with both the optimal non-preemptive and preemptive energy-efficient schedules (i.e., with energy efficiency at least E). A crucial lemma we used in the analysis would be proved in Section 4.2.

The algorithm of Goldman et al. schedules jobs in a processor with fixed unit speed. Each job is either (1) scheduled, (2) virtually scheduled, or (3) rejected. The job only runs if it is scheduled. A *virtually scheduled* job J does not run itself, but prevents any job of work less than $2w(J)$ from running. Thus, virtually scheduling a job J holds the processor for a longer job with a short wait time that may arrive during the interval when J is virtually scheduled. Assume the work of jobs is in $[1,\Delta]$. A number of queues are maintained each for jobs with different work. The queue for jobs of work in $(2^\ell, 2^{\ell+1}]$ is denoted by Q_ℓ. The adapted randomized algorithm RAN is given as follows. We assume that the processor runs at speed $1/E^{\frac{1}{\alpha-1}}$.

When a job J arrives
Suppose the work of J, i.e., $w(J)$, is in $(2^{\ell-1}, 2^\ell]$.
If the system is idle, or if another job J' is virtually scheduled where $w(J') \le 2^{\ell-1}$,
  With probability $\frac{1}{\lceil \log \Delta \rceil + 1 - \ell}$ we schedule J,
  otherwise virtually schedule J.
Else place J in Q_ℓ.

When a scheduled or virtually scheduled job finishes at time t

Let Q_ℓ be the non-empty queue with the largest ℓ possible.
Let J be the job having the smallest $wait(J,t)$[1] ≥ 0 in Q_ℓ.
Remove J from Q_ℓ.
With probability $\frac{1}{\lceil \log \Delta \rceil + 1 - \ell}$ we schedule J,
otherwise virtually schedule J.

In the followings, we state a property of RAN and the performance of RAN against the optimal non-preemptive schedules at fixed speeds. Since RAN always schedules jobs at speed $1/E^{\frac{1}{\alpha-1}}$ and it completes each job it ever schedules, it always gives schedules with energy efficiency exactly E.

Fact 1. RAN *always gives schedules with energy efficiency exactly* E.

Let RAN_s be the expected throughput of RAN and OPT_s^n (resp. OPT_s^p) the throughput of the optimal non-preemptive (resp. preemptive) schedule for a processor with a fixed speed s. Goldman et al. [8] proved that OPT_s^n is at most $6(\lceil \log \Delta \rceil + 1)$ times of RAN_s.

Theorem 3 ([8]). $6(\lceil \log \Delta \rceil + 1)RAN_s \geq OPT_s^n$.

By slightly amending the analysis of Goldman et al., we can prove that the throughput of the optimal non-preemptive schedule with a fixed speed twice that of RAN, i.e., OPT_{2s}^n, is at most $8(\lceil \log \Delta \rceil + 1)$ times of RAN_s.

Theorem 4. $8(\lceil \log \Delta \rceil + 1)RAN_s \geq OPT_{2s}^n$.

4.1 Performance of the Randomized Algorithm

Although RAN always gives non-preemptive schedules, we analyze RAN in both non-preemptive and preemptive models, i.e., against the optimal non-preemptive and the optimal preemptive schedules with energy efficiency at least E.

Non-preemptive model. For a processor with variable speed, we show that the throughput of the optimal non-preemptive energy-efficient schedule is at most $14(\lceil \log \Delta \rceil + 1)$ times the expected throughput of the schedules produced by RAN. Recall from Section 3.1 that, in the optimal non-preemptive energy-efficient schedule, S and M denote the amount of work finished with speed $\leq 1/E^{\frac{1}{\alpha-1}}$ and $\leq 2/E^{\frac{1}{\alpha-1}}$, respectively. (Note that the results in Section 3.1 applies to both non-preemptive and preemptive models.) Moreover, by Lemma 2, the throughput of the optimal non-preemptive energy-efficient schedule is at most $S + M$. Obviously, S and M are no more than the maximum throughput using fixed speeds $1/E^{\frac{1}{\alpha-1}}$ and $2/E^{\frac{1}{\alpha-1}}$, respectively, with no energy concern. Thus, $S \leq OPT_s^n$ and $M \leq OPT_{2s}^n$ where $s = 1/E^{\frac{1}{\alpha-1}}$. Therefore, by Theorems 3 and 4, we have $S \leq 6(\lceil \log \Delta \rceil + 1)RAN_s$ and $M \leq 8(\lceil \log \Delta \rceil + 1)RAN_s$, and hence the throughput of the optimal non-preemptive energy-efficient schedule is at most $14(\lceil \log \Delta \rceil + 1)RAN_s$. We have the following theorem.

[1] Recall that $wait(J,t)$ is defined in Section 3.1 to be $lst(J) - t$ where $lst(J)$ is the latest start time of the job J such that J can still be completed on time.

Theorem 5. RAN *is* $14(\lceil \log \Delta \rceil + 1)$*-competitive in the non-preemptive model.*

Preemptive model. For a processor with variable speed, we show that the throughput of the optimal preemptive energy-efficient schedule is at most $70(\lceil \log \Delta \rceil + 1)$ times the expected throughput of the schedules produced by RAN. Similar to the non-preemptive case, in the optimal preemptive energy-efficient schedule, M and S denote the amount of work finished with speed $\leq 1/E^{\frac{1}{\alpha-1}}$ and $\leq 2/E^{\frac{1}{\alpha-1}}$, respectively. We can establish the relations $S \leq OPT_s^p$ and $M \leq OPT_{2s}^p$ where $s = 1/E^{\frac{1}{\alpha-1}}$ (recall that the superscript p refers to the optimal preemptive schedule).

The key to obtain the claimed competitive ratio of RAN in the preemptive model is to relate OPT_s^p and OPT_s^n, i.e., the throughput of the optimal preemptive and non-preemptive schedules of a processor at fixed speed s. We can show that $5OPT_s^n \geq OPT_s^p$ and we give the analysis in next section. We continue to derive the competitive ratio of RAN as follows. By Lemma 2, the throughput of the optimal preemptive energy-efficient schedule is at most $S + M$. Together with the relations $S \leq OPT_s^p$ and $M \leq OPT_{2s}^p$ where $s = 1/E^{\frac{1}{\alpha-1}}$, and $5OPT_{s'}^n \geq OPT_{s'}^p$ for any fixed speed s', we have the throughput of the optimal preemptive energy-efficient schedule at most $5OPT_s^n + 5OPT_{2s}^n$. Further applying Theorems 3 and 4, we can bound this optimal throughput by $70(\lceil \log \Delta \rceil + 1)RAN_s$. Hence, we have the following theorem.

Theorem 6. RAN *is* $70(\lceil \log \Delta \rceil + 1)$*-competitive in the preemptive model.*

4.2 Comparing Non-preemptive and Preemptive Optimal Schedules at Fixed Speed

We prove the claim $5OPT_s^n \geq OPT_s^p$ we used in the previous section, i.e., we show that, for any job sequence, with a processor at a fixed speed the throughput of the optimal preemptive schedule is at most 5 times that of the optimal non-preemptive schedule. In fact, we achieve this ratio by comparing the optimal preemptive schedule with a non-preemptive schedule $\mathcal{S}$ (not necessarily optimal) we constructed below.

Without loss of generality, we assume that the processor is at unit speed, i.e., $s = 1$. An (offline) non-preemptive schedule is constructed as follows. The idea is to mark a potential job that could be run in $\mathcal{S}$. At any time, at most one job is marked. A marked job may be unmarked, and further be marked again later. When a job has been marked continuously for a time period equal its work, we put the job in the non-preemptive schedule. The details of construction are given below.

When a job J arrives
If there is no marked job, or if another job J' is marked with $w(J') \leq w(J)/2$, unmark J' and mark J.

When a job J is marked continuously for $w(J)$ units of time
Unmark J and put J in the non-preemptive schedule.

Mark the job J' with the largest $w(J')$ and $lst(J')$ not passed yet, if one exists.

The starting and finishing times of a job J in the schedule are the starting and ending times that the job is marked continuously for $w(J)$ units of time. It is clear that the schedule obtained above is non-preemptive since every job J in the schedule runs continuously for $w(J)$ units of time.

Before we compare the throughput of the non-preemptive schedule and the optimal preemptive schedule, we give some definitions and identify some properties of the non-preemptive schedule. Recall that $\mathcal{S}$ denotes the set of jobs in the non-preemptive schedule. For each job $J \in \mathcal{S}$, let $J_1, J_2, \ldots, J_k$ $(= J)$ be the sequence of jobs such that J_i is marked continuously for less than $w(J_i)$ time units and then unmarked because of the arrival and marking of J_{i+1} where $w(J_i) \leq w(J_{i+1})/2$, and finally J_k is marked continuously for $w(J_k)$ time units. Define an interval $I(J) = [a, b + 2w(J)]$ where a is the time that J_1 is initially marked in this sequence and b is the time that J_k $(= J)$ has been marked for $w(J_k)$ time units. We give some properties of the intervals as follows.

Fact 2. *For any time t if there is a job being marked at t, then $t \in I(J)$ for some job $J \in \mathcal{S}$.*

Lemma 6. *For any job $J \in \mathcal{S}$, the length of the interval $I(J)$ is at most $4w(J)$.*

Proof. Consider the sequence of jobs $J_1, J_2, \ldots, J_k$ $(= J)$ defined by J. We have $I(J) = [a, b + 2w(J)]$ where J_1 is initially marked at time a in this sequence and b is the time that J_k $(= J)$ has been marked for $w(J_k)$ $(= w(J))$ time units. By the definition, $b - a \leq w(J) + w(J)/2 + w(J)/2^2 + \ldots \leq 2w(J)$. Hence, we have $I(J) = b + 2w(J) - a \leq 4w(J)$. □

In the following lemma, we prove that the total work of all jobs in the optimal preemptive schedule but not in $\mathcal{S}$ is at most the sum of lengths of $I(J)$ for all jobs $J \in \mathcal{S}$. By this and the previous lemma, we can easily bound the throughput of the optimal preemptive schedule, as shown in Theorem 7.

Lemma 7. *The total work of all jobs in the optimal preemptive schedule but not in $\mathcal{S}$ is at most the sum of length of $I(J)$ for all jobs $J \in \mathcal{S}$.*

Proof. We prove the lemma by showing for all $J' \notin \mathcal{S}$ and for all $r(J') \leq t \leq d(J')$ that, $t \in I(J)$ for some job $J \in \mathcal{S}$. In other words, all the time that J' can be scheduled falls in the union of interval $I(J)$ for all $J \in \mathcal{S}$. Therefore, the total work of all jobs J' in the optimal preemptive schedule but not in $\mathcal{S}$ is bounded by the total length of the interval $I(J)$ for all $J \in \mathcal{S}$.

Let J' be a job in the optimal preemptive schedule but not in $\mathcal{S}$. First, we show for any time $r(J') \leq t \leq lst(J')$ that, $t \in I(J)$ for some $J \in \mathcal{S}$. Since $J' \notin \mathcal{S}$, there must be some job being marked during $[r(J'), lst(J')]$, as otherwise J' can be marked. Thus, by Fact 2, for any $r(J') \leq t \leq lst(J')$, we have $t \in I(J)$ for some job $J \in \mathcal{S}$.

Then, we also show for any time $lst(J') < t \leq d(J')$ that, $t \in I(J)$ for some $J \in \mathcal{S}$. Suppose $lst(J') \in I(X) = [a, b + 2w(X)]$ for the job $X \in \mathcal{S}$ where $I(X)$ has the maximum value of a. We claim that (i) $2w(X) > w(J')$ and (ii) $a \leq lst(J') \leq b$. Hence, for $lst(J') < t \leq d(J')$, we have $t \in [a, b + 2w(X)]$ as $d(J') - lst(J') = w(J') < 2w(X)$. Claim (i) is true because otherwise J' will be marked instead of X. Claim (ii) is true because if $b < lst(J') \leq b + 2w(X)$, J' or other jobs can be marked at $lst(J')$ and hence there exists another interval $I(Z) = [a', b' + 2w(Z)]$ that includes $lst(J')$ and $a' > a$, which is a contradiction. In conclusion, we have for any time $r(J') \leq t \leq d(J')$ that, $t \in I(J)$ for some $J \in \mathcal{S}$, and the lemma follows. □

Theorem 7. *The throughput of the optimal preemptive schedule is at most* 5 *times that of the optimal non-preemptive schedule, i.e.,* $5OPT_s^n \geq OPT_s^p$*, for a processor at a fixed speed* s *where* s *can be any constant greater than* 0.

Proof. By Lemma 7, the total work of jobs in the optimal preemptive schedule but not in $\mathcal{S}$ is at most the sum of length of $I(J)$ for all jobs $J \in \mathcal{S}$, which is at most 4 times the throughput of the non-preemptive schedule according to Lemma 6. Therefore, the throughput of the optimal preemptive schedule is at most 5 times the throughput of the non-preemptive schedule. As the throughput of the non-preemptive schedule must be at most that of the optimal non-preemptive schedule, the theorem follows. □

5 Without Demanding Jobs

We study the special case in which there are no demanding jobs that cannot be finished before deadline using unit speed. In other words, for every job J, $w(J) \leq d(J) - r(J)$. Because of the limited pages, we only the state the lemmas and theorems in this section without proofs. First we give a lower bound for this special case with no demanding jobs. The proof of which is similar to that of Theorem 1.

Theorem 8. *Without demanding jobs, if* $E \geq 1$*, any deterministic online algorithm is at least* Δ*-competitive on throughput, where* Δ *is the max-min ratio of the work of the jobs.*

Therefore, we only consider the case where $E < 1$ in the rest of this section. Without demanding jobs, we use the algorithm EFFICIENCY$_E$ (as defined in Section 3), which by Lemma 1, gives schedules with energy efficiency at least E. By Lemma 3, we only need to bound the ratio $\ell/w(A)$ to obtain the competitive ratio of EFFICIENCY$_E$. Recall the definition of I, A, N in Section 3. Similar to Lemma 4 and since jobs are not demanding, we can derive the following lemma.

Lemma 8. *For any* $X \in N$, $w(r(X), t) > (1/E^{\frac{1}{\alpha-1}} - 1)(t - r(X))$*, for any* $r(X) < t \leq d(X)$.

By repeating the argument for proving Lemma 5 and replacing the use of Lemma 4 by Lemma 8, we have the following lemma.

Lemma 9. $w(A) > (1/E^{\frac{1}{\alpha-1}} - 1)\ell$.

By Lemmas 3 and 9, we have a constant competitive ratio for EFFICIENCY_E.

Theorem 9. *Without demanding jobs,* EFFICIENCY_E *is* $2+\frac{3}{1-E^{\frac{1}{\alpha-1}}}$*-competitive on throughput, for any efficiency threshold* $E < 1$.

References

1. S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *Proc. STACS*, pages 621–633, 2006.
2. N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proc. FOCS*, pages 520–529, 2004.
3. N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *Proc. STACS*, pages 460–471, 2005.
4. N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *Proc. SODA*, pages 805–813, 2007.
5. D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
6. D. P. Bunde. Power-aware scheduling for makespan and flow. In *Proc. SPAA*, pages 190–196, 2006.
7. H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *Proc. SODA*, pages 795–804, 2007.
8. S. A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *J. Algorithms*, 34(2):370–389, 2000.
9. D. Grunwald, P. Levis, K. I. Farkas, C. B. M. III, and M. Neufeld. Policies for dynamic clock scheduling. In *OSDI*, pages 73–86, 2000.
10. S. Irani, R. K. Gupta, and S. Shukla. Algorithms for power savings. In *Proc. SODA*, pages 37–46, 2003.
11. S. Irani and K. Pruhs. Algorithmic problems in power managment. *SIGACT News*, 32(2):63–76, 2005.
12. W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems*, 4(1):211–230, Feb. 2005.
13. M. Li, B. J. Liu, and F. F. Yao. Min-energy voltage allocations for tree-structured tasks. In *Proc. COCOON*, pages 283–296, 2005.
14. M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput.*, 35(3):658–671, 2005.
15. P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, pages 89–102, 2001.
16. K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best resonse for your erg. In *Proc. SWAT04*, pages 15–25, 2004.
17. K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *Proc. WAOA*, pages 307–319, 2005.
18. M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, pages 13–23, 1994.
19. F. F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. FOCS*, pages 374–382, 1995.

Efficient Algorithms for Airline Problem

Shin-ichi Nakano[1], Ryuhei Uehara[2], and Takeaki Uno[3]

[1] Department of Computer Science, Faculty of Engineering, Gunma University, Gunma 376-8515, Japan
nakano@cs.gunma-u.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan
uehara@jaist.ac.jp
[3] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. The airlines in the real world form small-world network. This implies that they are constructed with an ad hoc strategy. The small-world network is not so bad from the viewpoints of customers and managers. The customers can fly to any destination through a few airline hubs, and the number of airlines is not so many comparing to the number of airports. However, clearly, it is not the best solution in either viewpoint since there is a trade off. In this paper, one of the extreme cases, which is the standpoint of the manager, is considered; we assume that customers are silent and they never complain even if they are required to transit many times. This assumption is appropriate for some transportation service and packet communication. Under this assumption, the airline problem is to construct the least cost connected network for given distribution of the populations of cities with no a priori connection. First, we show an efficient algorithm that produces a good network which is minimized the number of vacant seats. The resultant network contains at most n connections (or edges), where n is the number of cities. Next we aim to minimize not only the number of vacant seats, but also the number of airline connections. The connected network with the least number of edges is a tree which has exactly $n-1$ connections. However, the problem to construct a tree airline network with the minimum number of vacant seats is $\mathcal{NP}$-complete. We also propose efficient approximation algorithms to construct a tree airline network with the minimum number of vacant seats.

Keywords: Airline problem, approximation algorithm, efficient algorithm, $\mathcal{NP}$-completeness.

1 Introduction

Small-world networks are the focus of recent interest because of their potential as models for the interaction networks of complex systems in real world [2,8]. In a small-world network, the node connectivities follow a scale-free power-law distribution. As a result, a very few nodes are far more connected than other nodes, and they are called *hubs*. Through those hubs, any two nodes are connected by a short path (see, e.g., [5]). There are many well known small-world networks including the Internet and World Wide Web. Among them, airlines in the real world form small-world networks [1]. In fact,

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 428–439, 2007.
© Springer-Verlag Berlin Heidelberg 2007

some airports are known as airline "hubs." The fact implies that they can be constructed in the same manner as the Internet and World Wide Web; in other words, there were few global strategies for designing efficient airlines. The main reason is that there are so many considerable parameters to be optimized, and some objective functions conflict according to viewpoints; for example, passengers hate to transit, but only complete graph satisfies their demands, which is an impossible solution for airline companies. In fact, it is intractable to design the least cost airline network in general. Even if we fix three hubs, and aim to connect each non-hub to one of the hubs, to design the least cost airline network is $\mathcal{NP}$-hard problem [6,7].

In this paper, we simplify the design problem of an airline network to a simpler graph theoretical problem. We consider the design problem of an airline network as an optimization problem; we aim to give a reasonable strategy to design a network with minimum loss, which corresponds to the number of vacant seats. Let $\mathbb{Z}$ denote the set of positive integers. Then we define *airline problem* over weighted nodes as follows:

Input: A set V of nodes, and a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of edges $\{u, v\}$ in V^2, and a positive integer weight function $w : V \times V \to \mathbb{Z}$ such that for each $v \in V$, we have $w(v) \leq \sum_{\{v,u\}\in E} w(v, u)$, and the graph $G = (V, E)$ is connected[1].

Intuitively, each node v corresponds to a city, and the weight $w(v)$ gives the number of (potential) passengers in the city. If there are already airports, we can count the number of users; otherwise, the weights can be estimated from the populations of the cities. Each edge $\{u, v\}$ corresponds to an airline. An airplane can transport $w(u, v)$ passengers at one flight. Airplanes make regular flights along the edges, both ways, and simultaneously. Hence the number of passengers, $w(v)$, does not fluctuate in a long term. We consider that the condition $w(v) \leq \sum_{\{v,u\}\in E} w(v, u)$ for each v in V is appropriate condition as an airline network, if all cities are connected by the airline network. The condition is enough to supply the least service. In other words, we assume that passengers are silent; they never complain even if they have to transit many times. Therefore the airline problem can be an idealized model for planning some real network problem like transportation service, peer to peer file transfer network, and data network flow in the sense that producing a reasonable (or cheapest) network that can satisfy given demands. After designing the network, we will face the assignment problems. However, the assignment problems are separated in this context; we only mention that each cheap network dealt in this paper has at least one reasonable solution obtained by a random walk approach, which is omitted here.

To evaluate the "goodness" of a solution for the airline problem, we define the *loss* $L(v)$ at v by $(\sum_{\{v,u\}\in E} w(v, u)) - w(v)$. If the solution is feasible, we have $L(v) \geq 0$ for all $v \in V$. Intuitively, the loss $L(v)$ gives the total number of vacant seats of departure flights from the city v. We denote by $L(G) := \sum_{v\in V} L(v)$ the *total loss* of the graph (or solution) $G = (V, E)$. We here observe that $L(G)$ is given by $\sum_{v\in V} L(v) = \sum_{v\in V}((\sum_{\{v,u\}\in E} w(v, u)) - w(v)) = \sum_{v\in V} \sum_{\{v,u\}\in E} w(v, u) - \sum_{v\in V} w(v) = 2 \sum_{e\in E} w(e) - \sum_{v\in V} w(v)$.

We first consider the airline problem to generate a connected network with the minimum loss $L(G)$. We show an efficient algorithm that minimizes the total loss of the

[1] The weight of an edge $e = \{u, v\}$ should be denoted by $w(e) = w(\{u, v\}) = w(\{v, u\})$. However, we denote them by $w(e) = w(u, v)(= w(v, u))$ for short.

flights on the network. The algorithm generates a connected network of at most $|V|$ edges in $O(|V|)$ time and $O(|V|)$ space. Since the minimum number of the edges of a connected graph with $|V|$ vertices is $|V| - 1$, our algorithm produces the least cost connected network with $|V| - 1$ or $|V|$ airlines. Hence it is natural to ask that if we can restrict ourselves to construct a weighted tree with the minimum loss. It is worth mentioning that tree network has an advantage that a shortest route between two vertices are uniquely determined. However, interestingly, the problem is intractable; the airline problem to construct a tree airline network (of $|V| - 1$ edges) with the minimum weight (or the minimum loss) is $\mathcal{NP}$-complete. For the $\mathcal{NP}$-complete problem, we give two efficient approximation algorithms. First one always finds a tree airline network T of V of approximation ratio 2 in $O(|V|)$ time and space. More precisely, the algorithm constructs a weighted tree T that has additional weight $w_{\max}$ than the optimal weight among all weighted connected networks that is not necessarily a tree, where $w_{\max} = \max_{v \in V} w(v)$. The second one is based on an FPTAS for the weighted set partition. Assume we obtain a partition X and Y of V with $\left|\sum_{x \in X} w(x) - \sum_{y \in Y} w(y)\right| \leq \delta$ for some $\delta \geq 0$ by an FPTAS. Then, from X and Y, we can construct a weighted tree T with $L(T) \leq \max\{\delta, 2\}$ in $O(|V|)$ time and space.

2 Minimum Cost Network

In this section, we show efficient algorithms for constructing a connected network of minimum loss for given weighted nodes V. Hereafter, we denote $|V|$ by n and $\sum_{v \in V} w(v)$ by W. Since the case $n = 1$ is trivial, we assume that $n > 1$. The main theorem in this section is the following:

Theorem 1. *Let V be a set of n nodes, and w be a positive integer weight function $w : V \to \mathbb{Z}$. Then a connected network E over V of the minimum loss $L(G)$ with $|E| \leq n$ can be found in $O(n)$ time and $O(n)$ space.*

Procedure: `Span`(u, v, w)

Input : Two vertices u and v, and weight w with $w(u), w(v) \geq w$.
Output: An edge $\{u, v\}$ of weight w.

1 $w(u, v) := w$;
2 $w(u) := w(u) - w$; $w(v) := w(v) - w$;
3 **if** $w(u)=0$ **then** remove u;
4 **if** $w(v)=0$ **then** remove v;
5 **return** $(\{u, v\})$;

Throughout the paper, we will use procedure `Span` as a basic operation. Mainly, given two vertices u and v, we span an edge $\{u, v\}$ of weight $\min\{w(u), w(v)\}$ and remove one of them. We first have the following lemma:

Lemma 1. *If $L(G) = 1$, the solution is optimal.*

Proof. We remind that $L(G)$ is given by $\sum_{v\in V} L(v) = \sum_{v\in V}((\sum_{\{v,u\}\in E} w(v,u)) - w(v)) = \sum_{v\in V}\sum_{\{v,u\}\in E} w(v,u) - \sum_{v\in V} w(v) = 2(\sum_{e\in E} w(e)) - W$. Thus when $L(G) = 1$, W is odd, which is the input, and hence we cannot improve it. □

We start with the following three special cases:

Star condition: Let $v_{\max}$ be a heaviest vertex, i.e., $w(v_{\max}) \geq w(v)$ for all $v \in V$. We say *star condition* if we have $w(v_{\max}) - \sum_{v\in V\setminus\{v_{\max}\}} w(v) \geq 0$.

Uniform condition: When $w(v) = w > 0$ for all $v \in V$, we call it *uniform condition*.

Many-ones condition: If V contains at least two vertices of weight 1, we call that *many-ones condition*. To distinguish it from the uniform condition, we assume that V also contains at least one vertex of weight greater than 1.

We have either the star or uniform condition if $|V| \leq 2$. Hence, hereafter, we assume that $|V| > 2$. Under the star condition, Algorithm `Star` computes the solution with minimum loss $L(G) = w(v_{\max}) - \sum_{v\in V\setminus\{v_{\max}\}} w(v)$ in $O(n)$ time and space. It is also easy to see that $|E|$ contains $n-1$ edges.

Algorithm 2: `Star`

Input : A set V of n nodes, a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of $m = n-1$ edges $\{u, v\}$ such that (V, E) is connected, and a positive integer weight function $w : E \to \mathbb{Z}$.

```
1 let v_max be a vertex such that w(v_max) ≥ w(v) for all v ∈ V;
2 foreach v ∈ V \ {v_max} do Span(v,v_max,w(v));
3 pickup any e = (v', v_max) with w(e) > 0;
4 w(e) := w(e) + w(v_max);
5 return (E := {e | w(e) > 0});
```

Algorithm 3: `Uniform`

Input : A set V of n nodes, a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of m ($m = n$ or $m = n-1$) edges $\{u, v\}$ such that (V, E) is connected, and a positive integer weight function $w : E \to \mathbb{Z}$.

```
1 let w be a positive integer such that w = w(v) for all v ∈ V;
2 if w = 1 then make any spanning tree T over V, and w(e) := 1 for each edge e ∈ T;
3 else
4     foreach odd i = 1, 3, 5, ... do Span(v_i,v_{i+1},⌊w/2⌋);
5     foreach even i = 2, 4, 6, ... do Span(v_i,v_{i+1},⌈w/2⌉);
6     Span(v_n,v_1,⌈w/2⌉);
7 return (E := {e | w(e) > 0});
```

Lemma 2. *In the uniform condition, Algorithm* `Uniform` *produces a connected network E over V of the minimum loss $L(G)$ in $O(n)$ time and $O(n)$ space, where $|V| = n$. Moreover, $|E| \leq n$.*

Proof. Omitted. □

Algorithm 4: `Many-ones`

Input : A set V of n nodes, a positive integer weight function $w : V \to \mathbb{Z}$.
Output: V' of n' nodes and E of $n - n'$ edges s. t. V' has at most one vertex of weight 1.

1 let $v_{\max}$ be a vertex such that $w(v_{\max}) \geq w(v)$ for all $v \in V$;
2 let $V_1 := \{v \mid w(v) = 1\}$ and $V_2 := \{v \mid w(v) \geq 2\} \setminus \{v_{\max}\}$;
3 **while** $|V_1| > 1$ *and* $|V_2| > 0$ **do**
4 **if** *V satisfies the star condition* **then** call `Star` as a subroutine and halt;
5 for any vertices $v \in V_1$ and $v' \in V_2$, `Span`$(v,v',1)$;
6 **if** $w(v') = 1$ **then** move v' from V_2 to V_1;
7 **if** $|V_2| = 0$ **then**
8 **if** *V satisfies the star condition* **then** call `Star` as a subroutine and halt;
9 call `Span`$(v_{\max},v,1)$ for any $w(v_{\max}) - 1$ vertices $v \in V_1$; `// to make` $w(v_{\max}) = 1$
10 call `Uniform` as a subroutine and halt;
11 **return** (V and $E := \{e \mid w(e) > 0\}$);

We next turn to the many-ones condition. We first partition V into two disjoint subsets $V_1 := \{v \mid w(v) = 1\}$ and $V_2 := \{v \mid w(v) \geq 2\}$. Then we have many-ones condition iff $|V_1| > 1$ and $|V_2| > 0$. We also pick up a vertex $v_{\max}$ of the maximum weight as a special vertex to check if the vertex set satisfies the star condition. The purpose here is to reduce the number of vertices of weight 1 to one. Hence we join the vertices in V_1 to the other vertices and output the remaining vertices, which contains at most one vertex of weight 1. This is the preprocess of the main algorithm.

The vertex $v_{\max}$ can be found in $O(n)$ time, and $\sum_{v \in V \setminus \{v_{\max}\}} w(v)$ can be maintained decrementally. Hence Algorithm `Many-ones` runs in $O(n)$ time and space by maintaining V_1 and V_2 by two queues. `Many-ones` terminates if the vertex set satisfies the star condition, the uniform condition, or the set contains at most one vertex of weight 1. In the former two cases, we already have a solution.

Now, we can assume that the input V satisfies neither the star, uniform, nor many-ones conditions. Then, we have the following lemmas:

Lemma 3. *Algorithm* `Network` *outputs a connected network* (V, E) *with* $|E| \leq n$.

Proof. In the while-loop, the algorithm either (0) calls `Star` or `Uniform` and halts, or (1) sets $w(v, v')$ for some v, v' and removes one of v and v'. The algorithm also joints all vertices in some vertex set $\hat{V}$ with $\left|\hat{V}\right|$ edges. Hence we have an invariant that the total number of removed vertices is greater than or equal to the total number of added edges. Hence $|E|$ contains at most n edges. It is easy to see that the resultant network is connected. □

Lemma 4. *Algorithm* `Network` *always outputs a network with the minimum loss* $L(G)$.

Proof. Let V_1 be the set of vertices of weight 1. Then we have an invariant $|V_1| \leq 1$ throughout the execution of the algorithm. Let $v_{\max}$ be the heaviest vertex chosen in step 2. We have two cases; (I) all vertices in $V \setminus \{v_{\max}\}$ have the same weight w, and (II) there are two vertices v_i and v_j in $V \setminus \{v_{\max}\}$ with $w(v_i) < w(v_j)$.

(I) This case is handled in steps 8 to 18. We first note that V is in neither the uniform nor star cases. Thus, we have $w(v_{\max}) > w$ and $w(v_{\max}) < (n - 1)w$. Mainly, in the case,

Algorithm 5: `Network`

Input : A set V of n nodes, and a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of m ($m = n$ or $m = n - 1$) edges $\{u, v\}$ such that (V, E) is connected, and positive integer weight function $w : E \to \mathbb{Z}$.

1 $n := |V|$, and $W := \sum_{v \in V} w(v)$;
2 find $v_{\max}$ such that $w(v_{\max}) \geq w(v)$ for any other $v \in V$;
3 let $V_1 := \{v \mid w(v) = 1\}$ (we have $|V_1| < 2$);
4 **while** true **do**
5 **if** V *satisfies the star condition* **then** call `Star` as a procedure and halt;
6 **if** V *satisfies the uniform condition* **then** call `Uniform` as a procedure and halt;
7 **if** V *satisfies the uniform condition* **then**
8 **if** $(n-2)w < w(v_{\max})$ **then** `// we also have` $w(v_{\max}) < (n-1)w$
9 let v_i and v_j be any two vertices in $V \setminus \{v_{\max}\}$;
10 `Span`$(v_i, v_j, \lceil((n-1)w - w(v_{\max}))/2\rceil)$; `// to have star condition`
11 **else**
12 $r := w(v_{\max}) \bmod w$;
13 **if** $r = 0$ **then**
14 let S consist of any $(w(v_{\max})/w) - 1$ vertices from $V \setminus \{v_{\max}\}$;
15 **else**
16 let S consist of any $\lfloor w(v_{\max})/w \rfloor$ vertices from $V \setminus \{v_{\max}\}$;
17 **foreach** $v \in S$ **do** `Span`$(v, v_{\max}, w)$; ; `// we have` $S = \emptyset$
18 **if** $w(v_{\max}) = 1$ **then** put $v_{\max}$ into V_1;
19 **else**
20 **if** $V_1 \neq \emptyset$ **then**
21 let v_i be the vertex in V_1, and v_j be any vertex in $V \setminus \{v_{\max}\}$;
22 **else**
23 let v_i and v_j be any vertices in $V \setminus \{v_{\max}\}$ with $w(v_i) < w(v_j)$;
24 **if** $W - 2w(v_i) > 2w(v_{\max})$ **then**
25 `Span`$(v_i, v_j, w(v_i))$;
26 **if** $w(v_j) = 1$ **then** put v_j into V_1;
27 **else**
28 `Span`$(v_i, v_j, \lceil(W - 2w(v_{\max}))/2\rceil)$; `// to have star condition`
29 update V, n, W, $v_{\max}$ if necessary;

the algorithm takes the vertices of weight w by matching with $v_{\max}$ as follows. Let q be $\lfloor w(v_{\max})/w \rfloor$ and r be $w(v_{\max}) \bmod w$.

If $r = 0$, the last vertex of weight w cannot be matched to $v_{\max}$ since all vertices spanned by the edges have weight 0, and we will have a loss. Hence, in the case, the algorithm matches $q - 1$ vertices of weight w to $v_{\max}$, and then $w(v_{\max})$ becomes w. That is, we will have the uniform case in the next iteration. Through the process, the algorithm generates no loss, which is handled in steps 14, 17, 18. Hence if the uniform case will be handled properly, the algorithm generates no loss, which will be discussed later.

If $r \neq 0$, we can match q vertices of weight w to $v_{\max}$ by edges of weight w. After the matching, we remove q vertices from $V \setminus \{v_{\max}\}$, and $w(v_{\max})$ is updated by $w(v_{\max}) - qw$.

If $w(v_{\max}) - qw$ is enough large comparing to the total weight of the remaining vertices of weight w, the process is done properly in steps 16, 17, 18. However, the process fails when $w(v_{\max}) - qw$ is too light; for example, when $V = \{v_1, v_2, v_3\}$ with $w(v_1) = 8, w(v_2) = w(v_3) = 5$, we cannot make an edge $\{v_1, v_2\}$ of weight 5. The resultant vertex v_3 will generate loss 2. In the case, we have to make $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\}$ with $w(v_1, v_2) = w(v_1, v_3) = 4$ and $w(v_2, v_3) = 1$. To consider the case, we partition $V \setminus \{v_{\max}\}$ into V_a of q vertices and V_b of $n - q - 1$ vertices. The loss will be generated, after removing all vertices in V_a which are matched with $v_{\max}$, if (1) $\{v_{\max}\} \cup V_b$ satisfies the star condition, and (2) $w(v_{\max}) < w$. They occur only if $|V_b| = 1$, which is equivalent to $(n-2)w < w(v_{\max}) < (n-1)w$. This case is handled in steps 9 and 10. In the case, we can have the optimal solution with the following assignments of weights; pick up any two vertices v_i and v_j from $V \setminus \{v_{\max}\}$, and add the edge $\{v_i, v_j\}$ of weight $\lceil((n-1)w - w(v_{\max}))/2\rceil$. Then we have the star condition, and we have $L(G) \leq 1$, which is the optimal.

(II) This case is handled in steps 20 to 28. Let v_i and v_j be any two vertices of different weights with $w(v_i) < w(v_j)$. If $|V_1| = 1$, the algorithm takes the unique vertex of weight 1 as v_i. When $w(v_j)$ is not so heavy, we add an edge $\{v_i, v_j\}$ with $w(v_i, v_j) = w(v_i)$ and remove v_i in step 25. The exception is that removing $w(v_i)$ results in the star condition with loss, which is equivalent to $\sum_{v \in V \setminus \{v_{\max}\}} w(v) = W - w(v_{\max}) - 2w(v_i) < w(v_{\max})$. Hence the case occurs when $W - 2w(v_i) < 2w(v_{\max})$. On the other hand, we did not have the star condition before removing $2w(v_i)$ from $w(v_i)$ and $w(v_j)$. Thus, before removing, we had $\sum_{v \in V \setminus \{v_{\max}\}} w(v) = W - w(v_{\max}) > w(v_{\max})$, or consequently, $W > 2w(v_{\max})$. In the case, we can have the star condition without loss by the edge $\{v_i, v_j\}$ with $w(v_i, v_j) = \left\lceil \frac{\sum_{v \in V \setminus \{v_{\max}\}} w(v) - w(v_{\max})}{2} \right\rceil = \left\lceil \frac{W - 2w(v_{\max})}{2} \right\rceil$, and then we have the optimal in step 28.

Hence, in most cases, the algorithm achieves the optimal network. The last case is in the following case: The algorithm does not call `Uniform` at first, and it calls `Uniform`, which outputs a spanning tree since all vertices have the weight 1. However, this case is impossible since we have an invariant $|V_1| \leq 1$.

Thus, Algorithm `Network` always outputs a network with $L(G) \leq 1$, which is optimal by Lemma 1, if V does not satisfy one of three special conditions. □

Lemma 5. *Algorithm* `Network` *runs in* $O(n)$ *time and space.*

Proof. If we admit to sort the vertices, it is easy to implement the algorithm to run in $O(n \log n)$ time and $O(n)$ space. To improve the time complexity to $O(n)$, we show how to maintain $v_{\max}$ and determine if all vertices in a vertex set $V \setminus \{v_{\max}\}$ have the same weight efficiently. In step 2, the algorithm first finds $v_{\max}$ in $O(n)$ time. Then we can check if V satisfies the star condition or the uniform condition in $O(n)$ time. In the while-loop, two special vertices $v_{\max}$ and the unique vertex, say v_1, in V_1 (if exist) are maintained directly, and all other vertices in $V' = V \setminus \{v_{\max}, v_1\}$ are maintained in a doubly linked list. The number n of vertices are also maintained.

We first assume that all vertices in V' have the same weight w. If $(n-2)w < w(v_{\max})$, the algorithm halts in $O(n)$ time. Hence we assume that $w(v_{\max}) \leq (n-2)w$. (Note that $(n-1)w \leq w(v_{\max})$ implies the star condition.) In the case, the algorithm computes $r = w(v_{\max}) \bmod w$ in $O(1)$ time. If $r = 0$, the algorithm removes $(w(v_{\max})/w) - 1$ vertices from V'. After that, $w(v_{\max})$ becomes $w(v_{\max}) = w$, and we have the uniform case. Thus

the algorithm can call `Uniform` without checking the condition. The time complexity can be bounded above by $O(|V'|)$. If $r \neq 0$, the algorithm removes $\lfloor w(v_{\max})/w \rfloor$ vertices from V'. After that, $w(v_{\max})$ becomes $w(v_{\max}) < w$, and the other vertices have the same weight w. We update $v_{\max}$ by any vertex in $V' \setminus \{v_{\max}\}$. Through the step, the running time is proportional to the number of the vertex removed.

Next, we assume that there are some different weight vertices in V'. The pair v_i and v_j of different weights can be found by traversing the doubly linked list. Let $v_2, v_3, \ldots$ be the consecutive vertices in the list. If $V_1 \neq \emptyset$, the pair $v_i = v_1$ and $v_j = v_2$ can be found in $O(1)$ time. Otherwise, the algorithm checks if $w(v_1) = w(v_2)$, $w(v_2) = w(v_3)$, or $w(v_3) = w(v_4)$, ... until it finds $w(v_k) \neq w(v_{k+1})$. Then set $v_i := \min\{v_k, v_{k+1}\}$ and $v_j := \max\{v_k, v_{k+1}\}$. Moreover, in the case, the algorithm knows that $w(v_1) = w(v_2) = \cdots = w(v_k)$. When $W - 2w(v_k) \leq 2w(v_{\max})$, the algorithm connects all vertices and halts in time $O(|V'|)$. Hence we assume that $W - 2w(v_k) > 2w(v_{\max})$. Then the algorithm removes v_i or v_j in $O(1)$ time from the linked list. After updating n and W, the algorithm has to check if all vertices in V' have the same weight. Since the algorithm knows that $w(v_1) = w(v_2) = \cdots = w(v_{i-1})$, it is enough to check from v_{i-1}. Thus the total time to check if V' contains at least two vertices of different weights is bounded above by $O(n)$.

Hence, the algorithm runs in $O(n)$ time and space. □

By Lemmas 2, 3, 4, and 5, we immediately have Theorem 1.

3 Minimum Cost Spanning Tree

In this section, we first prove that the problem for finding a minimum loss tree airline network is $\mathcal{NP}$-complete. Next, we show approximation algorithms for the problem.

3.1 $\mathcal{NP}$-Hardness for Finding a Spanning Tree of Minimum Loss

We first modify the optimization problem to the decision problem as follows; the input of the algorithm consists of a set V of nodes, a positive integer weight function $w(v)$ for each $v \in V$, and an integer k. Then the decision problem is to determine if there is the set E of edges and a positive integer weight function $w(u, v)$ such that they provide a feasible solution of the airline problem with $L(G) \leq k$ and (V, E) induces a (connected) tree.

Theorem 2. *The decision problem for finding a tree airline network is $\mathcal{NP}$-complete.*

Proof. The problem is clearly in $\mathcal{NP}$. We reduce it to the following well known $\mathcal{NP}$-complete problem [4, [SP12]]:

Problem: Weighted Set Partition
Input: Finite set A and weight function $w'(a) \in Z^+$ for each $a \in A$;
Output: Determine if there is a subset $A' \subset A$ s. t. $\sum_{a \in A'} w'(a) = \sum_{a \in A \setminus A'} w'(a)$.

Let $W := \sum_{a \in A} w'(a)$. Without loss of generality, we assume that W is even. For given $A = \{a_1, a_2, \ldots, a_n\}$ and the weight function w', we construct the input V and w of the airline problem as follows; $V = A \cup \{u, v\}$, and $w(a) = w'(a)$ for each vertex a in A. We

define $w(u) = w(v) = \frac{W}{2} + 1$. The reduction can be done in polynomial time and space. We show that A can be partitioned into two subsets of the same weight if and only if V has a tree airline network with no loss. Let E be the set of weighted edges of the minimum loss. We first observe that if $G = (V, E)$ achieves $L(G) = 0$, E has to contain the positive edge $\{v, u\}$. Otherwise, the edges incident to u or v have to have total weight $W + 2 > W$, and then we have $L(G) > 0$.

First we assume that A has a partition A_1 and A_2 such that $A_1 \cup A_2 = A$, $A_1 \cap A_2 = \emptyset$, and $\sum_{a \in A_1} w(a) = \sum_{a \in A_2} w(a) = W/2$. We show that V has a tree airline network with no loss. We define the weight function w as follows; $w\{u, v\} = 1$, $w\{a, u\} = w(a)$ for all $a \in A_1$, and $w\{a, v\} = w(a)$ for all $a \in A_2$. By assumption and construction, the set E of positive weighted edges is a tree airline network with no loss.

Next we assume that V has a tree airline network with no loss, and show that A can be partitioned into A_1 and A_2 of the same weight. By the observation, the edge $\{u, v\}$ has a positive weight, say w'. We then partition the set A into A_1 and A_2 as follows; A_1 consists of vertices $a \in A$ of odd distance from u, and A_2 consists of vertices $a \in A$ of even distance from u. Since T is a tree, A_1 and A_2 satisfy $A_1 \cap A_2 = \emptyset$, $A_1 \cup A_2 = A$, and two sets A_1 and A_2 are independent sets. Moreover, since T has no loss, $\sum_{e \in (A_1 \setminus \{v\}) \times \{u\}} w(e) = \sum_{e \in (A_2 \setminus \{u\}) \times \{v\}} w(e) = \frac{W}{2} + 1 - w'$, and $\sum_{e \in (A_1 \setminus \{v\}) \times (A_2 \setminus \{u\})} = w' - 1$. Hence we have $\sum_{a \in A_1 \setminus \{v\}} w(a) = \sum_{a \in A_2 \setminus \{u\}} w(a) = \frac{W}{2}$. Thus A_1 and A_2 gives a solution of the weighted set partition problem.

Therefore, the weighted set partition problem can be polynomial time reducible to the problem for finding a tree airline network of minimum loss, which completes the proof. □

3.2 Approximation Algorithms for a Tree Airline Network

In this section, we show two approximation algorithms that aim at different goals. First one gives us a simple and efficient algorithm with approximation ratio 2. Second one is based on an FPTAS for the set partition problem, which gives us a polynomial time algorithm with arbitrary small approximation ratio.

Simple 2-approximation algorithm. The simple algorithm is based on the algorithm stated in Section 2. The algorithm in Section 2 outputs a connected network with at most n edges. The algorithm outputs the nth edge when (1) it is in the uniform case, or (2) the edge $\{v_i, v_j\}$ is produced in step 10 or step 31 by Algorithm `Network`. We modify each case as follows and obtain a simple approximation algorithm.

(1) In the uniform case with $w > 1$, pick up any pair of vertices $\{v_i, v_{i+1}\}$ such that $w(v_i, v_{i+1}) = \lfloor w/2 \rfloor$. Then, cut the edge, and add their weight to adjacent edges; $w(v_{i-1}, v_i) := w(v_{i-1}, v_i) + \lfloor w/2 \rfloor$, and $w(v_{i+1}, v_{i+2}) := w(v_{i+1}, v_{i+2}) + \lfloor w/2 \rfloor$. In the case, $L(G)$ increases by $2 \lfloor w/2 \rfloor \leq w$.

(2) In both cases, the vertices v_i and v_j will be joined to $v_{\max}$ in the next iteration since V satisfies the star condition. Hence we add the weight of the edge $\{v_i, v_j\}$ to $\{v_i, v_{\max}\}$ and $\{v_j, v_{\max}\}$. In the former case, $L(G)$ increases by $2 \lceil (w(v_{\max}) - (n-2)w)/2 \rceil \leq w(v_{\max}) - (n-2)w + 1 < w(v_{\max})$. In the latter case, $L(G)$ increases by $2 \lceil (W - 2w(v_{\max}))/2 \rceil < w(v_{\max})$.

From above analysis, we immediately have the following theorem:

Theorem 3. *The modified algorithm always outputs a connected tree $T = (V, E)$ with $L(T) \leq w(v_{\max})$ in $O(n)$ time and space.*

Let E' be any feasible solution (which does not necessarily induce a tree) of the airline problem. Then, clearly, $\sum_{e \in E'} w(e) \geq w(v_{\max})$. Thus we have the following corollary.

Corollary 1. *Let E be the set produced by the modified algorithm, and E_{opt} be an optimal solution (with the minimum loss) of the airline problem. Let $T := (V, E)$ and $G := (V, E_{opt})$. Then, $\sum_{e \in E_{opt}} w(e) \leq \sum_{e \in E} w(e) < 2 \sum_{e \in E_{opt}} w(e)$.*

Approximation algorithm based on FPTAS. A weighted set partition problem has an FPTAS based on a pseudo-polynomial time algorithm. The idea is standard and can be found in a standard text book, for example, [3, Chapter 35.5]. Hence, using the FPTAS algorithm, we can compute a partition X and Y of V with $\frac{\left|\sum_{v \in X} w(v) - \sum_{v \in Y} w(v)\right| - \left|\sum_{v \in X^*} w(v) - \sum_{v \in Y^*} w(v)\right|}{\sum_{v \in V} w(v)} < \epsilon$ for any positive constant ϵ in polynomial time of $|V|$ and ϵ, where X^* and Y^* are an optimal partition of V.

In this section, we show a polynomial time algorithm that constructs a tree airline network from the output of the the FPTAS for the weighted set partition problem for the same input V and w.

By the results in Section 2, if V satisfies either the star condition or the uniform condition, we can obtain a tree airline network that is an optimal solution. On the other hand, if V contains many vertices of weight 1, we can reduce them by Algorithm `Many-ones`. Hence, without loss of generality, we assume that V is neither in the star condition nor in the uniform condition, and V contains at most one vertex of weight 1.

We first regard V and w as an input to the weighted set partition problem. Then we run the FPTAS algorithm for the weighted set partition problem. Let X and Y be the output of the algorithm. That is, $\delta := \left|\sum_{v \in X} w(v) - \sum_{v \in Y} w(v)\right|$ is minimized by the FPTAS algorithm. We note that an optimal partition X^* and Y^* of V gives the lower bound of the optimal solution for the airline problem; we cannot have $L(T) < \left|\sum_{v \in X^*} w(v) - \sum_{v \in Y^*} w(v)\right|$ for any weighted tree T that spans V. We can make a tree airline network that achieves the same performance by the FPTAS.

Theorem 4. *Let X and Y be the partition of V produced by an FPTAS for the weighted set partition problem, and $\delta := \left|\sum_{v \in X} w(v) - \sum_{v \in Y} w(v)\right|$. Then, from X and Y, we can construct a connected network E such that $T = (V, E)$ is a tree with $L(T) \leq \max\{\delta, 2\}$. The tree T can be constructed in $O(|V|)$ time and space.*

Proof. The algorithm consists of two phases.

Let v_0 be the vertex in V of the minimum weight, i.e., $w(v_0) \leq w(v)$ for any $v \in V$. If v_0 is uniquely determined (or $w(v_0) \neq w(v)$ for each $v \in V \setminus \{v_0\}$), the algorithm performs the first phase, and otherwise, the algorithm runs from the second phase.

We first show the first phase, which runs if v_0 is uniquely determined. Without loss of generality, we assume that $v_0 \in X$. We let $X = \{x_0 = v_0, x_1, x_2, \ldots, \}$ and $Y = \{y_1, y_2, \ldots, \}$. (We note that $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ are ordered in arbitrary way.) The

first phase is given in Algorithm `Caterpillar`; it starts from a path $\{x_0, y_1\}$, and extend it as possible as it can until the next vertex pair becomes the same weight. (The resultant graph makes a graph that is known as a caterpillar which consists of path where each vertex on the path has some pendant vertices.) After the first phase, if $X = \emptyset$ or $Y = \emptyset$, we complete the tree by joining all vertices in the non-empty set (if it exists) to the last vertex touched in the empty set. In the case, the tree T admits $L(T) = \delta$. Hence we assume that $X \neq \emptyset$ and $Y \neq \emptyset$, and $w(x_i) = w(y_j) = w$ for some i, j, and w. By the algorithm and initial condition, we have $i > 0$ and one of $w(x_i)$ and $w(y_j)$ is updated, and the other one is not updated. Hence $w > w(v_0)$.

Algorithm 6: `Caterpillar`

```
i := 0; j := 1;
while w(x_i) ≠ w(y_j) and X ≠ ∅ and Y ≠ ∅ do
    if w(x_i) < w(y_j) then
        Span(x_i,y_j,w(x_i));
        i := i + 1;
    else
        Span(x_i,y_j,w(y_j));
        j := j + 1;
```

Now, we turn to the second phase. We now renumber the vertices as X=$\{x_0, x_1, x_2, \ldots\}$ and $Y = \{y_0, y_1, y_2, \ldots\}$ such that $w(x_0) \leq w(x_i)$ and $w(y_0) \leq w(y_i)$ for each $i > 0$. By assumption, the input V contains at most one vertex of weight 1. Hence now we have $w(x_0) = w(y_0) > 1$ by the first phase.

If the algorithm runs the first phase, one of x_0 and y_0 is an endpoint of the caterpillar. Without loss of generality, we assume that x_0 is the endpoint. (We regard that x_0 is the endpoint of the graph of size 1 if the algorithm runs from the second phase.) The algorithm extends the tree from x_i as follows. It searches y_j with $w(x_0) \neq w(y_j)$ from $\{y_1, y_2, \ldots\}$.

If the algorithm finds $w(y_j)$ with $w(x_0) \neq w(y_j)$, it calls $\mathtt{Span}(x_0,y_j,\min\{w(x_0), w(y_j)\} = w(x_0))$. Then the algorithm repeats the first phase with the vertex pair x_1 and y_j; we remark that the algorithm knows that $w(y_0) = w(y_1) = \cdots = w(y_{j-1}) = w$, which will be preferred than the other vertices in the next phase, and the algorithm can omit to check if they have the same weight.

When the algorithm do not find $w(y_j)$ with $w(x_0) \neq w(y_j)$, we have $w(x_0) = w(y)$ for all $y \in Y$. In the case, the algorithm searches x_i with $w(x_i) \neq w(x_0)$. If the algorithm finds $w(x_i)$ with $w(x_i) \neq w(x_0)$, it calls $\mathtt{Span}(x_i,y_0,\min\{w(x_i), w(y_0)\} = w(y_0))$. Then the algorithm repeats to join the vertices in Y to x_i until x_i is removed. If x_i is removed while $Y \neq \emptyset$, the last touched vertex y_j in Y satisfies $w(y_j) < w(y)$ for each $y \in Y$ and $w(y_j) < w(x_0)$ since all vertices in Y had the same weight equal to x_0. Thus the algorithm repeats the first phase for the pair $\{x_0, y_j\}$. If we have $Y = \emptyset$ and $w(x_i) > 0$, the algorithm picks up the last vertex y in Y and connect all vertices in X to y with their weights. In the case, the algorithm achieves the loss δ.

Now, we have the last case: $w(x) = w(y) = w$ for all $x \in X$ and $y \in Y$. We renumber $X = \{x_1, x_2, \ldots, x_k\}$ and $Y = \{y_1, y_2, \ldots, y_{k'}\}$. By the process above, every

connected subtree has exactly one node in $X \cup Y$. Thus we have a weighted tree T spanning V by joining those vertices. Moreover, since the vertices are preferred if they are touched, both of X and Y contain at least one vertex whose weight was not updated by the algorithm, respectively. If $|k - k'| > 1$, we can improve δ by moving the untouched vertex. Hence we have $k = k'$ or $|k - k'| = 1$. First, we assume that $|k - k'| = 0$. If $k = k' = 1$, the algorithm completes the tree by joining $\{x_1, y_1\}$ with $w(x_1, y_1) = w$. When $k = k' > 1$, the algorithm completes a spanning tree T by the path $(x_1, y_1, \ldots, x_k, y_k)$ with $w(x_1, y_1) = w(x_k, y_k) = w$, $w(x_i, y_i) = w - 1$ and $w(y_i, x_{i+1}) = 1$ for $1 < i < k$. Then we have $L(T) = 2$. If $k = k' + 1$, the path $(x_1, y_1, \ldots, x_{k-1}, y_{k-1}, x_k)$ with $w(x_1, y_1) = w$, $w(y_{k-1}, x_k) = w$, $w(x_i, y_i) = w - 1$ and $w(y_i, x_{i+1}) = 1$ for $1 < i < k - 1$ gives us the tree T with $L(T) = \delta$. The case $k = k' - 1$ is symmetric.

Thus, the algorithm outputs a tree airline network T with $L(T) \leq \max\{2, \delta\}$. By similar implementation using queue of the vertices of the same weight in the proof of Lemma 5, the algorithm runs in $O(n)$ time and $O(n)$ space. □

4 Concluding Remarks and Acknowledgment

In this paper, we do not deal with the assignment problem over the constructed network. When each vertex has its destination, the assignment problem is further challenging problem.

The authors are partially supported by the Ministry, Grant-in-Aid for Scientific Research (C).

References

1. L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. Classes of small-world networks. *Applied Physical Science*, 97(21):11149–11152, October 2000.
2. A.L. Barabasi. *Linked: The New Science of Networks*. Perseus Books Group, 2002.
3. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
4. M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.
5. M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
6. M. O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32:393–404, 1987.
7. J. Sohn and S. Park. The Single Allocation Problem in the Interacting Three-Hub Network. *Networks*, 35:17–25, 2000.
8. D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 2004.

Efficient Exact Arithmetic over Constructive Reals

Yong Li and Jun-Hai Yong

School of Software, Tsinghua University, Beijing 100084, P.R. China
exactreal@yahoo.com.cn

Abstract. We describe a computing method of the computable (or constructive) real numbers based on analysis of expressions. This method take precision estimate into account in order to get a better algorithm than Ménissier-Morain's method, which is also based on the representation of constructive reals. We solve two problems which appear in exact real arithmetic based on the representation of constructive reals. First, by balancing every item's precision in the expression, we can avoid unnecessary precision growth. Second, by distributing different weights to different operations, we can make sure that complex operations do not waste much time when to compute the whole expression. In these ways, we finally get a more efficient and proper method than prior implementations.

1 Introduction

The goal of exact arithmetic is to generate correct answers to numeric problems, within some user-specified error. Generally speaking, floating-point numbers are widely used to represent real numbers in computer systems, although the limitations are quit obvious. For instance, the real numbers cannot be arbitrarily large or small, and the rounding errors can lead to inaccurate or completely wrong results. Some examples can be found in [1,2]. In order to solve these limitations, many methods have been proposed, one of which is exact arithmetic that has attracted more and more attentions in recent years.

Several methods have been developed for exact arithmetic which can supply an approximation of the exact result that meets the user's any requirement of precise. V. Ménissier-Morain in [3] gives a brief introduction of these methods. However, only two of them are practical and proved completely correct. One is linear fractional transformation (LFT) which is mostly based on continued fractions. In 1976, Gosper first introduced this mthod to calculate the operations of real numbers [4]. In 1989, Vuillemin implemented the arithmetic using Lisp and improved this arithmetic in many ways [5]. In 2002, Edalat and Heckmann developed this arithmetic to LFT Framework [1] and they also supply an implementation in Haskell and C, which is named IC-Reals. The other one , constructive reals, is based on Cauchy Sequences, a theory that is well established by Bishop[6]. It can also be considered as a foundation of computable analysis. Boehm and Lee first made use of this theory for the purpose of exact arithmetic [7,8] and Boehm finished this algorithm using java [9]. Gowland and Lester in

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 440–449, 2007.
© Springer-Verlag Berlin Heidelberg 2007

[10,11] showed us the correctness of an implementation based on fast binary Cauchy sequence, which makes the base equal to 2. And they have described a Haskell implementation in [10]. In addition, V. Ménissier-Morain in [3] proved a similar algorithm whose base can be 2 or other integers which is bigger than 2. Briggs implemented this paper's algorithm using Python, C++, C respectively [12]. Of course, there are also many other implementations and you can get more information about exact real arithmetic in this survey [13]. But most of them are deficient in the proof or integrality of algorithms.

Actually, all these methods can be converted into interval representations. But different methods have different algorithms to calculate the operation concerning real numbers. In that case their performances are also different. In terms of basic operations, like addition, minus, multiplication and division, the second method may be more efficient, which can be found in this competence result in 2000 [14]. However, LFT approach also has some merits that it can easily handle irrational numbers [15,16], that Edalat argues its transcendental functions may more efficient, and that under special circumstances continued fractions can solve many problems which common representations is hard to handle. In this paper, our interest focus on the amelioration of the second methods, that is constructive reals. And we give new algorithms that solve two problems which can affect the implementation's performance. First, by balancing every item's precision, we can avoid unnecessary precision growth which is the first problem that has attracted many attentions in [17,18]. Second, by distributing different weights to different operations, we can make sure that complex operations do not waste most time when to compute the whole expression which is the second problem that we proposed.

The remaining part of the paper is arranged as follows. Related definitions and algorithms about Cauchy sequences are briefly introduced in section 2. Problems of present implementations based on Cauchy sequences are shown in section 3. A new simple algorithm about addition, which considers balancing every item's precision, is proved in section 4. And a new algorithm about calculating the expression that takes every operation's complexity into consideration will be given in section5. The experimental result and discussion are set in section 6.

2 Basic Definitions and Algorithms

We present some basic definitions and algorithms of computable real numbers, which derived from Boehm, V. Ménissier-Morain, and Gowland's work in [3,9,10]. First, we will show two definitions about the representation of real numbers. Second, we will give two simple algorithms about addition and multiplication that will help us understand the problem.

Definition 1. *(Effective Cauchy sequences). A computable real number x is represented as an Effective Cauchy sequence, if there is an infinite computable sequence of rational numbers* $\left\{\frac{n_0}{d_0}, \frac{n_1}{d_1}, \ldots, \frac{n_p}{d_p}, \ldots, \right\}$*, with* $d_i > 0$*, and a modulus*

of convergence function $e : N \to N$ *which is recursive , such that* $\forall p \in \mathbf{N} : k \geq e(p)$*implies* $\left|x - \frac{n_k}{d_k}\right| < 2^{-p}$.

In this definition, we use a sequence of rational numbers to represent the real number. When we need to get a number which meets the precision requirement, we can take a proper rational number from the sequence. In fact, we can get an interval which is small enough to meet the error range and includes the true value of the real number that is $x \in \left(\frac{n_k}{d_k} - 2^{-p}, \frac{n_k}{d_k} + 2^{-p}\right)$. But if we only want to use integer arithmetic to finish this kind of exact real arithmetic, we should do some changes.

Definition 2. *(Fast Binary Cauchy Sequence) A computable real number* x *is represented as a Fast Binary Cauchy Sequence if there is an infinite computable sequence of integers*$\{n_0, n_1, \ldots, n_p, \ldots\}$*, such that* $|x - 2^{-p}n_p| < 2^{-p}$.

This definition makes a little change with respect to Definition 1. We can use a sequence of integers to represent real number, which makes implementation only use integer arithmetic and is not confined by floating-point numbers' defects. Some implementations using floating point, such as IRRAM [19], which is based on real-RAMs, have faster speed. Although some real-RAMs can be realized approximately by floating point computations, real-RAMs cannot be realized by physical machines, they are unrealistic [20]. But the way of construct reals is completely accepted in that it only uses integer arithmetic. So in section 5 we do not compare time with IRRAM which use floating-point arithmetic in implementation. In this definition, if the error range is 2^{-p}, we can use n_p to represent the real number x, which meets the requirement of $x \in (2^{-p}n_p - 2^{-p}, 2^{-p}n_p + 2^{-p})$. In the rest of the article, we use $x[p]$ to represent n_p.

Also, there are other similar definitions, in[3], they use the qualification of $\left|x - \frac{n_k}{d_k}\right| < B^{-p}$ where $B \geq 2$ is some fixed integer, instead of $\left|x - \frac{n_k}{d_k}\right| < 2^{-p}$ in Definition 1. Obviously, modulating B can control the interval's and the error range. For example, if B is 10, we can get a sequence of error range:10^{-1}, 10^{-2}, ..., 10^{-p}, ...If B is 2, we can get a sequence of error range:2^{-1}, 2^{-2}, ..., 2^{-p}, ... which is more flexible than B is 10 and can easily meet more error requirements. In this paper, we only discuss our implementation based on $B = 2$ and these circumstances that $B > 2$ can be handled in the same thought.

Now we simply give some algorithms concerning addition and multiplication which is widely used in [3,9,10] and is showed in [10]. These algorithms can help us to illustrate our algorithms. Their proofs can be found in [10]. We suppose that x,x_1,x_2 are real numbers and the required error range of x is 2^{-p} which means we need to get $x[p]$.

Algorithm 1. *(Addition of real numbers) Computing* $x = x_1 + x_2$*, we have* $x[p]$*=round((*$x_1[p+2] + x_2[p+2]$*)/4).*

Algorithm 2. *(Multiplication of real numbers) Computing* $x = x_1 * x_2$*, we have* $x[p]$*=round(*$2^{-(p+s_1+s_2)}(n_1n_2)$*) where* $s_1 = \lfloor log_2(|x_1[0]| + 2)\rfloor + 3$ *and* $s_2 = \lfloor log_2(|x_2[0]| + 2)\rfloor + 3$.

3 Problems

Several authors have implemented the algorithms that are based on constructive reals. We can divide them into two kinds. One is to use Functional language which is easy and natural to implement exact real arithmetic in that we can regard a real number as a function and the expression can be evaluated automatically. However, this way is very slow and impractical in many applications. The other is to use DAG (directed acyclic graph) to represent the expression about operations between exact real numbers. For example, Figure 1 shows an expression based on DAG. This way can be finished by using C language, which can get much better performance and also is easier to use in other software. But both of them suffer from several flaws.

In the first problem, with the increase of operations' number, the original algorithms fail to use proper precise of every real number. For example, we want to compute $x = x_1 + x_2 + x_3 + x_4$ according to Algorithm 1. Its DAG is figure 1, and when we want to get $x[p]$, we need get $x_4[p+2]$, $x_3[p+4]$, $x_2[p+6]$, $x_1[p+8]$ respectively. Obviously, when the items' number is very large we may need some items that have extremely small precision. However, by scrutinizing this case carefully, we find it is not reasonable. If we consider it from a prospect of expression we find if we want to get an approximation of x up to the precision ϵ, we only need every item' approximation up to $\epsilon/4$. In this case, we should balance every item's precision to get a more proper means to compute the expression. This problem has been proposed in [17,18], which occurs in all exact real arithmetic based on Type2 theory [20]. In [18], the author proposed that each item's precision should be balanced to solve this problem. But in special representation how to solve it is also a question. In this paper we give the methods about constructive reals and give a proof in the follow section.

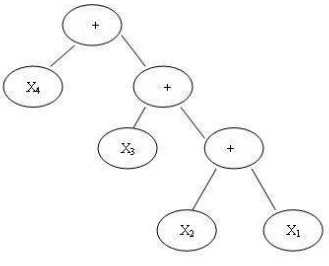


Fig. 1. A DAG for the expression $x_1 + x_2 + x_3 + x_4$

In the second problem, the original algorithm fails to take the complexity of each operation into consideration. Obviously, computing one multiplication takes more time than computing an addition when we need them to get the same precision which can be found in Algorithm 1 and Algorithm 2. The original algorithms

are not concerned about making the precision of operation of multiplication as large as possible. For instance, when we compute $x = x_1 + x_2 * x_3$, in order to get $x[p]$, a choice is to get $x_1[p+2]$ and $(x_2 * x_3)[p+2]$ which is an original algorithm's way. However, in terms of expression, we have many other ways. For example, when we want to get an approximation of x up to the precision ϵ, we also need x_1's approximation up to $\epsilon/4$ and $(x_2 * x_3)$'s approximation up to $3\epsilon/4$. In this case, the operation of addition will cost more time than the former and the operation of multiplication will cost little time than the former. But we think it is totally worthy because the multiplication is more complex.

In conclusion, original implementations do not consider the difference between exact real arithmetic and traditional computation. Traditional computation can use a common way to compute because they are not affected by error requirement. However, in exact arithmetic, distributing a proper precision to every item is very important which can make computation more effective.

4 Addition Algorithm with Balanced Precision

Addition and subtraction are very important in exact arithmetic in that summation plays an important role in science computing. And addition and subtraction's error analysis are also very easy to process. In this part we give an algorithm to compute an expression only including addition. Subtraction can be processed in a similar way in that it can be taken as a special addition.

Algorithm 3. *Computing $x = x_1 + x_2 + \ldots + x_n$, where x, x_1, x_2,..., x_n are real numbers and the required of error range of x is 2^{-p} which means we need to get $x[p]$, we have $x[p]$=round$((x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m})$. where $m = \lceil log_2 n + p + 1 \rceil$.*

Proof. We will prove that $|x - 2^{-p}x[p]| = |x_1 + x_2 + \ldots + x_n - 2^{-p}x[p]| < 2^{-p}$ i.e.

$$2^{-p}(x[p] - 1) < x_1 + x_2 + \ldots + x_n < 2^{-p}(x[p] + 1) \tag{1}$$

According to definition 2, we have

$$|x - (x_1[m] + x_2[m] + \ldots + x_n[m])2^{-m}|, n2^{-m} \tag{2}$$

We make $m = \lceil log_2 n + p + 1 \rceil$ such that

$$\frac{1}{2^m} \leq \frac{1}{n2^{p+1}} < \frac{1}{2^{m-1}} \tag{3}$$

According to (2) and (3), we get
$|x - (x_1[m] + x_2[m] + \ldots + x_n[m])2^{-m}| < n2^{-m} \leq 2^{-(p+1)}$ and

$$|x - (x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m}2^{-p}| < 2^{-(p+1)} \tag{4}$$

Then according to round's property, we have

$$|(x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m} - round((x_1[m] + x_2[m] + \ldots + x_n[m]) 2^{p-m})| \leq \frac{1}{2} \tag{5}$$

By multiplying 2^{-p}, Formula (5) can be changed into:

$$|(x_1[m] + x_2[m] + \ldots + x_n[m])2^{-m} - round((x_1[m] + x_2[m] + \ldots + x_n[m]) 2^{p-m})2^{-p})| \leq 2^{-(p+1)} \tag{6}$$

Then we can use $round((x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m})$ to replace $(x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m}$ in Formula (4), we can get

$$|x - round((x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m})2^{-p}| < 2^{-(p+1)} + 2^{-(p+1)} = 2^{-p} \tag{7}$$

i.e.

$$|x - 2^{-p}x[p]| = |x_1 + x_2 + \ldots + x_n - 2^{-p}x[p]| < 2^{-p} \tag{8}$$

In this addition algorithm, we distribute the same precision to every item. For example, if we want to get $x[p]$ whose expression is shown in Figure 1, we only need get $x_1[p+3]$, $x_2[p+3]$, $x_3[p+3]$ and $x_4[p+3]$ respectively. When n is a very large number this way can evidently reduce many items' precision required. Compared with the original algorithm, this way is more proper in terms of expression.

5 Algorithm with Precision Control

In section 4, we have solved the first problem that exits in constructive reals' computation. In this section we will consider solving the second problem. From what have been discussed, we can see that different operations have different complexity. A sin's operation is more complex than addition operation when is computed to up to the same precision. Our solution is to distribute different weight to single computation cell. The weight can control the precision that operation needs. First, we introduce a definition of single computation cell.

Definition 3. *(Single Computation Cell) A single computation cell is one of these cases: 1. A real number that interacts with other items only with addition or subtraction. 2. A result number that we get from operations except addition and subtraction and it interacts with other SCCs only with addition or subtraction.*

For example, when we want to deal with $x_1 + x_2 * x_3 + \sin x_4 + x_5 * x_6/x_7$, the number x_1, the result of $x_2 * x_3$, the result of $\sin x_4$ and the result of $x_5 * x_6/x_7$ are SCCs. We can simply put x_1's weight is 1, $x_2 * x_3$ is 4, $\sin x_4$ is 16 and

$x_5 * x_6/x_7$ is 8. The value of weight is based on the compute complexity of the SCC and is regulated to be the form of 2^k, $k \in N$, which make our computation convenient. We can compute single SCC use its original algorithm, but compute the whole expression use our algorithm. Now we will show an algorithm that can used to compute these expressions.

Algorithm 4. *If we want to compute* $x = x_1+x_2+\ldots+x_n$ *in order to meet some error range* 2^{-p}*, where* x_i*'s weight is* w_i *and every* x_i *is a SCC,* $i = 1, 2, \ldots, n$*. We have* $x[p] = round(x_1[m - log_2 w_1]w_1 2^{p-m} + x_2[m - log_2 w_2]w_2 2^{p-m} + \ldots + x_n[m - log_2 w_n]w_n 2^{p-m})$ *with* $m = \lceil -log_2(\frac{2^{-(p+1)}}{w_1+w_2+\ldots+w_n})\rceil$.

Proof. We will prove

$$|x - 2^{-p}x[p]| = |x_1 + x_2 + \ldots + x_n - 2^{-p}x[p]| < 2^{-p} \tag{1}$$

We make $m = \lceil -log_2(\frac{2^{-(p+1)}}{w_1+w_2+\ldots+w_n})\rceil$ such that

$$(w_1 + w_2 + \ldots + w_n)2^{-m} \leq 2^{-(p+1)} \tag{2}$$

According to definition 2, we have

$$\begin{aligned} &|x - (x_1[m - log_2 w_1]2^{log_2 w_1 - m} + x_2[m - log_2 w_2]2^{log_2 w_2 - m} + \ldots + x_n[m - \\ &log_2 w_n]2^{log_2 w_n - m})| = |x - (x_1[m - log_2 w_1]w_1 2^{-m} + x_2[m - log_2 w_2]w_2 2^{-m} \\ &+ \ldots + x_n[m - log_2 w_n]w_n 2^{-m})| < (w_1 + w_2 + \ldots + w_n)2^{-m} \leq 2^{-(p+1)} \end{aligned} \tag{3}$$

It is similar to the addition's proof above, we can get

$$\begin{aligned} &|(x_1[m - log_2 w_1]w_1 2^{-m} + x_2[m - log_2 w_2]w_2 2^{-m} + \ldots + x_n[m - log_2 w_n] \\ &w_n 2^{-m}) - round(x_1[m - log_2 w_1]w_1 2^{p-m} + x_2[m - log_2 w_2]w_2 2^{p-m} + \ldots + \\ &x_n[m - log_2 w_n]w_n 2^{p-m}) < 2^{-(p+1)} \end{aligned} \tag{4}$$

According to (3) and (4), we can get $|x - round(x_1[m - log_2 w_1]w_1 2^{p-m} + x_2[m - log_2 w_2]w_2 2^{p-m} + \ldots + x_n[m - log_2 w_n]w_n 2^{p-m})| < 2^{-(p+1)} + 2^{-(p+1)} = 2^{-p}$ that is to say $|x - x[p]| < 2^{-p}$.

For example, when we want to compute $x = x_1 + y_1 * y_2 + x_3 + x_4 + x_5$ that is $x = x_1 + x_2 + x_3 + x_4 + x_5$, where $x_2 = y_1 * y_2$ and $x_1, x_2, x_3, x_4, x_5$are SCCs. We can appoint x_1, x_3, x_4, x_5's weight to be 1 and x_2's weight to be 4. Then when we need to get $x[p]$, we should compute $x_1[p + 4]$, $x_2[p + 2]$, $x_3[p + 4]$, $x_4[p + 4]$, $x_5[p + 4]$ which is more efficient than compute $x_1[p + 4]$, $x_2[p + 4]$, $x_3[p + 4]$, $x_4[p + 4]$, $x_5[p + 4]$ that only use algorithm 3 to balance the precision. According to Algorithm 2 we can get $x_2[p + 2]$, and then we use Algorithm 4 to compute the whole expression.

6 Conclusions and Discussions

One of our aim is to avoid the unnecessary precision growth which has been talked in [17,18], in this paper we supply Algorithm 3 to balance every item in an expression including addition and subtraction. This way indeed improves exact arithmetic's speed. For example, we will do a computation that gets the sum of the harmonics series $\sum_{i=1}^{n} \frac{1}{i}$. The Table 1 below shows Algorithm 3's performance is much better than xrc1.1 which is an implementation based on V.Mnissier-Morain's Algorithm [3]. The software xrc1.1 can be gotten from [21].

Table 1. Computing $\sum_{i=1}^{n} \frac{1}{i}$

n	result'sprecision	xrc1.1(ms)	Algorithm 3 (ms)
1000	100	9.641	4.92
1000	1000	77.377	10.948
1000	10000	168.708	161.989
5000	100	141.016	58.564
5000	1000	187.183	106.209
5000	10000	455.335	364.182
10000	100	373.373	81.788
10000	1000	446.627	150.112
10000	10000	956.082	619.227

Table 2. Computing $\sum_{i=1}^{n} (\frac{1}{i*(i+1)} + \frac{1}{i})$

n	result'sprecision	xrc1.1(ms)	Algorithm 3 (ms)	Algorithm 4 (ms)
100	100	3.358	4.909	2.901
100	1000	11.811	10.911	10.256
1000	100	162.803	81.511	65.246
1000	1000	220.861	150.396	133.057
10000	100	19166.903	324.03	288.016
10000	1000	21070.736	900.51	894.279

But our way can only solve some of this problem. For example, $x_1 * x_2 * x_3$ also should have a way to make sure that every item has the similar precision, but because the result of multiplication's interval is too complex we failed to get an easy means just like we compute addition or subtraction. In the future, we may find a way to completely solve this problem.

Another aim is to take into account the operation's complexity which is first introduced by us. We control every item's precision according to its complexity. In this way, we can improve the performance of the exact arithmetic furthermore. For example, we will do a computation that gets the value of $\sum_{i=1}^{n} (\frac{1}{i*(i+1)} + \frac{1}{i})$. The Table 2 shows that Algorithm 4 has a better performance than xrc1.1 and Algorithm 3.

There is also a question in this algorithm that how large the weight should be. In our implementation, we can easily appoint it by ourselves according to test many times basic operations and we also can adjust it according to need.

Acknowledgements. The research was supported by Chinese 973 Program (2004CB719400), and the National Science Foundation of China (60403047, 60533070). The second author was supported by the project sponsored by a Foundation for the Author of National Excellent Doctoral Dissertation of PR China (200342), and a Program for New Century Excellent Talents in University (NCET-04-0088).

References

1. A. Edalat, R. Heckmann, Computing with real numbers: (i) LFT approach to real computation, (ii) Domain-theoretic model of computational geometry, Lecture Notes in Computer Science, vol.2395, Springer, 2002, pp.193-267.
2. J. Blanck, Efficient exact computation of iterated maps, The Journal of Logic and Algebraic Programming, vol.64, 2005, pp. 41-59.
3. V. Mnissier-Morain, Arbitrary precision real arithmetic: design and algorithms. The Journal of Logic and Algebraic Programming, Vol.64, 2005, pp. 13-19.
4. W. Gosper, Continued fraction arithmetics, Technical Report HAKMEM Item 101B, Artificial Intelligence Memo 239, MIT, 1972.
5. Jen Vuillemin, Exact real computer arithmetic with continued fractions, IEEE Transactions on Computers, Vol. 39, 1990, pp. 1087-1105.
6. E.Bishop, D.Bridges, Constructive Analysis, Springer-Verlag, 1985.
7. Hans-J. Boehm, Robert Cartwright, Mark riggle, and Michael J.O'Donnell, Exact real arithmetic:A case study in higher order programming, In Proceedings of the 1986 Lisp and Functional Programming Conference, 1986, pp. 162-173.
8. V. Lee, Optimizing Programs over the Constructive Reals, PhD thesis, Rice University, 1991.
9. Hans-J. Boehm The constructive reals as a Java library, The Journal of Logic and Algebraic Programming, vol.64, 2005, pp. 3-11.
10. P. Gowland, D. Lester, The correctness of an implementation of exact arithmetic, in: Proceedings of the Fourth Conference on Real Numbers and Computers, 2000.
11. D. Lester, P. Gowland, Using PVS to validate the algorithms of an exact arithmetic, Theoretical Computer Science, Vol. 291, 2003, pp. 203-218.
12. K. Briggs, Implementing exact real arithmetic in python, C++ and C, Theoretical Computer Science, vol.351, 2006, pp. 74-81.
13. P. Gowland, D. Lester A survey of exact arithmetic implementations, Lecture Notes in Computer Science, vol.2064, Springer, 2001, pp. 30-47.
14. J. Blanck, Exact real arithmetic systems:Results of competition, Lecture Notes in Computer Science, vol.2064, Springer, 2001, pp.389.
15. A. Edalat, P. J. Potts, A new representation for exact real numbers, Electronic Notes in Theoretical Computer Science, Vol.6, 1997, pp. 119-132.
16. P.J. Potts, Exact real arithmetic using M?bius transformations, PhD thesis, Imperial College, 1999.
17. Branimir Lambov, RealLib:An Efficient Implementation of Exact Real Arithmetic, http://www.bric.dk/ barnie/RealLib/, 2006.

18. Joris van der Hoeven, Computations with effective real numbers, Theoretical Computer Science, vol.351, 2006, pp. 52-60.
19. Norbert Th. Mller, The iRRAM: Exact Arithmetic in C++. Lecture Notes in Computer Science, vol.2064, Springer, 2001, pp. 222-252.
20. K. Weihrauch, An Introduction to Computable Analysis, Springer, 2000.
21. K. Briggs, xrc homepage, http://keithbriggs.info/xrc.html, 2005.

Bounding Run-Times of Local Adiabatic Algorithms

M.V. Panduranga Rao

Department of Computer Science and Automation
Indian Institute of Science
Bangalore
India
pandurang@csa.iisc.ernet.in

Abstract. A common trick for designing faster quantum adiabatic algorithms is to apply the adiabaticity condition locally at every instant. However it is often difficult to determine the instantaneous gap between the lowest two eigenvalues, which is an essential ingredient in the adiabaticity condition. In this paper we present a simple linear algebraic technique for obtaining a lower bound on the instantaneous gap even in such a situation. As an illustration, we investigate the adiabatic unordered search of van Dam et al. [17] and Roland and Cerf [15] when the non-zero entries of the diagonal final Hamiltonian are perturbed by a polynomial (in $\log N$, where N is the length of the unordered list) amount. We use our technique to derive a bound on the running time of a local adiabatic schedule in terms of the minimum gap between the lowest two eigenvalues.

1 Introduction

Adiabatic Quantum Computation (AQC) has attracted a lot of interest in recent times. First introduced by Farhi et al. [11], this paradigm of computing makes use of the adiabatic theorem of quantum mechanics. Informally, the adiabatic theorem says that if a physical system is in the ground state of an initial Hamiltonian that evolves "slowly enough" to a final Hamiltonian, with a non-zero gap between the ground state and the first excited state of the Hamiltonian at all times, then the probability that the system ends up in the ground state of the final Hamiltonian approaches unity as the total time of evolution tends to infinity. This fact is used for solving computational problems as follows. To begin with, the system is in the ground state of a suitable Hamiltonian. This initial Hamiltonian is slowly evolved to a final Hamiltonian whose ground state represents the solution to the problem. If the running time required for a high probability of reaching the ground state of the final Hamiltonian is at most polynomial in the size of the input, we have an efficient AQC algorithm for the problem. The maximum rate at which the Hamiltonian can evolve at any instant without violating the adiabaticity condition depends inversely on square of the gap between

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 450–461, 2007.
© Springer-Verlag Berlin Heidelberg 2007

the two instantaneous lowest eigenvalues. In many cases, it is difficult to estimate the gap at every instant of the evolution. Then, we strike a compromise by imposing a constant *global* delay schedule determined conservatively by the minimum gap over the entire interval. However, if we do have an estimate of this gap for the entire duration, we can apply the adiabaticity condition *locally* and speed up the rate of evolution wherever possible.

In the early days of AQC, efforts were largely focused on adiabatic optimization algorithms. Given a function $\phi : \{0,1\}^n \to \mathbb{R}$, the problem is to find an $x \in \{0,1\}^n$ that minimizes ϕ.

van Dam et al. [17] and Roland and Cerf [15] demonstrated the quantum nature of AQC by showing a quadratic speed-up for unordered search, matching Grover's discrete algorithm [12].

However, any adiabatic quantum computer can be efficiently simulated by a standard quantum computer [17]. Applications of AQC techniques to small sized random instances of several NP complete problems like EXACT-COVER [8,9], CLIQUES [6] and SAT [8,9,11] have been explored with some success. Aharonov et al. [1] generalized this model by considering local[1] final Hamiltonians instead of only diagonal ones and showed that this generalized model can efficiently simulate any discrete quantum computation. Thus, in this general sense, AQC is equivalent to discrete quantum computation.

Adiabatic quantum computation is particularly interesting because of indications that it is more resilient to decoherence and implementation errors than the discrete model. While most schemes for implementing AQC oracles involve approximating them by a sequence of discrete unitary gates [3,17], robustness of potential implementations of the time-dependent Hamiltonians has also been studied. For example, Childs et al. [7] considered errors due to environmental decoherence and imperfect implementation in the latter approach. If the time dependent algorithm Hamiltonian is $H(t)$, they considered the actual Hamiltonian to be $H(t) + K(t)$, where $K(t)$ is an error Hamiltonian. In particular, $K(t)$ can be a perturbation in the final Hamiltonian. Through numerical simulations, they demonstrated robustness of AQC for small instances of combinatorial search problems against such errors. Åberg et al. [4,5] investigated robustness to decoherence in the instantaneous eigenvectors for local [5] and global adiabatic quantum search [4]. They showed that as long as Hamiltonian dynamics is present, asymptotic time complexity is preserved in both local and global cases. However, in case of pure decoherence, the time complexity of local search climbs to that of classical. For global evolution, it becomes costlier than classical: $N^{\frac{3}{2}}$, where N is the size of the list.

In this paper, we show how to use simple results from linear algebra to obtain bounds on the running time of algorithms that obey the adiabaticity condition locally. The technique is useful when the gap between the two lowest eigenvalues is not known at every instant of the evolution. As an illustration and running example, we investigate the behaviour of the eigenvalue spectrum when the final

[1] A Hamiltonian is said to be local if it involves interactions between only a constant number of particles.

oracle Hamiltonian for the unordered search problem is perturbed in all non-zero elements by an amount at most polynomial in $\log N$.

In the unperturbed case, the gap between the lowest two eigenvalues is specified at every instant by a nice closed form expression [15,17]. This makes it rather easy to apply local adiabaticity. Perturbation deprives us of this facility. We show a work-around by lower-bounding the gap between the two lowest eigenvalue curves with straight lines whose slopes are obtained using the Wielandt-Hoffman theorem from the eigenvalue perturbation theory of symmetric matrices. The schedule can then be adjusted to satisfy the adiabaticity condition locally. The bound obtained by our method is commensurate with existing results–we obtain only a polynomial speed-up over the global algorithm [10,16,18,19]. Tighter bounds on eigenvalue perturbations will possibly yield Grover speed-up, indicating the resilience of the adiabatic algorithm to perturbations in the final Hamiltonian. However, we believe that the techniques that we introduce in this paper can be applied in other AQC settings as well.

The paper is arranged as follows. The next section gives a brief discussion of the adiabatic quantum computing paradigm. Section 3 discusses the adiabatic search problem and its perturbed version. In section 4.1 we show that for the Hamiltonian in question, there exists a non-zero gap between the two lowest eigenvalues at all times. The proof largely follows that of Rao [14], simplified for the present case. However the minimum gap turns out to be exponentially small, and therefore a global schedule is of limited value. In section 4.2 we show our method to obtain a local schedule that provides a polynomial speed-up. Section 5 concludes the paper.

2 Preliminaries

In this section we give a brief overview of the adiabatic gap theorem and its application to quantum computing. For details the reader is referred to the text by Messiah [13]. Let $H(s)$, $(0 \leq s \leq 1)$, be a time dependent single-parameter Hamiltonian for a system having an N dimensional Hilbert space. Let the eigenstates of $H(s)$ be given by $|l;s\rangle$ and the eigenvalues by $\lambda_l(s)$, with $\lambda_0(s) \leq \lambda_1(s) \leq \ldots \leq \lambda_{N-1}(s)$ for $0 \leq s \leq 1$. Suppose we start with the initial state of the system $|\psi(0)\rangle$ as $|0;0\rangle$ (the ground state of $H(0)$) and apply the Hamiltonian $H(s)$, $0 \leq s \leq 1$, to evolve it to $|\psi(1)\rangle$ at $s = 1$. Then the quantum adiabatic theorem states that for a "large enough" delay, the final state of the system $|\psi(1)\rangle$ will be arbitrarily close to the ground state $|0;1\rangle$ of $H(1)$. Specifically, $|\langle 0;1|\psi(1)\rangle|^2 \rightarrow 1$ if the delay schedule $\tau(s)$ satisfies the adiabaticity condition at every s:

$$\tau(s) \gg \frac{||\frac{d}{ds}H(s)||_2}{g(s)^2}, \tag{1}$$

where $g(s)$ is the gap between the two lowest eigenvalues at s. In case $g(s)$ is difficult to determine for every s, which is the case with most Hamiltonians, we impose a more conservative *global* delay schedule for the entire duration T of the evolution:

$$T \gg \frac{\max_{0\le s\le 1} ||\frac{d}{ds}H(s)||_2}{g_{min}^2}, \tag{2}$$

where $g_{min} = \min_{0\le s\le 1}(\lambda_1(s) - \lambda_0(s))$. However, if we do have an estimate of $g(s)$ for every s, we can use a varying delay schedule that satisfies the adiabaticity condition *locally* at every instant $0 \le s \le 1$. Then we can reduce the running time to

$$\int_{s=0}^{1} \frac{||\frac{d}{ds}H(s)||_2 ds}{g(s)^2}. \tag{3}$$

If $H(s)$ is a polynomial-sized linear interpolation, $||\frac{d}{ds}H(s)||_2$ is a polynomial-sized quantity independent of s and the running time is of the order $\int_{s=0}^{1} \frac{ds}{g(s)^2}$.

3 Perturbed Unordered Search

Consider a list of $N = 2^n$ elements, say bit-strings from $\{0,1\}^n$. The unordered search problem may be stated as follows. Given a function $g : \{0,1\}^n \to \{0,1\}$ such that $g(u) = 0$ for a special element u and 1 for all others, find u. While any classical algorithm requires $O(N)$ queries to g, Grover's celebrated "discrete" quantum algorithm accomplishes the search in $O(\sqrt{N})$ queries only [12]. A similar speed-up was demonstrated by van Dam et al. [17] and Roland and Cerf [15] for this problem with AQC, which we discuss now. To begin with, note that the initial Hamiltonian $H(0)$ should be independent of the solution, with the restriction that it should not be diagonal in the computational basis [17]. Moreover, the ground state of the initial Hamiltonian should be a uniform superposition of all candidate solutions and easy to prepare. The following Hamiltonian satisfies the above conditions for searching an unordered list [17,15]:

$$H(0) = \sum_{z\in\{0,1\}^n \setminus 0^n} |\hat{z}\rangle\langle\hat{z}|, \tag{4}$$

where each $|\hat{z}\rangle$ is a basis vector in the 'Hadamard' basis given by

$$|\hat{z}\rangle = \frac{1}{\sqrt{2^n}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{\otimes n} |z\rangle$$

and the ground state is $\frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} |z\rangle$, which is easy to construct. The final Hamiltonian, then, is $\sum_{z\in\{0,1\}^n \setminus u} |z\rangle\langle z|$.

In this paper, we consider the case when the final Hamiltonian is perturbed in the non-zero entries. In other words, the final Hamiltonian is given by

$$H(1) = \sum_{z\in\{0,1\}^n} f(z)|z\rangle\langle z|,$$

where $\{|z\rangle\}$ form the "computational basis" of the Hilbert space of the system and $f : \{0,1\}^n \to \mathbb{R}_{poly(n)}$ is a function that behaves as follows. $f(u) = 0$ for a

special element $u \in \{0,1\}^n$. For all other elements z, $f(z) > 0$. The problem is to minimize f, that is, to find u.

Given the initial and final Hamiltonians, define the interpolating Hamiltonian as

$$H(s) = (1-s)H(0) + sH(1). \tag{5}$$

We start in the ground state of $H(0)$ and evolve to $H(1)$ slowly enough and end up in its ground state. Since $f(z)$ is bounded by a polynomial in n for all $z \in \{0,1\}^n$, so is $||\frac{d}{ds}H(s)||_2$. The factor deciding the running time is therefore the denominator g_{min}^2.

4 The Bounds

In what follows, we will assume $f(z) \neq f(z')$ for $z \neq z'$. The motivation for this assumption is two-fold: first, it makes for a neater presentation of our technique; secondly, given the random nature of noise and implementational error, it is reasonable to assume that no two non-zero elements of the diagonal will be perturbed by the same amount. For the case when there do exist $z \neq z'$ such that $f(z) = f(z')$, the subsequent discussion requires only minor modification.

4.1 Global Evolution

By the nature of $H(s)$, the characteristic equation is independent of the permutations of the diagonal elements of the final Hamiltonian. Therefore, without loss of generality, we will follow the convention that $f(u) = 0 < f(z_1) < \ldots < f(z_{N-1})$. First, we make sure that the gap between the two lowest eigenvalues is non-zero at all times. For that, we evaluate the characteristic equation of $H(s)$, in much the same way as in lemma 1 of [14], and [19].

Lemma 1. *The characteristic equation of $H(s)$ is*

$$c(\lambda) = (1-s-\lambda)\left[\prod_{k=1}^{N-1}(1-s+sf(z_k)-\lambda) - \frac{1-s}{N}\sum_{j=1}^{N-1}\frac{\prod_{k=1}^{N-1}\left(1-s+sf(z_k)-\lambda\right)}{1-s+sf(z_j)-\lambda}\right] - \frac{1-s}{N}\prod_{k=1}^{N-1}\left(1-s+sf(z_k)-\lambda\right) = 0.$$

Proof. To evaluate the eigenvalue curves, we evaluate $|H(s) - \lambda I| = 0$. Subtracting the last column of this determinant from all other columns and using x_0 for $1-s-\lambda$, x_1 for $1-s+sf(z_1)-\lambda$ and so on, up to x_{N-1} for $1-s+sf(z_{N-1})-\lambda$, we have

$$\begin{vmatrix} x_0 & 0 & 0 \ldots & \ldots & -\frac{(1-s)}{N} \\ 0 & x_1 & 0 \ldots & \ldots & -\frac{(1-s)}{N} \\ \vdots & 0 & \ddots \vdots & x_{\frac{N}{2}-1} & \vdots \\ -x_{N-1} & -x_{N-1} & \ldots & -x_{N-1} & (x_{N-1} - \frac{(1-s)}{N}) \end{vmatrix}_{N\times N} = 0.$$

Expanding this determinant gives the required characteristic equation. □

The following lemma is reminiscent of lemma 2 of [14], and [19]. We provide a somewhat simpler proof for the present case.

Lemma 2. *There exists exactly one root (i.e. an eigenvalue curve) of the characteristic equation in the intervals* $(0, 1-s)$, $(1-s, 1-s+sf(z_1))$, ..., $(1-s+sf(z_{N-2}), 1-s+sf(z_{N-1}))$; *for* $0 < s < 1$.

Proof. For any interval, let the $c_l(\lambda)$ and $c_u(\lambda)$ denote the value of the characteristic polynomial at the lower and upper boundary respectively.

We analyze the problem as three cases:

(1) The interval $(0, 1-s)$:

In this case, $c_l(\lambda)=(1-s)\left[\prod_{k=1}^{N-1}(1-s+sf(z_k))-\frac{1-s}{N}\sum_{j=1}^{N-1}\frac{\prod_{k=1}^{N-1}\left(1-s+sf(z_k)\right)}{1-s+sf(z_j)}\right]-\frac{1-s}{N}\prod_{k=1}^{N-1}\left(1-s+sf(z_k)\right)$. This is a positive quantity for $s \in (0,1)$, as can easily be verified. Moreover, $c_u(\lambda) = -\frac{1-s}{N}\prod_{k=1}^{N-1} sf(z_k)$ is negative. Therefore, there exists a root in the interval $(0, 1-s)$ for $0 < s < 1$.

(2) The interval $(1-s, 1-s+sf(z_1))$:

Clearly, $c_l(\lambda)$ for this interval is same as $c_u(\lambda)$ of the previous case, which is negative. But $c_u(\lambda) = -sf(z_1)\left[-\frac{1-s}{N}\prod_{k=2}^{N-1} s(f(z_k)-f(z_1))\right]$ is positive. Thus, there exists at least one root in this interval also.

(3) The intervals $(1-s+sf(z_i), 1-s+sf(z_{i+1}))$, $1 \le i \le N-2$:

The values of the characteristic polynomial at the boundaries are respectively

$c_l(\lambda) = -sf(z_i)\left[-\frac{1-s}{N}\prod_{k=1}^{i-1} s(f(z_k)-f(z_i))\prod_{k=i+1}^{N-1} s(f(z_k)-f(z_i))\right]$ and

$c_u(\lambda) = -sf(z_{i+1})\left[-\frac{1-s}{N}\prod_{k=1}^{i} s(f(z_k)-f(z_{i+1}))\prod_{k=i+2}^{N-1} s(f(z_k)-f(z_{i+1}))\right]$.

Notice that in either case, every element in the first product is negative and that in the second is positive. This is because $f(z_k) < f(z_i)$ for $1 \le k \le i-1$ and $f(z_k) > f(z_i)$ for $i+1 \le k \le N-1$. Therefore, the over-all sign is decided by the number of elements in the first product only. But the number of such elements in $c_l(\lambda)$ and $c_l(\lambda)$ differ by one. Thus $c_l(\lambda)$ and $c_u(\lambda)$ differ in sign. Therefore, there exists a root in each of these intervals.

Hence, there is a root in each interval. Given that (i) there are N open intervals bounded by N straight line eigenvalue curves, (ii) there is at least one root in each interval and (iii) there are N roots in all, the lemma follows. □

By virtue of the above lemma, we can speak of one curve being "above" another. We label the curves as $\lambda_0(s), \lambda_1(s), \ldots, \lambda_{N-1}(s)$, starting from below. Thus, $\lambda_0(s)$ lies between the lines $\lambda(s) = 0$ and $\lambda_1(s)$. See figure 1 for example.

As a consequence of the above lemma, we are guaranteed a non-zero gap between $\lambda_0(s)$ and $\lambda_1(s)$ for all $s \in [0,1]$.

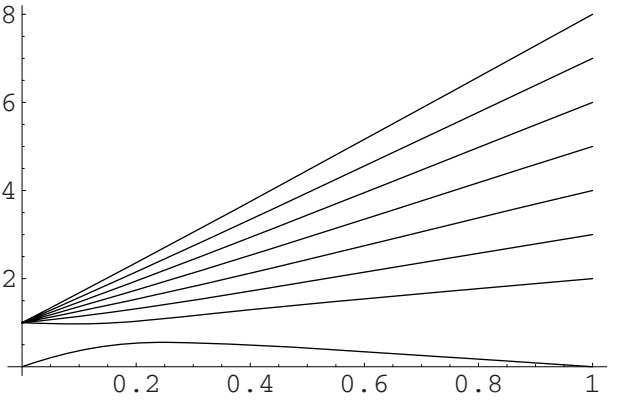


Fig. 1. Example Eigenvalue Curves

What about the minimum gap between $\lambda_0(s)$ and $\lambda_1(s)$? It turns out that this gap is exponentially small [19]. Making use of the properties of the present problem, we arrive at this conclusion in a different manner.

We first discuss the case when $f(z_1)$ is large, say greater than 1. Consider the characteristic polynomial $c(\lambda) = (1-s-\lambda)\left[\prod_{k=1}^{N-1}(1-s+sf(z_k)-\lambda) - \frac{1-s}{N}\sum_{j=1}^{N-1}\frac{\prod_{k=1}^{N-1}\left(1-s+sf(z_k)-\lambda\right)}{1-s+sf(z_j)-\lambda}\right] - \frac{1-s}{N}\prod_{k=1}^{N-1}\left(1-s+sf(z_k)-\lambda\right)$.

By lemma 2, the line $\lambda(s) = 1-s$ separates λ_0 and λ_1. Thus, if

(i) there exists an s_0 where the above polynomial is divisible by two "$(1-s_0-\lambda)$ factors" and

(ii) all other curves ($\lambda_2(s)$ through $\lambda_{N-1}(s)$) are at least an inverse polynomial distance above the point $(s_0, 1-s_0)$,

it would imply that $\lambda_0(s_0)$ and $\lambda_1(s_0)$ are exponentially close to the line $\lambda(s) = 1-s$, and in turn, to each other.

Clearly, points very close to $s=0$ are not candidates, as there are too many curves in the vicinity. Points close to $s=1$ are also ruled out, as there is only one $(1-s-\lambda)$ factor, corresponding to the curve $\lambda_0(s)$: $\lambda_1(s)$ is $f(z_1)$ units above.

Consider a candidate point $s_0 = \frac{1}{p(n)}$ where $p(n)$ is a polynomial in n such that $p(n) > max_k(f(z_k))$. At $s_0 = \frac{1}{p(n)}$, $\lambda_2 > 1 + \frac{f(z_1)-1}{p(n)}$. Thus, by lemma 2, all curves above $\lambda_2(s)$ are at least an inverse polynomial distance above $1-s_0 = 1-\frac{1}{p(n)}$.

Notice that the polynomial can be divided by one $(1-s-\lambda)$ factor at an s such that $\frac{1-s}{N}\prod_{k=1}^{N-1}\left(1-s+sf(z_k)-\lambda\right) \to 0$. Let us test the candidate point s_0. Substituting $1-s$ for λ and $\frac{1}{p(n)}$ for s, we get $\frac{p(n)-1}{Np(n)}\prod_{k=1}^{N-1}\left(\frac{f(z_k)}{p(n)}\right)$, which indeed tends to zero. Therefore, the existence of one $(1-s-\lambda)$ factor at s_0

has been established. Let us now see if there exists another. On dividing the polynomial by $(1 - s_0 - \lambda)$ at $s_0 = \frac{1}{p(n)}$, we obtain $c(\lambda) = \left[\prod_{k=1}^{N-1} \frac{f(z_k)}{p(n)} - \frac{p(n)-1}{Np(n)} \sum_{j=1}^{N-1} \frac{\prod_{k=1}^{N-1} \frac{f(z_k)}{p(n)}}{\frac{f(z_j)}{p(n)}} \right]$, which tends to zero.

Suppose now that $f(z_1)$ is "small". If $f(z_1)$ is a constant independent of n or even $O(\frac{1}{poly(n)})$, the argument given above holds exactly. However, if $f(z_1)$ is greater than zero by only an exponentially small quantity, the two $(1 - s - \lambda)$ factors that we obtained in the above discussion could correspond to $\lambda_2(s)$, thus leaving the possibility of $\lambda_0(s)$ being at a larger distance below $\lambda(s) = 1 - s$. But this is not the case. Since by assumption $\lambda_2(s)$ is exponentially close to the line $\lambda(s) = 1 - s$, it can be factored out from the polynomial, leaving two factors that correspond to $\lambda_0(s)$ and $\lambda_1(s)$.

Thus, it turns out that the minimum gap is too small to yield any significant speed-up for a constant rate global adiabatic algorithm.

4.2 Local Evolution

We will now obtain an upper bound on $\int_{s=0}^{1} \frac{d(s)}{g(s)^2}$ where $g(s) = \lambda_1(s) - \lambda_0(s)$, for local evolution. To that end, the following result from the perturbation theory of real symmetric matrices will be useful.

Theorem 1. *Wielandt-Hoffman Theorem (WHT): Let A and E be real symmetric matrices and let $\hat{A} = A + E$. Let the eigenvalues of A be $\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{N-1}$ and those of $\hat{A}$ be $\hat{\lambda}_0 \leq \hat{\lambda}_1 \leq \ldots \leq \hat{\lambda}_{N-1}$.*
Then

$$\sum_{j=0}^{N-1} (\lambda_j - \hat{\lambda}_j)^2 \leq \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |E_{ij}|^2. \tag{6}$$

□

Let E of WHT be the 'perturbation matrix' $H(s + ds) - H(s)$. This provides us with the r.h.s. of (6):

$$\Big(\frac{N-1}{N}\Big)^2 (ds)^2 + \frac{N-1}{N}(ds)^2 + \sum_{k=1}^{N-1} \Big(f(z_k) - \frac{N-1}{N}\Big)^2 (ds)^2.$$

For the l.h.s., note that $\lambda_1(s) \ldots \lambda_{N-1}(s)$ are $N - 1$ non-crossing curves packed inside a polynomially bounded gap and each curve is bounded by two straight lines by lemma 2.

Thus, there are at most a polynomial $q(n)$ number of the gaps $f(z_i) - f(z_{i-1})$ that are at least inverse polynomially wide. All other curves are packed between the straight lines $\lambda = (f(z_i) - 1)s + 1$ and $\lambda = (f(z_{i-1}) - 1)s + 1$ where $f(z_i) - f(z_{i-1}) = O(\frac{n^c}{N})$ for a constant c. Therefore the slopes of these curves can be approximated by the slopes of one of the enclosing lines. Hence, the l.h.s. is

$$d\lambda_0^2 + d\lambda_{i_1}^2 + \ldots + d\lambda_{i_{q(n)}}^2 + \sum_{k\in\{1,\ldots,N-1\}\setminus\{i_1,\ldots,i_{q(n)}\}} (f(z_k)-1)^2(ds)^2.$$

Substituting in 6, we get an upper bound on $d\lambda_0^2 + d\lambda_{i_1}^2 + \ldots + d\lambda_{i_{q(n)}}^2$. In particular, this is also an upper bound on $\frac{d\lambda_0}{ds}$ and $\frac{d\lambda_1}{ds}$.[2] This can be used to obtain an upper bound on the total time required, as we will see soon.

We illustrate the technique for the case when $q(n) = 1$ and $f(z_1) \geq 1$.

Lemma 3. *If* $q(n) = 1$ *and* $f(z_1) \geq 1$,

$$\left(\frac{d\lambda_0(s)}{ds}\right)^2 + \left(\frac{d\lambda_1(s)}{ds}\right)^2 \leq (f(z_{N-1})-1)^2 + 2n + \frac{2}{N}\sum_{k=1}^{N-2} f(z_k). \qquad (7)$$

Proof. By the preceding argument, the l.h.s. of 6 is given by

$\sum_{k=1}^{N-2}(f(z_k)-1)^2(ds)^2 + (d\lambda_0(s))^2 + (d\lambda_1(s))^2$. Substituting in equation 6, we get

$\sum_{k=1}^{N-2}(f(z_k)-1)^2(ds)^2 + (d\lambda_0(s))^2 + (d\lambda_1(s))^2$
$\leq (\frac{N-1}{N})^2(ds)^2 + \frac{N-1}{N}(ds)^2 + \sum_{k=1}^{N-1}(f(z_k) - \frac{N-1}{N})^2(ds)^2$.
Rearranging some terms we get,
$(\frac{d\lambda_0}{ds})^2 + (\frac{d\lambda_1}{ds})^2 \leq (\frac{N-1}{N})^2 + \frac{N-1}{N} + (f(z_{N-1}) - \frac{N-1}{N})^2$
$+ \sum_{k=1}^{N-2}\left((f(z_k) - \frac{N-1}{N})^2 - (f(z_k)s - \frac{N-1}{N})^2\right)^2$.
Or,
$(\frac{d\lambda_0}{ds})^2 + (\frac{d\lambda_1}{ds})^2 \leq (\frac{N-1}{N})^2 + \frac{N-1}{N} + (f(z_{N-1})s - \frac{N-1}{N})^2 + \frac{1}{N}\sum_{k=1}^{N-2}(2n + 2f(z_k) - \frac{2N-1}{N})$.

Simplifying and ignoring small terms, we get

$$\left(\frac{d\lambda_0}{ds}\right)^2 + \left(\frac{d\lambda_1}{ds}\right)^2 \leq (f(z_{N-1})-1)^2 + 2n + \frac{2}{N}\sum_{k=1}^{N-2} f(z_k). \qquad \square$$

The lemma implies that $|\frac{d\lambda_0}{ds}|, |\frac{d\lambda_1}{ds}| \leq \sqrt{(f(z_{N-1})-1)^2 + 2n + \frac{2}{N}\sum_{k=1}^{N-2} f(z_k)}$.

Using this, we will estimate $g(s) = \lambda_1(s) - \lambda_0(s)$. We conservatively approximate $\lambda_0(s)$ and $\lambda_1(s)$ by straight lines to get a lower bound on $g(s) = \lambda_1(s) - \lambda_0(s)$.

We divide the interval $[0,1]$ into three parts $[0,a]$, $[a,b]$ and $[b,1]$, corresponding to the parts when λ_0 approaches $\lambda = 1-s$, when both λ_0 and λ_1 are close to $\lambda = 1-s$, and when $\lambda_1(s)$ rises away from $\lambda = 1-s$ respectively.

Consider the first interval. Suppose $l_0^1(s)$ and $l_1^1(s)$ are lines such that $l_0^1(s) \geq \lambda_0(s)$ and $l_1^1(s) \leq \lambda_1(s)$. Then the gap between these lines is a lower bound on $\lambda_1(s) - \lambda_0(s)$ in the interval $(0,a)$.

[2] A similar bound can be obtained using a general result of Ambainis and Regev ([2], Lemma 4.1). Nevertheless, we presented a different way to demonstrate the possibility of problem specific approaches which can yield tighter bounds than the general one.

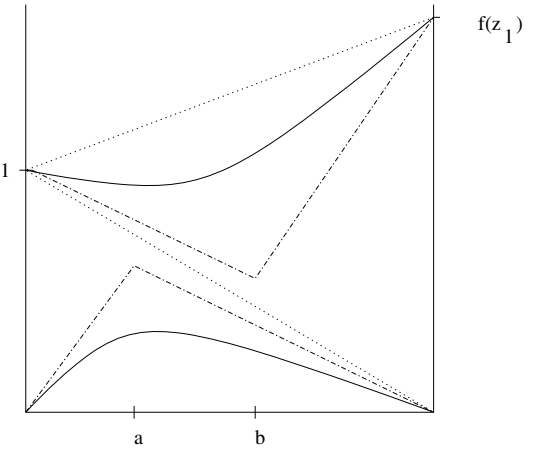


Fig. 2. Bold lines show actual eigenvalue curves and the dashed lines show lines that are assumed to bound the gap

Denote $\sqrt{(f(z_{N-1})-1)^2+2n+\frac{2}{N}\sum_{k=1}^{N-2}f(z_k)-1}$ by m. We take $l_0^1(s) = ms$ and $l_1^1(s) = 1-s$. Thus, the gap in the first interval is given by $g_1(s) \geq 1-s-ms$. Similarly, for the third interval, we choose $l_0^3(s) = 1-s$ and $l_1^3(s) = ms-m+f(z_1)$. Therefore, the gap $g_3(s)$ in this interval is greater than $ms-m+f(z_1)-(1-s)$. We will conservatively take the gap in the entire second interval to be g_{min}. See figure 2 for a rough sketch.

Let a be specified by a point on $l_0^1(s) = ms$ that lies vertically below the line $\lambda = 1-s$ by a distance g_{min}. Thus, $a = \frac{1-g_{min}}{m+1}$. Similarly, b is specified by a point on the line $l_1^3(s) = ms-m+f(z_1)$ that is above $\lambda = 1-s$ by g_{min}. Then, $b = \frac{m+1-f(z_1)+g_{min}}{m+1}$.

For the gap $g_2(s)$, recall that one of λ_0 and λ_1 is away from the line $1-s$ by a margin of g_{min}. Since this gives the deviation of only one of λ_0 and λ_1 from $1-s$, this is a conservative estimate of $g_2(s)$.

Therefore the total delay factor is given by

$$\begin{aligned}
T &\gg \int_0^a \frac{ds}{g_1(s)^2} + \int_a^b \frac{ds}{g_2(s)^2} + \int_b^1 \frac{ds}{g_3(s)^2} \\
&\leq \int_0^a \frac{ds}{\Big(1-s(1+m)\Big)^2} + \frac{1}{g_{min}^2}\int_a^b ds + \int_b^1 \frac{ds}{\Big((m+1)(s-1)+f(z_1)\Big)^2} \\
&= \frac{1}{m+1}\Big(\frac{1}{g_{min}}-1\Big) + \frac{m+2g_{min}-f(z_1)}{g_{min}^2(m+1)} + \frac{1}{m+1}\Big(\frac{1}{g_{min}}-\frac{1}{f(z_1)}\Big) \\
&\approx \frac{m-f(z_1)}{(m+1)g_{min}^2}.
\end{aligned}$$

Let us summarize:

Theorem 2. *Let the non-zero elements of the final Hamiltonian of the local adiabatic search algorithm be $f(z_1) \neq f(z_2) \neq f(z_{N-1})$, where each $f(z_i)$ is of size at most $O(\log N)$. Then, the time taken to evolve to the solution state of the final Hamiltonian is $O(D\frac{m-f(z_1)}{(m+1)g_{min}^2})$, where $m=\sqrt{(f(z_{N-1})-1)^2+2n+\frac{2}{N}\sum_{k=1}^{N-2} f(z_k)}-1$, $g_{min} = \min_{0\leq s\leq 1}(\lambda_1(s) - \lambda_0(s))$, and $D = ||\frac{d}{ds}H(s)||_2$.* □

To obtain tighter upper bounds on the running time, we would require the interval $b - a$ to be shorter: short enough to balance the denominator g_{min}^2 in the second term. This in turn requires tighter bounds on $\frac{d\lambda_0}{ds}$ and $\frac{d\lambda_1}{ds}$.

5 Conclusions

We introduced a technique for obtaining upper bounds on the running time of adiabatic quantum algorithms if the eigenvalue spectrum behaves in certain ways. We used this technique to investigate the robustness of the adiabatic quantum algorithm for unordered search when the final Hamiltonian is perturbed in the non-zero entries. Interesting open problems include tightening of the bounds, and application of our technique to other quantum adiabatic algorithms.

References

1. D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. In *Annual IEEE Symposium on Foundations of Computer Science*, pages 42–51, 2004.
2. A. Ambainis and O. Regev. An elementary proof of the quantum adiabatic theorem, 2004.
3. M. Andrecut and M. K. Ali. Adiabatic quantum oracles. *Journal of Physics A: Mathematical and General*, 37:L421–L427, 2004.
4. J. Åberg, D. Kult, and E. Sjöqvist. Quantum adiabatic search with decoherence in the instantaneous energy eigenbasis. *Physical Review A*, 72(042317), 2005.
5. J. Åberg, D. Kult, and E. Sjöqvist. Robustness of the adiabatic quantum search. *Physical Review A*, 71(060312(R)), 2005.
6. A. M. Childs, E. Farhi, J. Goldstone, and S. Gutmann. Finding cliques by quantum adiabatic evolution. *Quantum Information and Computation*, 2(3):181–191, 2002.
7. A. M. Childs, E. Farhi, and J. Preskill. Robustness of adiabatic quantum computation. *Physical Review A*, 65(012322), 2002.
8. E. Farhi, J. Goldstone, and S. Gutmann. A numerical study of the performance of a quantum adiabatic evolution algorithm for satisfiability. *quant-ph/0007071*, 2000.
9. E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*, 292(5516):472–476, 2001.
10. E. Farhi, J. Goldstone, S. Gutmann, and D. Nagaj. How to make the quantum adiabatic algorithm fail. quant-ph/0512159, 2005.
11. E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. *arXiv:quant-ph/0001106*, 2002.

12. L. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, 1997.
13. A. Messiah. *Quantum Mechanics*. John Wiley and Sons, New York, 1958.
14. M. V. Panduranga Rao. Solving a hidden subgroup problem using the adiabatic quantum computing paradigm. *Physical Review A*, 67:052306, 2003.
15. J. Roland and N. Cerf. Quantum search by local adiabatic evolution. *Physical Review A*, 65(042308), 2002.
16. G. Schaller, S. Mostame, and Ralf Schutzhold. General error estimate for adiabatic quantum computing. *Physical Review A*, 73:062307, 2006.
17. W. van Dam, M. Mosca, and U. V. Vazirani. How powerful is adiabatic quantum computation?. In *Annual IEEE Symposium on Foundations of Computer Science*, pages 279–287, 2001.
18. Z. Wei and M. Ying. Quantum adiabatic evolutions that can't be used to design efficient algorithms. arXiv:quant-ph/0604077, 2006.
19. M. Znidaric and M. Horvat. Exponential complexity of an adiabatic algorithm for an NP-complete problem. *Physical Review A*, 73:022329, 2006.

A Note on Universal Composable Zero Knowledge in Common Reference String Model*

Andrew C.C. Yao[1], Frances F. Yao[2], and Yunlei Zhao[3,**]

[1] Center of Advanced Study, Tsinghua University, Beijing, China
andrewcyao@tsinghua.eud.cn
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China
csfyao@cityu.edu.hk
[3] Software School, Fudan University, Shanghai 200433, China
ylzhao@fudan.edu.cn

Abstract. Pass observed that universal composable zero-knowledge (UCZK) protocols in the common reference string (CRS) model, where a common reference string is selected trustily by a trusted third party and is known to all players, lose deniability that is a natural property of any ZK protocol in the plain model [33]. An open problem (or, natural query) raised in the literature is: are there any other essential security properties, other than the well-known deniability property, that could be lost by universal composable zero-knowledge in the common reference string model, in comparison with UC security in the plain model? In this work, we answer this open question (or, natural query), by showing that UCZK protocols in the CRS model could lose concurrent general composability (CGC) and proof of knowledge (POK) properties that are very important and essential security implications of UCZK in the plain model. This is demonstrated by concrete attacks.

1 Introduction

Universal composability (UC) is a powerful notion proposed by Canetti [6] to describe cryptographic protocols that behave like ideal functionality, and can be composed *in arbitrary way*. The salient feature of UC secure protocol is that its security preserves even when it is composed with any arbitrary protocols (unpredictable environment) concurrently in asynchronous networks. In such settings, a protocol execution may run concurrently with an unknown number of other protocols. *These arbitrary protocols may be executed by the same parties or other parties, they may have potentially related inputs and the scheduling of message delivery may be adversarily coordinated. Furthermore, the local outputs of a protocol execution may be used by other protocols in an unpredictable way.*

* The work described in this paper was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Number CityU 122105) and CityU Research Grant (9380039) and 973 project of China (No. 2007CB807901).

** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 462–473, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In the framework of UC security, a generic definition is given for what it means for a protocol to "securely realize a given ideal functionality". Here, an "ideal functionality" is a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs. Informally, a protocol securely carries out a given ideal functionality if no adversary can gain more advantages from an attack on a real execution of the protocol, than from an attack on an ideal process where the parties merely hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it (without any other interaction). In other words, it is required that a real execution can be *emulated* in the ideal process.

Traditionally, emulation means that for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ attacking a real protocol execution, in which $\mathcal{A}$ controls the communication channels and potentially corrupts parties, there should exist an "ideal process adversary" or simulator $\mathcal{S}$ that causes the outputs of the parties in the ideal process to be essentially the same as the outputs of the parties in a real execution. In the UC framework, an additional *adversarial* entity $\mathcal{Z}$, called the environment, is introduced. As is hinted by its name, $\mathcal{Z}$ represents the external environment that consists of arbitrary protocol executions that may be running concurrently with the given protocol. This environment generates the inputs to all parties, read all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. Then, a protocol is said to UC realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties interacting with $\mathcal{F}$ in the ideal process. (In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$). *One salient and advantageous feature of UC security is the implication of concurrent general composability (CGC), i.e., composability concurrently with arbitrary protocols or unpredictable environment.*

Zero-knowledge (ZK) protocols allow a prover to validate theorems to a verifier without giving away any other knowledge other than the theorems being true (i.e., existing witnesses). This notion was introduced by Goldwasser, Micali and Rackoff [23] and its generality was demonstrated by Goldreich, Micali and Wigderson [22]. Since its introduction ZK has found numerous and extremely useful applications, and by now has been playing the central role in modern cryptography.

The concept of "proof of knowledge (POK)" was informally introduced in [23], and was formally treated in [4,19,5]. POK systems, especially zero-knowledge POK (ZKPOK) systems, play a fundamental role in the designing of cryptographic schemes and protocols, and enable a formal complexity theoretic treatment of what does it mean for a machine to "know" something. Very roughly, by "proof of knowledge" we mean that a possibly malicious prover can convince that an $\mathcal{NP}$ statement is true if and only if it, in fact, "knows" (i.e., possesses) a witness to the statement (rather than only convincing the language membership of the statement, i.e., the fact that a corresponding witness exists).

Clearly, achieving US secure protocols, in particular UCZK protocols, would be highly desirable in modern cryptography, especially for cryptographic protocols running over Internet. *We note that UCZK implies POK besides the above general CGC security implication in the plain model.* In general, it has been shown that any ideal functionality can be UC realized, as long as a majority of players are assumed to be honest [6]. But, for the more general case where a majority of players may be corrupted (in particular, for the important case of two-party protocols where each player wishes to maintain its security even if the other player is corrupted), it is shown that large classes of functionalities, in particular most two-party protocols, cannot be UC realized in the plain model where no trusted setup is assumed [6,9,10,29,31]. The impossibility results of [6,9,10] is further shown to be hold for *any* definition that implies security under the composition operation considered by the UC framework. Therefore, in the natural setting of no trusted setup and no honest majority (*including the important two-party case*), it is impossible to obtain security in a setting where protocols are run concurrently with arbitrary other protocols. Therefore, whenever this level of security is desired, some setup assumptions are necessary.

The typical setup assumption (in particular, considered in this work) is the common reference string (CRS) model (we note that our observations also apply to the public-key model). In the CRS model all parties are given a common, public reference string that is *ideally and trustily* chosen from a given distribution. A large number of round-efficient UC-secure protocols have been developed in the CRS model. In this work, we focus on UC security for (round-efficient) ZK protocols in the CRS model.

Pass observed that universal composable zero-knowledge protocols in the common reference string model lose deniability that is a natural property of any ZK protocol in the plain model [33]. An open problem (or, natural query) raised in the literature is: are there any other essential security properties, other than the well-known deniability property, that could be lost by universal composable zero-knowledge in the common reference string model, in comparison with UC security in the plain model? In this work, we answer this open question (or, natural query), by showing that UCZK protocols in the CRS model could lose concurrent general composability (CGC) and proof of knowledge (POK) properties that are very important and essential security implications of UCZK in the plain model. This is demonstrated by concrete attacks.

1.1 Related Works

Very recently, we noted the related *independent* work of [8]. The work of [8] clarifies the potential weakness of UC security in the common reference string model *in general*, with deniability loss as an illustrative example for ZK. In a sense, our work could also be viewed to exemplify, in another essential way (other than the well-known deniability loss), the general theme observed in the independent work of [8] on UC with global setup. More detailed discussions are presented in Section 4.

2 Preliminaries

We briefly recall preliminaries in this section. We assume the reader is familiar with some basic definitions: witness indistinguishability, argument/proof of knowledge, commitments, public-key encryption and signatures, etc. We also assume the reader is familiar with the UC framework (cf. [18,6,12,7]). Detailed presentation is deferred to the full version, due to space limitation.

Definition 1 (Σ-protocol [13]). *A 3-round public-coin protocol $\langle P, V\rangle$ is said to be a Σ-protocol for a relation R if the following hold:*

- *Completeness. If P, V follow the protocol, the verifier always accepts.*
- *Special soundness. From any common input x of length n and any pair of accepting conversations on input x, (a, e, z) and (a, e', z') where $e \neq e'$, one can efficiently compute w such that $(x, w) \in R$. Here a, e, z stand for the first, the second and the third message respectively and e is assumed to be a string of length k (that is polynomially related to n) selected uniformly at random in $\{0,1\}^k$.*
- *Special honest verifier zero-knowledge (SHVZK). There exists a PPT simulator S, which on input x (where there exists a w such that $(x, w) \in R$) and a random challenge string $\hat{e}$, outputs an accepting conversation of the form $(\hat{a}, \hat{e}, \hat{z})$, with the probability distribution that is indistinguishable from that of the real conversation (a, e, z) between the honest $P(w)$ and V on input x. A Σ-protocol is called* perfect/statistical *Σ-protocol, if it is perfect/statistical SHVZK.*

Σ-protocols are very useful cryptographic tools. A very large number of Σ-protocols have been developed in the literature. In particular, (the parallel repetition of) Blum's protocol for DHC [3] is a computational Σ-protocol for $\mathcal{NP}$, and most practical Σ-protocols for number-theoretical languages (e.g., DLP and RSA [34,25], etc) are of *perfect* SHVZK property. More details about Σ-protocols and their applications can be found in [16].

The OR-proof of Σ-protocols [14]. One basic construction with Σ-protocols allows a prover to show that given two inputs x_0, x_1, it knows a w such that either $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$, but without revealing which is the case (i.e., witness indistinguishable). Specifically, given two Σ-protocols $\langle P_b, V_b\rangle$ for R_b, $b \in \{0,1\}$, with random challenges of, without loss of generality, the same length k, consider the following protocol $\langle P, V\rangle$, which we call Σ_{OR}. The common input of $\langle P, V\rangle$ is (x_0, x_1) and P has a private input w such that $(x_b, w) \in R_b$.

- P computes the first message a_b in $\langle P_b, V_b\rangle$, using x_b, w as private inputs. P chooses e_{1-b} at random, runs the SHVZK simulator of $\langle P_{1-b}, V_{1-b}\rangle$ on input (x_{1-b}, e_{1-b}), and lets $(a_{1-b}, e_{1-b}, z_{1-b})$ be the output. P finally sends a_0, a_1 to V.
- V chooses a random k-bit string e and sends it to P.

- P sets $e_b = e \oplus e_{1-b}$ and computes the answer z_b to challenge e_b using (x_b, a_b, e_b, w) as input. He sends $((e_0, z_0), (e_1, z_1))$ to V.
- V checks that $e = e_0 \oplus e_1$ and that conversations (a_0, e_0, z_o), (a_1, e_1, z_1) are accepting conversations with respect to inputs x_0, x_1, respectively.

Theorem 1. [14] *The protocol Σ_{OR} above is a Σ-protocol for R_{OR}, where $R_{OR} = \{((x_0, x_1), w) | (x_0, w) \in R_0$ or $(x_1, w) \in R_1\}$. Moreover, Σ_{OR}-protocols are witness indistinguishable (WI) proof of knowledge systems.*

Ω-protocols [18]. An Ω-protocol is a Σ-protocol in the common reference string (CRS) model, with a special straight-line simulation/extraction property. Specifically, an Ω-protocol $\langle P, V \rangle_{[\sigma]}$ for an $\mathcal{NP}$-relation R and common reference string σ, is a Σ-protocol for relation R with the following additional properties:

- For a given distribution ensemble $\mathcal{D}$, on security parameter 1^n a common reference string σ is drawn from $\mathcal{D}_n$. The players take σ as an additional input (to generate messages from them). Naturally, the simulator S in the definition of Σ-protocol may also take σ as an additional input.
- There exists a polynomial-time extractor $E = (E_1, E_2)$ such that the first element of the output of $E_1(1^n)$ is statistically indistinguishable from $\mathcal{D}_n$. Furthermore, given $(\sigma, \tau) \leftarrow E_1(1^n)$, if there exist two accepting conversations (a, e, z) and (a, e', z') with $e \neq e'$ on common input x and CRS σ, then $E_2(x, \tau, (a, e, z))$ outputs w such that $(x, w) \in R$.

Notice that the above second property is similar to the special soundness of Σ-protocols. For a Σ-protocol, there could exist an accepting conversation even for an invalid proof, but two accepting conversations (with the same first-round message but different second-round challenges) guarantee that the proof is valid. Here, for a Ω-protocol, the extractor E can always extract something from any conversation, but it might not be the witness if there is only one accepting conversation. However, having two different accepting conversations guarantees the extracted value is indeed a witness.

A natural way to construct Ω-protocols is as follows: the common reference string will consist of a random public-key pk for a semantically-secure encryption scheme. Then for a given $(x, w) \in R$, we will construct an encryption c of w under public-key pk, and then construct a Σ-protocol to prove that the value encrypted in c is indeed a witness w such that $(x, w) \in R$.

As with Σ-protocol, we can construct the OR-proof combining a Ω-protocol and a Σ-protocol.

The zero-knowledge functionality [6,12]. The ZK functionality $\mathcal{F}_{\text{ZK}}^{R}$, parameterized by a relation R, is presented in Figure 1 (page 467). In the functionality, the prover sends to the functionality the input x together with a witness w. If $R(x, w)$ holds, then the functionality forwards x to the verifier. As pointed in [6], this is actually a *proof of knowledge* in that the verifier is assured that the prover actually knows w.

Functionality $\mathcal{F}_{\mathbf{ZK}}^{R}$

$\mathcal{F}_{\mathrm{ZK}}^{R}$ proceeds as follows, running with security parameter 1^n, a prover P_i, a verifier P_j, and an adversary $\mathcal{S}$:

- Upon receiving $(\mathsf{zk\text{-}prover}, sid, P_i, P_j, x, w)$ from P_i: If $R(x, w)$ then send $(\mathsf{ZK\text{-}PROOF}, sid, P_i, P_j, x)$ to P_j and $\mathcal{S}$ and halts. Otherwise, ignore.

Fig. 1. The zero-knowledge functionality (for relation R)

Functionality $\hat{\mathcal{F}}_{\mathbf{ZK}}^{R}$

$\hat{\mathcal{F}}_{\mathrm{ZK}}^{R}$ proceeds as follows, running with security parameter 1^n, parties $P_1, \cdots, P_n$, and an adversary $\mathcal{S}$:

- Upon receiving $(\mathsf{zk\text{-}prover}, sid, ssid, P_i, P_j, x, w)$ from P_i: If $R(x, w)$ then send $(\mathsf{ZK\text{-}PROOF}, sid, ssid, P_i, P_j, x)$ to P_j and $\mathcal{S}$ and halts. Otherwise, ignore.

Fig. 2. The multi-session ZK functionality (for relation R)

One shortcoming of the above formulation is that we will be designing and analyzing protocols in the common reference string model, and so they will be operating in the $\mathcal{F}_{\mathrm{CRS}}^{\mathcal{D}}$-hybrid model, where $\mathcal{F}_{\mathrm{CRS}}^{\mathcal{D}}$ is the CRS generation functionality that, for a given security parameter 1^n, chooses a string from distribution $\mathcal{D}_n$ and hands it to all parties and the adversary (*but not directly to the environment*). However, directly realizing $\mathcal{F}_{\mathrm{ZK}}^{R}$ in the $\mathcal{F}_{\mathrm{CRS}}^{\mathcal{D}}$-hybrid model and using the universal composition theorem would result in a composed protocol where a new instance of the reference string is needed for each proof. This is extremely inefficient and does not reflect the notion of the CRS model, where an unbounded number of protocol instances would use the same copy of the string. Canetti and Rabin [12] suggested the following notion to cope with this problem:

- Universal composition with joint state: Let $\mathcal{F}$ and $\mathcal{G}$ be ideal functionalities, and let $\hat{\mathcal{F}}$ denote the "multi-session extension of $\mathcal{F}$", in which $\hat{\mathcal{F}}$ will run multiple copies of $\mathcal{F}$, where each copy is identified by a special *sub-session identifier* (*ssid*). Now, let π be a protocol in the $\mathcal{F}$-hybrid model, and let $\hat{\rho}$ be a protocol that securely realizes $\hat{\mathcal{F}}$ in the $\mathcal{G}$-hybrid model. Then, construct the composed protocol $\pi^{[\hat{\rho}]}$ by replacing all the copies of $\mathcal{F}$ in π by a single copy of $\hat{\rho}$. The universal composition with joint state theorem of [12] states that $\pi^{[\hat{\rho}]}$, running in the $\mathcal{G}$-hybrid model, correctly emulates π in the $\mathcal{F}$-hybrid model.

The definition $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R}$, the multi-session extension of $\mathcal{F}_{\mathrm{ZK}}^{R}$, is presented in Figure 2 (page 467). Note that there are two types of indices: the *sid* differentiates messages to $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R}$ from messages sent to other functionalities; and the sub-session ID *ssid* is unique per input message (or proof).

3 Concurrent General Composition Attack on UCZK in the Common Reference String Model

In this section, we present a concurrent general composition attack to the protocol of [18] that is UCZK in the common reference string model.

3.1 The Protocol Structure of UCZK of [18]

We first recall the protocol structure of the UCZK protocol of [18].

Common reference string: $(verk, \sigma')$, where $verk$ is a random verification key of a signature scheme secure against chosen message attacks, σ' is the public reference string for the underlying Ω-protocol (typically, σ' is a random public-key of semantically-secure PKE).
Common input: $x \in L$, where L is an $\mathcal{NP}$-language with $\mathcal{NP}$-relation R_L.
Auxiliary date: An auxiliary data aux that may contain any arbitrary public values.
Prover's private input: w s.t. $(x, w) \in R_L$.
Main-proof stage: consists of three phases (in real implementation, phases are combined):

- **Phase-1:** Prover P generates a key-pair (vk, sk) for a one-time strong signature, sends vk to the verifier V.
- **Phase-2:** Give a OR-proof: one Ω-proof for showing the knowledge of w (typically, send a encryption c of w using σ', and prove by Σ-protocol that the encrypted value is indeed a witness for $x \in L$); one Σ-protocol for showing the knowledge of a signature on vk under $verk$. We denote by $a = (a_L, a_{vk})$, e, and $z = ((e_L, z_L), (e_{vk}, z_{vk}))$ the first-round, second-round and the third-round message of the OR-proof respectively, where (a_L, e_L, z_L) constitute the (partial) conversation of the Ω-protocol (specifically, the conversation of the Σ-protocol in the Ω-protocol for showing the knowledge of the value encrypted in c is indeed a valid witness for $x \in L$), and (a_{vk}, e_{vk}, z_{vk}) constitutes the conversation of the Σ-protocol for showing the knowledge of the signature on vk under $verk$, and $e = e_L \oplus e_{vk}$.
- **Phase-3:** Applying sk on the whole transcript to get a one-time strong signature s, send s to V.

Notes: The above protocol is shown to be UCZK in the common reference string model, assuming static corruptions [18]. For UCZK with adaptive corruptions, the above protocol is augmented as follows: In Phase-2, the prover does not send $a = (a_L, a_{vk})$ directly. Rather, it first commits to a and the auxiliary information aux by using a special trapdoor commitment scheme, called simulation-sound trapdoor commitments (SSTC), following the paradigm of [15]. Then, in the third-round of the OR-proof of Phase-2, the prover decommits accordingly and reveals a. The following CGC attack is described against the above UCZK with static corruption, but it can be trivially extended to work on the augmented adaptive-corruption version as well.

3.2 The CGC Attack

To present a CGC attack, we need to first design a (different) protocol, and then show that when composed with the designed protocol the UCZK protocol of [18] is not secure. We present a natural and also very useful protocol, and show that when composed with this natural and practical protocol, a malicious adversary can convince the honest verifier of any statement in the original UCZK protocol of [18] *but without knowing any witness for the statement being proved*. This shows that UCZK protocol in the common reference string model could lose concurrent general composability property and also the POK property. We suggest that such security losses might be more harmful, in comparison with the loss of deniability observed in [33].

The encrypt/commit-then-proof protocol. The protocol to be composed with the UCZK of [18] is the natural and very useful *encrypt/commit-then-proof* protocol $\langle P', V' \rangle$, described as follows.

Common input: $x \in L$.
Prover's private input: w s.t. $(x, w) \in R_L$.
Main-proof stage: consists of two phases:

Phase-1: The verifier V' generates and sends to the prover P' a random public-key σ' for a semantically-secure PKE scheme. Here, σ' can also be viewed as the first-round message of a commitment scheme.

Phase-2: The prover P' encrypts (i.e., commits) w to c using the public-key σ'. Then, P' proves to V' that the value committed is indeed a witness for $x \in L$, by executing a Σ-protocol with V'. We denote by a_L, e_L, z_L the first-round, second-round and third-round message of the Σ-protocol.

We remark that the above encrypt/commit-then-proof protocol is a natural and very useful protocol in practice. The encrypt/commit-then-proof paradigm has been employed in a number of works for various cryptographic tasks and settings (e.g., [28,30,11,1], etc). When the protocol works in the public-key model with σ' as the verifier's public-key, such protocol is also a common paradigm for achieving *plaintext-aware* (interactive and verifiable) encryption (e.g., [27]), which is also used in group signature and group encryption systems.

Message schedule of the CGC attack. We now describe the message schedule of the CGC attack, that enables an adversary to convinces any statement in the original UCZK protocol of [18] but without knowing any corresponding $\mathcal{NP}$-witness.

The adversary $\mathcal{A}$ runs the UCZK protocol of [18] and the above commit-then-proof protocol concurrently, by playing the role of prover in the UCZK protocol of [18] and playing the role of verifier in the commit-then-proof protocol. In other words, the adversary $\mathcal{A}$ corrupts and controls the prover P of UCZK of [18] and the verifier V' of the commit-then-proof protocol at the onset of the computation. $\mathcal{A}$ schedules the messages as follows.

1. $\mathcal{A}$ first executes the UCZK with V on common input x and the common reference string $(verk, \sigma')$. For presentation simplicity, we call such execution *the first session*. Specifically, it generates a key-pair (vk, sk) for a one-time strong signature, sends vk to the verifier V, just as the honest prover does. When it moves into Phase-2 of the UCZK, $\mathcal{A}$ suspends the first session.
2. $\mathcal{A}$ executes the commit-then-prove protocol with P' on common input x (x could be set by $\mathcal{A}$ via the environment). For presentation simplicity, we call the execution of the commit-then-prove protocol *the second session*. Specifically, $\mathcal{A}$ sends σ' (*got from the CRS of the first session*) to P' as the Phase-1 message of the second session. After receiving from P' the first-round message of Phase-2 of the second session, $\mathcal{A}$ suspends the second session. Note that the first-round message of Phase-2 of the second session from P' consists of c (that encrypts w) and the first-round message a_L of the underlying Σ-protocol executed in Phase-2 of the second session.
3. Now, $\mathcal{A}$ continues the first session, and works as follows. On $(vk, verk)$, it generates a simulated conversation (a_{vk}, e_{vk}, z_{vk}) for the Σ-protocol of Phase-2 of the first session (that is used to prove the knowledge of a signature of vk under $verk$), by running the underlying SHVZK simulator. Then, $\mathcal{A}$ sends (c, a_L, a_{vk}) to V as the first-round message of Phase-2 of the first session. After receiving from V the random challenge e (i.e., the second-round message of the OR-proof of Phase-2 of the first session), $\mathcal{A}$ sets $e_L = e \oplus e_{vk}$ and suspends the first session again. *Note that* (c, a_L) *are got from the second session.*
4. $\mathcal{A}$ continues the second session again, sends $e_L = e \oplus e_{vk}$ to P' as the second-round message of Phase-2 of the second session. After receiving from P' the last-round message e_L of the second session, $\mathcal{A}$ stops the second session.
5. $\mathcal{A}$ continues the first-session again, sends $z = ((e_L, z_L), (e_{vk}, z_{vk}))$ to V as the last-round message of the OR-proof of Phase-2 of the first session.
6. Finally, $\mathcal{A}$ applies the one-time strong signing key sk on the whole transcript of the first session to get a valid signature s, and sends s to V. Note that $\mathcal{A}$ can do this, as the one-time strong key pair (vk, sk) are generated by itself.

Note that (c, a_L, e_L, z_L) is an accepting conversation of the Ω-protocol for showing $x \in L$, $(a_{vk}, e_{vk}, z_{vk}$ is an accepting conversation for showing the knowledge of the signature of vk under $verk$, and also $e = e_L \oplus e_{vk}$. Furthermore, the one-time strong signature s is also valid. This means that, from the viewpoint of V, $\mathcal{A}$ has successfully convinced V of the statement "$x \in L$" in the first session with the UCZK protocol, *but* $\mathcal{A}$ *actually does not know any corresponding* $\mathcal{NP}$*-witness*! It is also easy to see that the above CGC attack schedule can be trivially extended to the augmented adaptive corruption version of the UCZK of [18].

Notes: The adversary $\mathcal{A}$ does not use the same CRS in the second session, but a part of the CRS. Also, the commit-then-proof protocol is run in the plain model. We remark that $\mathcal{A}$ can potentially use a completely different (but maliciously related to CRS) message in Phase-1 of the second session. *In general,* $\mathcal{A}$ *can*

potentially malleate the CRS of one session into some message of another concurrent session that is completely different with the CRS but maliciously related. We also note that it is impossible to prevent transparent adversaries. Specifically, an adversary runs the same protocol twice in two session (in one session, the *same* CRS could be sent by a player) and forwards the messages from one session to another session (i.e., the transcripts in the two sessions are identical). Such transparent adversary is impossible to prevent, and is not viewed as a harmful adversarial activity *by definition*, analogue to the definition of non-malleability [17].

4 Comments

We comment that the above CGC attack contradicts our intuition for the security guaranteed by universal composable security. We remark that it is true that UC security in the plain model does guarantee the composability security with arbitrary protocols (environment), and the POK property for ZK in particular. The implicit reason, as discussed in the independent work of [8], is that in the traditional UC formulation in the common reference string model the common reference string is *not* given to the environment. In general, in the UC formulation of [6], the environment is not allowed to directly invoke (interact) with subroutines. That is, the environment is not allowed to interact directly with the ideal functionality $\mathcal{F}$ (in particular, the CRS generation functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ in our case) in the $\mathcal{F}$-hybrid model [6,12]. This fact allows the CRS simulation by the ideal-process adversary in security analysis. The work of [8] pointed out the potential security losses for UC in the common reference string model in general, with deniability loss as an illustrative example for UCZK in the common reference string model. Our attack shows that UCZK in the common reference string model could lose more essential and important security guarantees, i.e., concurrent general composability and POK that are very essential security implications of UCZK in the plain model. In this sense our work could also be viewed to exemplify, in another essential way (other than the well-known deniability loss [33]), the general theme observed in [8] on UC with global setup.

Acknowledgements. We thank Ran Canetti for kind clarifications about UC in the CRS model, and referring us to [8].

References

1. B. Barak, M. Prabhakaran, and A. Sahai. Concurrent Non-Malleable Zero-Knowledge. Cryptology ePrint Archive, Report No. 2006/355. Extended abstract appears in FOCS 2006.
2. M. Blum. Coin Flipping by Telephone. In *proc. IEEE Spring COMPCOM*, pages 133-137, 1982.
3. M. Blum. How to Prove a Theorem so No One Else can Claim It. In Proceedings of the International Congress of Mathematicians, Berkeley, California, USA, 1986, pp. 1444-1451.

4. M. Bellare and O. Goldreich. On Defining Proofs of Knowledge In *E. F. Brickell (Ed.): Advances in Cryptology-Proceedings of CRYPTO 1992, LNCS 740*, pages 390-420, Springer-Verlag, 1992.
5. M. Bellare and O. Goldreich. On Probabilistic versus Deterministic Provers in the Definition of Proofs Of Knowledge. Electronic Colloquium on Computational Complexity, 13(136), 2006. Available also from Cryptology ePrint Archive, Report No. 2006/359.
6. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136-145, 2001.
7. R. Canetti. Security and Composition of Cryptographic Protocols: A Tutorial. *Distributed Computing column of SIGACT News*, Vol. 37, Nos. 3 and 4, 2006. Available also from Cryptology ePrint Archive, Report 2006/465.
8. R. Canetti, Y. Dodis, R. Pass and S. Walfish. Universal Composable Security with Global Setup. TCC 2007, to appear. Available from: Cryptology ePrint Archive, Report No. 2006/432.
9. R. Canetti and M. Fischlin. Universal Composable Commitments. In *Advances in Cryptology-Proceedings of CRYPTO 2001, LNCS 2139*, pages 19-40. Springer-Verlag, 2001.
10. R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universal Composition Without Set-Up Assumptions. In *E. Biham (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 2003, LNCS 2656*, pages 68-86. Springer-Verlag, 2003.
11. R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *ACM Symposium on Theory of Computing*, pages 494-503, 2002.
12. R. Canetti and T. Rabin. Universal Composition with Joint State. In *Advances in Cryptology-Proceedings of CRYPTO 2002, LNCS 2729*, pages 265-281, Springer-Verlag, 2003.
13. R. Cramer. Modular Design of Secure, yet Practical Cryptographic Protocols, PhD Thesis, University of Amsterdam, 1996.
14. R. Cramer, I. Damgard and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Y. Desmedt (Ed.): Advances in Cryptology-Proceedings of CRYPTO 1994, LNCS 893*, pages 174-187. Springer-Verlag, 1994.
15. I. Damgard. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In *B. Preneel (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 2000, LNCS 1807*, pages 418-430. Springer-Verlag, 2000.
16. I. Damgard. Lecture Notes on Cryptographic Protocol Theory, BRICS, Aarhus University, 2003.
17. D. Dolev, C. Dwork and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, 30(2): 391-437, 2000. Preliminary version in *ACM Symposium on Theory of Computing*, pages 542-552, 1991.
18. J. A. Garay, P. MacKenzie and K. Yang. Strengthening Zero-Knowledge Protocols Using Signatures. *Journal of Cryptology*, to appear. Preliminary version appears in *E. Biham (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 2003, LNCS 2656* , pages 177-194. Springer-Verlag, 2003.
19. O. Goldreich. *Foundation of Cryptography-Basic Tools*. Cambridge University Press, 2001.

20. O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design. In *IEEE Symposium on Foundations of Computer Science*, pages 174-187, 1986.
21. O. Goldreich, S. Micali and A. Wigderson. How to Prove all $\mathcal{NP}$-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design. In *A. M. Odlyzko (Ed.): Advances in Cryptology-Proceedings of CRYPTO 1986, LNCS 263*, pages 104-110, Springer-Verlag, 1986.
22. O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing But Their Validity or All language in $\mathcal{NP}$ Have Zero-Knowledge Proof Systems. *Journal of the Association for Computing Machinery*, 38(1): 691-729, 1991. Preliminary version appears in [20,21].
23. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems In *ACM Symposium on Theory of Computing*, pages 291-304, 1985.
24. S. Goldwasser, S. Micali and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks. *SIAM Journal on Computing*, 17(2): 281-308, 1988.
25. L. Guillou and J. J. Quisquater. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing both Transmission and Memory. In *C. G. Gnther (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 1988, LNCS 330* , pages 123-128, Springer-Verlag, 1988.
26. J. Hastad, R. Impagliazzo, L. A. Levin and M. Luby. Construction of a Pseudorandom Generator from Any One-Way Function *SIAM Journal on Computing*, 28(4): 1364-1396, 1999.
27. J. Katz. Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications. In *E. Biham (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 2003, LNCS 2656* , pages 211-228. Springer-Verlag, 2003.
28. J. Kilian. Uses of Randomness in Algorithms and Protocols. MIT Press, Cambridge, MA, 1990.
29. Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *IEEE Symposium on Foundations of Computer Science*, pages 394-403, 2003.
30. Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology*, 16(3): 143-184, 2003. Preliminary version appears in Crypto 2001, LNCS 2139, pages 171-189, Springer-Verlag, 2001.
31. Y. Lindell. Lower Bounds for Concurrent Self Composition. In *M. Naor (Ed.): Theory of Cryptography (TCC) 2004, LNCS 2951*, pages 203-222, Springer-Verlag, 2004.
32. M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology*, 4(2): 151-158, 1991.
33. R. Pass. On Deniabililty in the Common Reference String and Random Oracle Models. In*D. Boneh (Ed.): Advances in Cryptology-Proceedings of CRYPTO 2003, LNCS 2729*, pages 316-337, Springer-Verlag 2003.
34. C. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3): 24, 1991.

A Note on the Feasibility of Generalized Universal Composability* (Extended Abstract)

Andrew C.C. Yao[1], Frances F. Yao[2], and Yunlei Zhao[3,**]

[1] Center of Advanced Study, Tsinghua University, Beijing, China
andrewcyao@tsinghua.eud.cn
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China
csfyao@cityu.edu.hk
[3] Software School, Fudan University, Shanghai 200433, China
ylzhao@fudan.edu.cn

Abstract. We clarify the potential limitation of the general feasibility for generalized universal composability (GUC) proposed in the recent work [8], and discuss a general principle for fully realizing universal composability. This in particular demonstrates the hardness of achieving generalized universal composability, and prevents potential misinterpretation in applications. We also propose some fixing approaches, which involve a source/session-authentic ID-based trapdoor commitment scheme via the hash-then-commit paradigm that could possibly be of independent interest.

1 Introduction

The intuitive security goal aimed by the universal composability (UC) notion of [6] is to provide cryptographic systems a robust composability with *arbitrary* protocols (*unpredictable* environment). That is, a UC-secure protocol (henceforth referred to as the "challenge protocol") should preserve its security even when it is composed concurrently with any arbitrary protocols in any arbitrary malicious (unpredictable) way in asynchronous networks like the Internet. These arbitrary protocols may be executed by the same parties or other parties, they may have potentially related inputs and the scheduling of message delivery may be adversarially coordinated. Furthermore, the local outputs of a protocol execution may be used by other protocols in an unpredictable way. Clearly, achieving UC-secure protocols is highly desirable in modern cryptography.

But, the recent works of [8,23] show that traditional UC formulations in [6,12] do *not* capture the intuitive security goal of concurrent general composabililty

* The work described in this paper was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Number CityU 122105) and CityU Research Grant (9380039) and 973 project of China (No. 2007CB807901).

** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 474–485, 2007.
© Springer-Verlag Berlin Heidelberg 2007

described above, i.e., composability concurrently with *arbitrary* protocols or unpredictable environment. Roughly speaking, traditional UC-secure protocols only guarantee composability with "independent (stateless)" other different protocols *that do not share state information with it* [8].[1] This requirement is unrealistic to reflect adversarial activities in asynchronous networks like the Internet (that is the original motivation for introducing UC), and limits the applicability of UC theory. For example, most developed UC-secure protocols are with trusted global setups, e.g., a common reference string (CRS) or public-keys (PKs) that are drawn randomly and trustily from some predefined distributions and are known to all parties, thus protocols trivially share common state information[2]. The necessity of trusted setups for UC security is from the fact that large classes of functionalities cannot be UC realized in the plain model without assuming a majority of honest players, in particular, for the important case of two-party protocols, most of them cannot be UC realized [6,9,10,19,20].

To redeem the situation, the work of [8] reformulates UC security by explicitly allowing the external arbitrary protocols can share any maliciously-related state information with the challenge protocol. Such augmented UC notion is named *generalized UC* (GUC) [8] (in the rest of this work, we also write *generalized* UC just as UC for presentation simplicity as it captures the original intuition and motivation of UC, distinguished from *traditional* UC). The work of [8] also proposes an approach for reestablishing UC feasibility in the more complex GUC setting with some trusted setups (specifically, the key registration with knowledge (KRK) model and the augmented common reference string (ACRS) model). In this work, we consider the general feasibility of GUC in the ACRS model. The key difference between ACRS model and the traditional CRS model is that: the ACRS model additionally allows *corrupted* parties to ask the trusted CRS generator to obtain "personalized" secret-keys that are derived from the common reference string, their public identities, and some "global secret" that is related to the common reference string and remains secret to all parties.

The general UC feasibility proposed in [8] relies on the following assumption: the arbitrary different protocols (environment) can share state information with the challenge protocol only via a trusted functionality, e.g., the CRS generation functionality. That is, any state information that will be (potentially maliciously) shared by the challenge protocol with the external environment, no matter what adversarial strategies can be employed by the adversary, can be derived from queries to the trusted functionality, e.g., from the common reference string (in a way, the adversary is essentially restricted to only malleate the common reference string with external environment). We note that such limitation on state information sharing is still unrealistic to reflect adversarial activities happening in asynchronous networks like the Internet. Typically, any message received

[1] We note that the intuitive goal of composability with arbitrary protocols (or, unpredictable environment) was informally claimed in most existing works, which could potentially cause misinterpretation partially due to the high system complexity.

[2] As noted in [8], the approach proposed in [12] for handling universal composition with joint state (JUC) also does not fully work in this case.

by an honest player in the challenge protocol could be maliciously dependent on not only the transcript of the challenge protocol, but also, more harmfully, the whole transcript of the external arbitrary protocols running concurrently with the challenge protocol. That is, it is a natural adversarial strategy of the adversary that it manages to make the challenge protocol maliciously related to the external environment by concurrent interleaving and malleating attacks, thus sharing some maliciously-related state-information amongst them. The key point here is, the external environment (i.e., the arbitrary protocols running concurrently with the challenge protocol) and adversarial strategies employed by the adversary are *unpredictable*. In a sense, the general UC feasibility proposed in [8] is w.r.t. "*predictable*" environment and adversarial strategies. Though the general UC feasibility proposed in [8] provides much better composability guarantee than traditional UC, where for traditional UC the environment is required to be "independent" (stateless) with the challenge protocol, it seems to be still not satisfactory enough and contradicts the intuition and motivation for universal composability.

To illustrate the potential weakness of assuming "predictable" environment and adversarial strategies, we demonstrate a concrete (simple) attack on the UCZK protocol implied by the general UC feasibility of [8], showing that it could lose concurrent general composability and proof of knowledge properties. This is helpful for a precise understanding of the general UC feasibility of [8], and prevents potential misinterpretation in applications. We then discuss a generic principle for fully achieving universal composability, which in turn demonstrates the hardness of achieving generalized universal composability (even with trusted setups and/or a majority of honest players). Finally, we propose some fixing approaches, which involve a source/session-authentic ID-based trapdoor commitment scheme via the hash-then-commit paradigm that could possibly be of independent interest.

2 Preliminaries

We briefly recall preliminaries in this section. To save space, we do not represent the UC framework in this work. The reader is referred to [6,12,7,8] (or, to the related work [23]) for details.

We use standard notations and conventions below for writing probabilistic algorithms, experiments and interactive protocols. If A is a probabilistic algorithm, then $A(x_1, x_2, \cdots; r)$ is the result of running A on inputs $x_1, x_2, \cdots$ and coins r. We let $y \leftarrow A(x_1, x_2, \cdots)$ denote the experiment of picking r at random and letting y be $A(x_1, x_2, \cdots; r)$. If S is a finite set then $x \leftarrow S$ is the operation of picking an element uniformly from S. If α is neither an algorithm nor a set then $x \leftarrow \alpha$ is a simple assignment statement. By $[R_1; \cdots; R_n : v]$ we denote the set of values of v that a random variable can assume, due to the distribution determined by the sequence of random processes $R_1, R_2, \cdots, R_n$. By $\Pr[R_1; \cdots; R_n : E]$ we denote the probability of event E, after the ordered execution of random processes $R_1, \cdots, R_n$.

Let $\langle P, V\rangle$ be a probabilistic interactive protocol, then the notation $(y_1, y_2) \leftarrow \langle P(x_1), V(x_2)\rangle(x)$ denotes the random process of running interactive protocol $\langle P, V\rangle$ on common input x, where P has private input x_1, V has private input x_2, y_1 is P's output and y_2 is V's output. We assume w.l.o.g. that the output of both parties P and V at the end of an execution of the protocol $\langle P, V\rangle$ contains a transcript of the communication exchanged between P and V during such execution.

Definition 1 ((public-coin) interactive argument/proof system). *A pair of interactive machines, $\langle P, V\rangle$, is called an interactive argument system for a language L if both are probabilistic polynomial-time (PPT) machines and the following conditions hold:*

- *Completeness. For every $x \in L$, there exists a string w such that for every string z,*
  $\Pr[\langle P(w), V(z)\rangle(x) = 1] = 1$.
- *Soundness. For every polynomial-time interactive machine P^*, and for all sufficiently large n's and every $x \notin L$ of length n and every w and z,* $\Pr[\langle P^*(w), V(z)\rangle(x) = 1]$ *is negligible in n.*

An interactive protocol is called a proof *for L, if the soundness condition holds against any (even power-unbounded) P^* (rather than only PPT P^*). An interactive system is called a public-coin system if at each round the prescribed verifier can only toss coins and send their outcome to the prover.*

Definition 2 (statistically/perfectly binding bit commitment scheme). *A pair of PPT interactive machines, $\langle P, V\rangle$, is called a perfectly binding bit commitment scheme, if it satisfies the following:*

Completeness. *For any security parameter n, and any bit $b \in \{0,1\}$, it holds that*
$\Pr[(\alpha, \beta) \leftarrow \langle P(b), V\rangle(1^n); (t, (t, v)) \leftarrow \langle P(\alpha), V(\beta)\rangle(1^n) : v = b] = 1$.

Computationally hiding. *For all sufficiently large n's, any PPT adversary V^*, the following two probability distributions are computationally indistinguishable:* $[(\alpha, \beta) \leftarrow \langle P(0), V^*\rangle(1^n) : \beta]$ *and* $[(\alpha', \beta') \leftarrow \langle P(1), V^*\rangle(1^n) : \beta']$.

Perfectly Binding. *For all sufficiently large n's, and* any *adversary P^*, the following probability is negligible (or equals 0 for perfectly-binding commitments):*
$\Pr[(\alpha, \beta) \leftarrow \langle P^*, V\rangle(1^n); (t, (t, v)) \leftarrow \langle P^*(\alpha), V(\beta)\rangle(1^n); (t', (t', v')) \leftarrow \langle P^*(\alpha), V(\beta)\rangle(1^n) : v, v' \in \{0,1\} \bigwedge v \neq v']$.

*That is, no (*even computational power unbounded*) adversary P^* can decommit the same transcript of the commitment stage both to 0 and 1.*

One-round perfectly-binding (computationally-hiding) commitments can be based on any one-way permutation OWP [2,17]. Loosely speaking, given a OWP f with a hard-core predict b (cf. [14]), on a security parameter n one commits a bit σ by uniformly selecting $x \in \{0,1\}^n$ and sending $(f(x), b(x) \oplus \sigma)$ as a

commitment, while keeping x as the decommitment information. Statistically-binding commitments can also be based on any one-way function (OWF) but run in two rounds [21,18].

Definition 3 (system for argument/proof of knowledge [4,14,5]). *Let R be a binary relation and $\kappa : N \to [0,1]$. We say that a probabilistic polynomial-time (PPT) interactive machine V is a knowledge verifier for the relation R with knowledge error κ if the following two conditions hold:*

- *Non-triviality: There exists an interactive machine P such that for every $(x,w) \in R$ all possible interactions of V with P on common input x and auxiliary input w are accepting.*
- *Validity (with error κ): There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine K such that for every interactive machine P^*, every $x \in L_R$, and every $w, r \in \{0,1\}^*$, machine K satisfies the following condition:*
  Denote by $p(x,w,r)$ the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by $P^*_{x,w,r}$ (where $P^*_{x,w,r}$ denotes the strategy of P^* on common input x, auxiliary input w and random-tape r). If $p(x,w,r) > \kappa(|x|)$, then, on input x and with oracle access to $P^*_{x,w,r}$, machine K outputs a solution $w' \in R(x)$ within an expected number of steps bounded by
  $$\frac{q(|x|)}{p(x,w,r) - \kappa(|x|)}$$
  The oracle machine K is called a knowledge extractor.

An interactive argument/proof system $\langle P, V \rangle$ such that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a system for argument/proof of knowledge (AOK/POK) for the relation R.

We mention that Blum's protocol for directed Hamiltonian Cycle DHC [3] is just a 3-round public-coin (witness-indistinguishable) POK system for $\mathcal{NP}$, which is recalled below.

Blum's protocol for DHC [3]. The n-parallel repetitions of Blum's basic protocol for proving the knowledge of Hamiltonian cycle on a given directed graph G [3] is just a 3-round public-coin (witness-indistinguishable) POK system for $\mathcal{NP}$ (with knowledge error 2^{-n}) under any one-way permutation (as the first round of it involves one-round perfectly-binding commitments of a random permutation of G). The following is the description of Blum's *basic* protocol for DHC:

Common input. A directed graph $G = (V, E)$ with $q = |V|$ nodes.
Prover's private input. A directed Hamiltonian cycle C_G in G.
Round-1. The prover selects a random permutation, π, of the vertices V, and commits (using a perfectly-binding commitment scheme) the entries of the adjacency matrix of the resulting permutated graph. That is, it sends a q-by-q matrix of commitments so that the $(\pi(i), \pi(j))^{th}$ entry is a commitment to 1 if $(i,j) \in E$, and is a commitment to 0 otherwise.

Round-2. The verifier uniformly selects a bit $b \in \{0,1\}$ and sends it to the prover.

Round-3. If $b = 0$ then the prover sends π to the verifier along with the revealing of all commitments (and the verifier checks that the revealed graph is indeed isomorphic to G via π); If $b = 1$, the prover reveals to the verifier only the commitments to entries $(\pi(i), \pi(j))$ with $(i, j) \in C_G$ (and the verifier checks that all revealed values are 1 and the corresponding entries form a simple q-cycle).

Definition 4 (collision-resistant hash function). *Let $\mathcal{H}$ be a family of hash functions: $\mathcal{K} \times D \to R$, where D is the domain of $\mathcal{H}$ and R is the range of $\mathcal{H}$ and $\mathcal{K}$ denotes the key space of $\mathcal{H}$. For a particular key $K \in \mathcal{K}$, $\mathcal{H}_K$: $D \to R$ is defined as $\mathcal{H}(K, \cdot)$. We say that $\mathcal{H}$ is collision-resistant, if for a randomly chosen key $K \leftarrow \mathcal{K}$ and for any (even non-uniform) polynomial-time algorithm A the probability that A, given the randomly chosen key K as its input, outputs two distinct points x_1 and x_2 in D such that $\mathcal{H}_K(x_1) = \mathcal{H}_K(x_2)$ is negligible. The probability is taken over the choice of K and the randomness of A.*

Identity-based trapdoor commitments (IBTC). In the setting of IBTC, a single "master-key" is made public. Additionally, all parties can obtain a private-key that is associated to their party identifier. Intuitively, an IBTC scheme is a commitment scheme with the additional property that a committer who *knows* the receiver's secret-key can *equivocate* commitments (i.e., it can open up commitments to any value) at its wish. Furthermore, it should hold that an adversary that obtains the secret-keys of multiple parties, still should not be able to violate the binding property of commitments sent to parties for which it has not obtained the secret-key. We refer to [8] for the formal definition of IBTC. IBTC constructions are presented in [1] in the random oracle model, and in [8] under any one-way function in the standard model.

3 On the Hardness of Achieving Generalized UC

In this section, we first briefly recall the UCZK protocol implied by the works of [8,9,11]. Then, we demonstrate, by a simple attack, the hardness (subtleties) of achieving generalized UC security, even with trusted setups and/or a majority of honest parties. Finally, we discuss the underlying reasons.

3.1 The UCZK Protocol Implied by [8,9,11]

The work of [8] reestablishes the general feasibility of UC in the ACRS model as follows: Firstly, it implements a UC-secure commitment scheme in the ACRS model (reminiscent of the traditional UC-secure commitments of [11] in the traditional CRS model). Then using the UC-secure commitments as building tool, UCZK can be implemented by following the construction of traditional UCZK in the traditional CRS model of [9]. Finally, with UCZK as a key tool, general feasibility of UC for any cryptographic functionality can be finally reestablished with

some other techniques. The UCZK protocol implied by [8,9] is a modification of Blum's protocol for DHC (recalled in Section 2), with the statistically-binding commitments used in its first-round are replaced by the UC-secure commitments of [8] in the ACRS model. The following is the brief structure of the UCZK protocol implied by [8,9] in the ACRS model (in which we omit the augmented CRS for presentation simplicity).

Common input: $x \in L$, where L is an $\mathcal{NP}$-language with $\mathcal{NP}$-relation R_L.
Prover's private input: w s.t. $(x, w) \in R_L$.
Main-proof stage: consists of two phases:

Phase-1: Phase-1 is a coin-tossing protocol, in which the verifier V commits to a random string using the IBTC scheme and the prover P responds with a public random string. Denote by K the output of the coin-tossing.

Phase-2: Phase-2 is a modified Blum's protocol for DHC. Specifically, the statistically-binding bit commitment in the first-round of Blum's protocol for DHC is replaced as follows: For a bit $b \in \{0, 1\}$, the prover P commits to b as follows. It first forms $(c, d) = IBTC(b)$ (i.e., commits b by running the IBTC scheme, where c is the commitment and d is the decommitment information); then if $b = 0$ it forms $e = PKE_K(d)$ (i.e., it encrypts the decommitment information d by a public-key encryption scheme with the coin-tossing output K as the public-key), otherwise (i.e., $b = 1$) e is set to be a random value.

The above protocol is shown to be GUC secure in the ACRS model, assuming that the arbitrary protocols (environment) can share state information with it only via the trusted ACRS generation functionality. That is, any state information that will be (potentially maliciously) shared by the challenge protocol with the environment can be derived from the common reference string (in a way, the adversary is essentially restricted to only malleate the common reference string). In a sense, the general UC feasibility proposed in [8] is w.r.t. "predictable" environment and adversarial strategies. To illustrate the potential weakness of assuming predictable environment and adversarial strategies, we describe a simple state-information-sharing attack that violates the POK property of the above UCZK protocol.

3.2 A Simple State-Information-Sharing Attack

The idea is simple, the protocol designed to be composed with the UCZK of [8] is essentially its Phase-2 subprotocol. Specifically, consider the following protocol $\langle P', V' \rangle$:

Common input: $x \in L$.
Prover's private input: w s.t. $(x, w) \in R_L$.
Main-proof stage: consists of two phases:

Phase-1: The verifier V' sends a string K to the prover P', where K is a random encryption public-key.

Phase-2: Phase-2 is the modified Blum's protocol for DHC. Specifically, the statistically-binding bit commitment in the first-round of Blum's protocol for DHC is replaced as follows: For a bit $b \in \{0,1\}$, the prover P' commits to b as follows. It first forms $(c,d) = IBTC(b)$ (i.e., commits b by running the IBTC scheme, where c is the commitment and d is the decommitment information); then if $b = 0$ it forms $e = PKE_K(d)$ (i.e., it encrypts the decommitment information d by the encryption public-key K received in Phase-1), otherwise (i.e., $b = 1$) e is set to be a random value.

Then, the state-information-sharing attack proceeds as follows:

- Static corruption: the adversary $\mathcal{A}$ corrupts P (the prover of the UCZK protocol) and V' (the verifier of the above protocol) at the onset of computation.
- In the execution of the UCZK protocol, referred as the first session for presentation simplicity, $\mathcal{A}$ works just as the honest prover does until Phase-1 (i.e., the coin-tossing) finishes. Denote by K the output of the coin-tossing in the first session. $\mathcal{A}$ suspends the first session.
- Now, $\mathcal{A}$ runs the above protocol, referred as the second session, and sends to the prover P' the value K as the Phase-1 message of the second session, where K is the output of the coin-tossing of the first session.
- Then, $\mathcal{A}$ just copies messages received from P' in the second session to the verifier V of the first session.

Clearly, $\mathcal{A}$ can successfully finish the execution of the first session, but without knowing any corresponding $\mathcal{NP}$ witness to the statement being proved in the first session.

3.3 Comments and Discussion

We remark that the above state information sharing attack is very simple and even unnatural, which just copies messages from one session to another session. But, in general it is possible that messages sent in one session by the adversary are not directly copied from another session, but messages in the concurrent interleaving sessions are malicious related. And, also the protocol to be composed with the UCZK of [8] could be potentially a naturally existing and widely used protocol (as demonstrated in [23]). The simple attack is also helpful for a precise understanding of the GUC feasibility established in [8] (i.e., it is conditioned on certain limitation on the environment and adversarial strategies), and for realizing potential weakness of such limitation to prevent potential misinterpretation in applications due to high system complexity.

Typically, any message received by an honest player in the challenge protocol could be maliciously dependent on not only the transcript of the challenge protocol, but also, more harmfully, the whole transcript of the external arbitrary protocols running concurrently with the challenge protocol. That is, it is a natural adversarial strategy of the adversary that it manages to make the

challenge protocol maliciously related to the external environment by concurrent interleaving and malleating attacks, thus sharing some maliciously-related state-information amongst them. The key point here is, the external environment (i.e., the arbitrary protocols running concurrently with the challenge protocol) and adversarial strategies employed by the adversary are *unpredictable*. Even if a given challenge protocol is proved to satisfy certain level of composability, like the UCZK implied by [8,9,11], but it does not imply that the subprotocols or building tools used by the challenge protocol guarantee the same level of composability. An adversary could maliciously relate the (weaker) subprotocols or building tools to the external environment *in an unpredictable way*, so that some (*not necessarily global but possibly local and session-specific*) state information is maliciously shared amongst them. The key observation here is, for a protocol to be GUC secure, any subprotocol used by the protocol, and even each message sent in the protocol, should also to be GUC-secure. This shows the hardness of achieving generalized universal composability, i.e., composability with arbitrary protocols (unpredictable environment) that is the original intuition and motivation for UC, even with trusted setups and a majority of honest players.

4 Fixing Suggestion with Source/Session-Authentic Commitments

According to above discussion, to achieve a GUC-secure protocol we may require any subprotocol used by the protocol, and even each message sent in the protocol, are also to be GUC-secure, i.e., composable with arbitrary protocols (unrestricted and unpredictable environment). In other words, we need the subprotocols used by the protocol, and even each message sent in the protocol, to be source/session-authentic, that is, they should not be maliciously generated (malleated) by the adversary from the unpredictable environment. We note this is a very strong requirement. With respect to the specific UCZK implied by [8], we propose some fixing suggestion, which provides very useful session-authentic non-malleability in many applied scenarios (though it may still not fully provide GUC security).

The idea is to augment the underlying IDBC scheme used in the UCZK of [8] as follows. For any message m to be committed, we do not commit to m directly by running $IBTC(m)$ as did in the UCZK of [8]. Rather, we mask the message m, together with some (public) source/session-specific auxiliary information aux (in particular, aux could include the session ID, the committer ID, the receiver ID, and some partial transcript), by a collision-resistant hash function $\mathcal{H}$, and commits to the hashed value. Specifically, in this case, the commitment is $IBTC(\mathcal{H}(m, aux))$. We call such commitments *source/session-authentic ID-based commitments*.

We note that for any commitment scheme Com, any message m and any (public) auxiliary information aux, source/session-authentic commitments $Com(\mathcal{H}(m, aux))$ does not lose the hiding and binding properties, in comparison with $Com(m)$. We roughly argue about this fact, with the formal analysis

deferred to the full version of this work. Firstly, the hiding property of source/session-authentic commitments is directly from that of the original commitment scheme Com. For the binding property, we note that the ability to equivocate source/session-authentic commitments implies the ability to break the collision-resistance of the underlying hash function $\mathcal{H}$. This in particular means that the augmented version of the UCZK of [8], with the $IBTC$ replaced by source/session-authentic $IBTC$, remains the same UC security in accordance with the UC formulation of [8]. But, the augmented version provides some additional non-malleability/composability guarantee, in particular, the attack demonstrated in Section 3 fails in this case.

On the usefulness of source/session-authentic commitments. The usefulness of the source/session-authentic commitments lies in the following observation: even if the original commitment scheme Com is very weak to against man-in-the-middle attacks, i.e., *not non-malleable*, the source/session-authentic commitments via the *hash-then-commit* paradigm still provide very reasonable and practical non-malleability guarantee. This is from the fact that: given a source/session-authentic commitment c (seen from one session), it could be possible for the adversary to maliciously malleate c into a commitment c' (possibly in another concurrent session). But, it is intuitively hard for the adversary to successfully open c' to satisfy the additional source/session-specific restriction via the collision-resistant hashing. Thus, we suggest source/session-authentic commitments could be very useful, in particular in many applied scenarios involving commitments, to provide practical and reasonable non-malleability/composability guarantee against man-in-the-middle attacks.

Acknowledgements. We thank Ran Canetti for kind clarifications about UC in the CRS model, and referring us to [8].

References

1. G. Atenise and B. De Medeiros. Identity-Based Chameleon Hash and Applications. Cryptology ePrint Archive, Report No. 2003/167.
2. M. Blum. Coin Flipping by Telephone. In *proc. IEEE Spring COMPCOM*, pages 133-137, 1982.
3. M. Blum. How to Prove a Theorem so No One Else can Claim It. In Proceedings of the International Congress of Mathematicians, Berkeley, California, USA, 1986, pp. 1444-1451.
4. M. Bellare and O. Goldreich. On Defining Proofs of Knowledge In *E. F. Brickell (Ed.): Advances in Cryptology-Proceedings of CRYPTO 1992, LNCS 740*, pages 390-420, Springer-Verlag, 1992.
5. M. Bellare and O. Goldreich. On Probabilistic versus Deterministic Provers in the Definition of Proofs Of Knowledge. Electronic Colloquium on Computational Complexity, 13(136), 2006. Available also from Cryptology ePrint Archive, Report No. 2006/359.

6. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136-145, 2001.
7. R. Canetti. Security and Composition of Cryptographic Protocols: A Tutorial. *Distributed Computing column of SIGACT News*, Vol. 37, Nos. 3 and 4, 2006. Available also from Cryptology ePrint Archive, Report 2006/465.
8. R. Canetti, Y. Dodis, R. Pass and S. Walfish. Universal Composable Security with Global Setup. TCC 2007, to appear. Available from: Cryptology ePrint Archive, Report No. 2006/432.
9. R. Canetti and M. Fischlin. Universal Composable Commitments. In *Advances in Cryptology-Proceedings of CRYPTO 2001, LNCS 2139*, pages 19-40. Springer-Verlag, 2001.
10. R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universal Composition Without Set-Up Assumptions. In *E. Biham (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 2003, LNCS 2656*, pages 68-86. Springer-Verlag, 2003.
11. R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *ACM Symposium on Theory of Computing*, pages 494-503, 2002.
12. R. Canetti and T. Rabin. Universal Composition with Joint State. In *Advances in Cryptology-Proceedings of CRYPTO 2002, LNCS 2729*, pages 265-281, Springer-Verlag, 2003.
13. J. A. Garay, P. MacKenzie and K. Yang. Strengthening Zero-Knowledge Protocols Using Signatures. *Journal of Cryptology*, to appear. Preliminary version appears in *E. Biham (Ed.): Advances in Cryptology-Proceedings of EUROCRYPT 2003, LNCS 2656* , pages 177-194. Springer-Verlag, 2003.
14. O. Goldreich. *Foundation of Cryptography-Basic Tools*. Cambridge University Press, 2001.
15. O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design. In *IEEE Symposium on Foundations of Computer Science*, pages 174-187, 1986.
16. O. Goldreich, S. Micali and A. Wigderson. How to Prove all $\mathcal{NP}$-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design. In *A. M. Odlyzko (Ed.): Advances in Cryptology-Proceedings of CRYPTO 1986, LNCS 263*, pages 104-110, Springer-Verlag, 1986.
17. O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing But Their Validity or All language in $\mathcal{NP}$ Have Zero-Knowledge Proof Systems. *Journal of the Association for Computing Machinery*, 38(1): 691-729, 1991. Preliminary version appears in [15,16].
18. J. Hastad, R. Impagliazzo, L. A. Levin and M. Luby. Construction of a Pseudorandom Generator from Any One-Way Function *SIAM Journal on Computing*, 28(4): 1364-1396, 1999.
19. Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *IEEE Symposium on Foundations of Computer Science*, pages 394-403, 2003.
20. Y. Lindell. Lower Bounds for Concurrent Self Composition. In *M. Naor (Ed.): Theory of Cryptography (TCC) 2004, LNCS 2951*, pages 203-222, Springer-Verlag, 2004.

21. M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology*, 4(2): 151-158, 1991.
22. R. Pass. On Deniabililty in the Common Reference String and Random Oracle Models. In*D. Boneh (Ed.): Advances in Cryptology-Proceedings of CRYPTO 2003, LNCS 2729*, pages 316-337, Springer-Verlag 2003.
23. A. C. C. Yao, F. F. Yao and Y. Zhao. A Note on Universal Composable Zero-Knowledge in the Common Reference String Model. Appears in the same proceedings.

t-Private and Secure Auctions

Markus Hinkelmann[1], Andreas Jakoby[1], and Peer Stechert[2]

[1] Institut für Theoretische Informatik, Universität zu Lübeck, Germany
{hinkelma,jakoby}@tcs.uni-luebeck.de
[2] Fachgruppe Didaktik der Informatik und E-Learning, Universität Siegen, Germany
stechert@die.informatik.uni-siegen.de

Abstract. In most of the used auction systems the values of bids are known to the auctioneer. This allows him to manipulate the outcome of the auction. Hence, one is interested in hiding these values. Some cryptographically secure protocols for electronic auctions have been presented in the last decade. Our work extends these protocols in several ways. Based on garbled circuits, i.e. encrypted circuits, we present protocols for sealed-bid auctions that fulfill the following requirements:

1. Protocols are information-theoretically t-private for *honest but curious parties*.
2. The number of bits that can be learned by *active adversaries* is bounded by the output length of the auction. Hence, if the result of the auction has to remain unchanged, then we present protocols that are secure against malicious attacks.
3. The computational requirements for participating parties are very low: only random bit choices and bitwise computation of the XOR-function are necessary.
4. The protocols are perfectly correct, i.e. they have a zero probability of failure.

Note that one can distinguish between the protocol that generates a garbled circuit for an auction and the protocol to evaluate the bids in an auction based on the garbled circuit. Usually previous papers are focused on the problem of evaluating the bids of an auction. In this paper we address both problems. In addition to the generalization of the concept of garbled circuit we will present a t-private protocol for the construction of a garbled circuit that reaches the lower bound of $2t + 1$ parties and a more randomness efficient protocol for $(t + 1)^2$ parties.

Finally we will present a strategy that allows new bidders to join a running auction or to change their bids dynamically. Our goal is that all bidders who do not change their bids are allowed to stay inactive in the process of bid changing.

1 Introduction

Traditional auctions involve an auctioneer and numerous bidders who want to sell or buy an item, respectively. In a secure electronic auction system, secure EAS for short, a bid has to remain hidden for all other bidders and for the auctioneer if it is not a part of the output. In particular, the auctioneer has to be prevented from learning the bids.

There exist numerous different auction types and models: The *English* or ascending auction is the most common auction type. Here, in each round a bidder can increase

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 486–498, 2007.
© Springer-Verlag Berlin Heidelberg 2007

his previous bid. At the end, the highest bidder takes the item and pays the price he has bid. Hence, the auctioneer learns the bids of all bidders. In a *Dutch* or descending auction, the potential price decreases over time. The first person who places a bid takes the item and has to pay the current price. Note that the privacy of other bidders' bids are preserved. In a *sealed-bid* auction, every player turns in a secret bid. The auctioneer opens all bids and computes the winner. Again, the auctioneer learns the bids of all bidders. Dealing with sealed-bid we distinguish between *first-price* and *second-price* auctions. In a first-price auction, the highest bidder takes the item and pays the price of his bid, i.e. the highest bid. In a second-price auction also called *Vickrey auction* the highest bidder takes the item and pays the second highest bid (see e.g. [10]).

In this paper we present protocols for sealed-bid auctions which keep the privacy of the bidders' bids even if a collusion of parties tries to attack the protocol. Our work relies on two concepts. First, it relies on the cryptographically privacy preserving auction scheme of Naor et al. [24], using the garbled circuit (GC) construction, i.e. encrypted Boolean circuits. Second, it relies on the concept of perfect privacy, i.e. on the concept of privacy in the information-theoretic sense [2,7]. In [17] Ishai and Kushilevitz showed how to use the model of GCs to compute a function in a perfect private way. We will generalize both concepts:

1. We present *protocols to construct GCs t-privately*: The construction does not leak more information to any collusion of up to t parties than the GC itself.
2. We generalize the concept of garbled circuits such that the *garbled circuits will be t-private* according to the inputs of the circuit. This implies that any collusion of up to t parties does not get more information about the inputs of the circuit than the collusion can deduce from result of the circuit.

Related work concerning Electronic-Auction-Systems: In 1996, Franklin and Reiter [12] proposed the first auction scheme that has a privacy contribution with regard to the bidders whereas the auctioneer is allowed to learn all bids. They also demand for a deposit of digital cash for each player to ensure non-repudiation in their cryptographic auction scheme. Sadeghi, Schunter, and Steinbrecher [27] presented a combinatorial auction that sells multiple interdependent items by proceeding multiple rounds. They consider the German UMTS auction as an example, where every winner had to win at least two licenses. In this model, public key cryptography is used to obtain verifiable and private bidding, and the possibility to repudiate a bid is mentioned whereby the involved bidder has to pay a fine.

Kikuchi, Harkavy, and Tygar [20] introduced a model based on Shamir's secret sharing scheme [26]. In each round, a bid-vector of k prices is distributed to the players who can mark all prices they want to bid. Kikuchi et al. [15] also described sealed-bid auctions via a verifiable secret sharing scheme [8,2]. Their approach deals with n bidders and m auctioneers and uses an error correcting code to share the secrets. Accountability is reached with public key encryption.

Kurosawa and Ogata [22] dealt with a bit-slice auction circuit. Traditional techniques to evaluate an auction compare the incoming bids one-by-one, but their approach alternatively compares the bids bit-by-bit, beginning with the most significant one. Additionally, they combine this approach with the mix and match method of Jakobsson and

Juels [18] that uses a homomorphic encryption scheme. Omote and Miyaji [25] proposed a second-price sealed-bid auction that satisfies public verifiability of the auction without revealing the highest bid. To achieve this, they use a cryptographic primitive and two auction managers. Brandt [4,5] introduced a cryptographic auction protocol without a special trusted party. Full privacy is achieved by distributing shares of every bid via Shamir's secret sharing scheme. Cachin [6] presented a scheme based on homomorphic encryption, where an auctioneer wants to receive at least a certain price A for his item and a bidder who wants to pay at most B. To answer the question, whether the deal takes place, an oblivious third party is introduced, that neither learns anything about both values nor about the result, i.e. whether $A > B$.

Naor et al. [24] presented an EAS based on GCs. Using pseudo-random generators these circuits allow to define a cryptographical secure and verifiable EAS. To construct the GCs the authors introduce a new party called auction issuer. They assume that the auction issuer does not collude with the auctioneer or a bidder. Juels and Szydlo [19] relied directly on the GC construction of Naor et al. [24]. The proxy oblivious transfer of Naor et al. has the disadvantage that the correct input of a bidder is not verifiable by other parties. Juels and Szydlo introduce a verifiable proxy oblivious transfer resulting in less computation for the bidders.

Related work concerning Private Computation: Private computation was introduced by Yao [29]. He considered the problem under cryptographic assumptions. Private Computation with unconditional, i.e. information-theoretical, security has been introduced by Ben-Or et al. [2] and Chaum et al. [7]. Kushilevitz et al. [23] proved that the class of Boolean functions that have linear size circuit is exactly the class of functions that can privately be computed using a constant number of random bits.

Franklin and Yung [13] investigated the role of the connectivity of the underlying network in private computations. Bläser et al. [3] completely characterized the class of privately computable Boolean functions, if the underlying network is connected but not 2-connected. In particular, no non-degenerate function can privately be computed if the network consists of three or more blocks. On networks with two blocks only a small class of functions can privately be computed. This result has been generalized for non-Boolean functions by Beimel [1]. He has shown that only functions with a restricted type of communication matrix can be computed on a non-2-connected network. One can easily verify that most types of auctions do not fulfill these restrictions of the function. Hence, the underlying network has to be 2-connected.

Chaum et al. [7] proved that any Boolean function can privately be computed, if at most one third of the participating parties are dishonest. For this model, Ben-Or et al. [2] proved that any n-ary Boolean function can be computed $\lfloor \frac{n-1}{2} \rfloor$-private, i.e. at most $\lfloor \frac{n-1}{2} \rfloor$ players collude. Chor and Kushilevitz [9] showed that if a function can be computed at least $\frac{n}{2}$-privately, then it can be computed n-privately as well.

Randomizing polynomials are introduced in [16] by Ishai and Kushilevitz as a new algorithmic approach for round-efficient private computations with low error probability. In [17] they have shown that all functions can privately be computed in a constant number of rounds and 0 error probability. Damgård and Ishai presented a constant-round protocol for general multiparty computation that uses randomized polynomials, black-box pseudorandom generators, and Shamir shares [11]. The protocol uses

multiplications and additions over a finite field $F = \mathrm{GF}(2^k)$ with security parameter k for construction and evaluation of the polynomials. Damgård and Ishai distinguish between input players, output players, and servers, i.e. parties which construct the randomized polynomials. Hence, this protocol can be used to implement auctions. It is cryptographically secure against an active adversary corrupting $t < n/2$ servers.

Our Contributions: For private electronic auction systems one has to find a concept and protocols for evaluating auctions preserving the privacy of the bidders and the bids.

Our Concept: We generalize the concept of GCs in order to evaluate a circuit t-privately, i.e. no collusion of up to t parties can deduce any knowledge about the inputs of the circuit that cannot be deduced from the result of the circuit and the input of the colluding parties. Focusing on EASs fulfills the following requirements:

– In contrast to previous EASs our system is information-theoretically t-private.
– Some protocols presented in the literature may fail with small probability. GCs allow to evaluate an auction in a perfectly correct way.

In most papers cited above verification is only investigated regarding the bidders and their secret inputs. In our EAS it is possible that all parties t-privately verify the correctness of the auction evaluation. Furthermore, our auction scheme guarantees that the total number of bits of information revealed by a collusion of up to t malicious parties that try to stay undetected is bounded by the number of output-bits of the auction.

The Protocols: (1) We present information-theoretically t-private protocols to construct a GC $\Gamma(C)$ for a given circuit C. The first protocol requires $(t+1)^2$ parties and uses a number of random bits that is polynomial in $|\Gamma(C)|$ and t. The second protocol is based on the first protocol and reaches the lower bound of $2t+1$ parties. In return the second protocol uses an exponential number of additional random bits in t. (2) We present two information-theoretically t-private protocols for the bidding process.

In our protocols there are four kinds of parties: (i) auctioneer who evaluates the auction, (ii) bidders, (iii) auction issuers who determine the encryption of the auction, and (iv) slaves who perform the encryption. If the parties have access to a source of random bits, the computation performed by auction issuers, bidders and slaves can be implemented by circuits of depth $\mathcal{O}(\log t)$ that consists of binary XOR-gates and one level of gates to multiplex one bit out of two. Thus, the computational requirements are very low.

Introducing some minor changes to these protocols one can use them to translate intermediate data of an evaluation process of a GC into intermediate data of the evaluation process of another GC of the same circuit. Based on this observation our protocols allow dynamic sealed-bid auction, i.e. a bidder can change his bid at will or they can enter an auction that has already started and no further bidder is involved in this procedure.

All protocols presented in this paper are information-theoretically t-private. In return the GC requires a large number of random bits in the encryption of the underlying circuit. But, the protocols for generating the GCs might be useful for generating GCs that are cryptographically t-private. As one can see in [24,11] these encryptions are often more efficient than information-theoretically private encryptions. Hence, this paper

will be a first step towards an efficient implementation of t-private EASs with very low requirements for hardware. Our approach is based on [28].

This work is organized as follows: In Section 2 we present some basic definitions and notations of private computation. We will redraw the privacy preserving cryptographic auction of Naor et al. Furthermore, we discuss randomizing polynomials that can be used as an evaluation technique for GCs. Section 3 presents information-theoretically t-private, static auction protocols. In Section 4 we analyze the security of our protocol against active attacks. Section 5 deals with dynamic aspects of the EAS. The concluding Section 6 summarizes the results.

2 Notation and Preliminaries

Throughout this work, we will use $\oplus$ to denote the (bit-wise) XOR-operation on Boolean values and binary strings, as well as on vectors and matrices of binary strings.

Let $x = x[1]x[2]\dots x[n] \in \{0,1\}^n$ be a binary string of length n. For a set $J = \{j_1, \dots, j_k\}$ where $1 \leq j_1 < j_2 < \dots < j_k \leq n$ let $x[J] = x[j_1]\dots x[j_k]$. We extend this notation to arbitrary sequences. Let $C = (c_1, \dots, c_n)$. Then $C[J]$ denotes the subsequence $(c_{j_1}, \dots, c_{j_k})$. The operator $\circ$ denotes the concatenation of strings. For $i, j \in \mathbb{N}$, let $[i,j] := \{i, \dots, j\}$. All logarithms in this paper are to the base 2.

An n-input l-output circuit $\mathcal{C}_n^l$ is described by a DAG $G = (V, E)$. V contains $2n$ nodes of in-degree 0, the inputs $x_1, \bar{x}_1, \dots, x_n, \bar{x}_n$. The remaining nodes have in-degree 2 and are called gates. The gates $g_1, \dots, g_m$ are labeled by either AND or OR. G has l nodes with out-degree 0 called output-gates. For convenience we write $\mathcal{C}$ instead of $\mathcal{C}_n^1$ and we use the variables x_i synonymously to the corresponding input values. If the input of a circuit is fixed by the context we denote the value of a gate g by b_g. The value b_e of a wire e is determined by the value of the gate g the wire comes from resp. by the input x_i if e is an input wire.

Privacy and Security: In this work we consider the model of honest but curious and the model of malicious parties. We do not want any party to learn more about the inputs of other parties than it can deduce from the result of the function and its own input. We assume that every party has access to a random tape. Parties can send messages to other parties via point-to-point communication using secure channels. Computing a function f on the input $x = (x_1, \dots, x_n)$ we assume that x_i is the private input of party P_i.

Definition 1 (t-Privacy). *For a subset of parties $U \subseteq V$ let C_U resp. R_U denote the random variables of the communication string resp. of the random string seen by parties of U and let c_U be a particular communication string. An n-party protocol $\mathcal{P}$ for computing a function $f : \Sigma^n \to \Sigma$ is private with respect to the parties of U, if for every pair of input vectors x, y with $f(x) = f(y)$ and $x[U] = y[U]$, for every sequence of random strings r_U provided to the parties of U, $Pr[C_U = c_U | R_U = r_U, x] = Pr[C_U = c_U | R_U = r_U, y]$ where all parties follow protocol $\mathcal{P}$. A protocol is t-private if it is private with respect to any subset U of at most t parties. A protocol is called private if it is 1-private.*

In other words, a protocol is called t-private, $1 \leq t \leq n-1$, if no collusion of t parties learns anything about the distributed secret that cannot be deduced from the result and

the input of the members of the collusion while honestly participating in the protocol. From [3,1] we can conclude that most auction protocols cannot be run privately on non-2-connected networks. Therefore, we demand a communication network to be at least 2-connected.

Alternatively, one can define t-privacy by using conditional entropy. Here the inputs of parties are represented as a discrete random variable X. Let U be a subset of parties. We define the **leakage** of information of a protocol with respect to U as the minimum value τ such that for any x, c_U and r_U it holds that

$$\mathrm{H}[X|f(X), X[U] = x[U], r_U] \; - \; \mathrm{H}[X|f(X), X[U] = x[U], r_U, c_U] \; \leq \; \tau \; .$$

The definition above corresponds to the conditional mutual information between X and C_U. A protocol is t-private if its leakage is 0 with respect to every subset U of at most t parties. We call a protocol $\mathcal{P}$ **(t, τ)-secure** if for every subset U of at most t parties and for every protocol $\mathcal{P}'$ where the parties of U follow $\mathcal{P}'$ and the honest parties in $V \setminus U$ follow the protocol $\mathcal{P}$ the leakage is bounded by τ. Note that malicious parties may run an arbitrary attack.

The NPS Protocol: In this section, we discuss the cryptographic EAS of Naor et al. [24], that is based on Yao's garbled circuits and pseudo-random generators [30,14]. GCs are encrypted versions of standard Boolean circuits where the Boolean values are replaced by encrypted values. Knowing the encrypted input one can evaluate the circuit gate-by-gate and produce the garbled output without learning anything about the original input and intermediary results. To achieve privacy of an auction Naor et al. [24] introduce a third party: the *auction issuer AI*. This party generates the GC and they assume that *AI* does not conspire with any other party. To send the encrypted bids to the auctioneer the bidders use a protocol realizing the *proxy oblivious transfer*. For one bit bids the proxy oblivious transfer is defined as follows: Each bidder has his private value $x_i \in \{0, 1\}$ and *AI* provides two encryptions W^0, W^1 for the bit. The goal is to send W^{x_i} to the auctioneer such that neither the bidder learns W^0 or W^1 nor the *AI* learns x_i. Furthermore, the auctioneer does not learn anything about x_i or W^{1-x_i}. Using the GC, the auctioneer can evaluate the auction. We will refer to this protocol as the NPS-protocol.

The NPS-protocol provides 1-privacy, only. I.e. a party cannot break the secrets of the other parties. But if the *AI* and the auctioneer collude, both will gain knowledge about all bids.

Garbled Circuits: In this section, we shortly discuss GCs. A GC consists of a circuit of encrypted gates. To encrypt a gate g we XOR the truth table of g by a randomly generated table W_g and permute their entries. To compute the result of the encrypted gate we add to the result of predecessor gates the corresponding rows resp. columns of some random tables such that XOR-ing these random values allows us to compute exactly one entry of W_g. To keep track of the permutation of the truth table of g, we XOR the results of the predecessor gates by a random bit. Note that this allows us to encrypt the result of the predecessor gates and simultaneously to permute the truth table of g.

Let us now continue with a more formal description of GCs. We use random values of two types: For every wire e of a circuit C there exists a vector of random strings $W_e = (W_e^0, W_e^1)$ and for every gate g and every input bit x_i there exists a random bit

r_g resp. r_i. The length of the strings W_e^b is defined recursively from top to bottom: For every $b \in \{0,1\}$, every wire e and every gate g let $|W_e^b| = 0$ if e is an output wire and $|W_e^b| = 2(1 + \sum_{\text{output wire } e' \text{ of } g} |W_{e'}^0|)$ if e is an input wire of gate g.

Each string $W_e^b = W_e^{b,0} \circ W_e^{b,1}$ consists of two equally sized halves. In order to encrypt a Boolean value $x_i \in \{0,1\}$ on an input wire e, we use a random bit r_i and compute the polynomial $W_e^{x_i} \circ (x_i \oplus r_i)$. Note that $W_e^{x_i}$ can be written as $(x_i \cdot W_e^1) \oplus ((1 - x_i) \cdot W_e^0)$.

Let g be a gate with input wires e_1, e_2, output wires $o_1, \ldots, o_k$ and gate function $g(b_1, b_2)$ where $b_1, b_2 \in \{0,1\}$. We encrypt the values b_{g_i} by $c_{g_i} = b_{g_i} \oplus r_{g_i}$ where g_i denotes the predecessor of g with output wire e_i. Hence, we use one additional random bit r_g for every gate g. Each random bit r_{g_i} induces a permutation of the rows (resp. columns) of the garbled table in the succeeding gates. For each of the four possible values $c_{g_1}, c_{g_2} \in \{0,1\}$ of the preceding gates of g, we define a corresponding polynomial on strings of length $1 + \sum_{1 \le l \le k} |W_{o_l}^0|$. This polynomial defines the entry of the garbled table indexed by (c_{g_1}, c_{g_2}). Informally, these expressions encrypt the values of the (permuted) truth table of g. The garbled table is given by

$$Q_g \;=\; \begin{pmatrix} Q_g^{0,0} & Q_g^{0,1} \\ Q_g^{1,0} & Q_g^{1,1} \end{pmatrix}$$

with $Q_g^{a,b} = W_{e_1}^{a \oplus r_{g_1}, b} \oplus W_{e_2}^{b \oplus r_{g_2}, a} \oplus (W_g^{g(a \oplus r_{g_1}, b \oplus r_{g_2})} \circ (r_g \oplus g(a \oplus r_{g_1}, b \oplus r_{g_2})))$ and $W_g^b = W_{o_1}^b \circ \ldots \circ W_{o_k}^b$ where g_1 denotes the left and g_2 the right predecessor of g.

In every entry of the table, we store a string $W_g^{g(a \oplus r_{g_1}, b \oplus r_{g_2})}$ that is used as a key in the table for the succeeding gate. To decrypt a table we use only one half of W_{g_1}, resp. W_{g_2}. Thus, the length of these strings grows exponentially in the depth of the corresponding circuit. But if the circuit has logarithmic depth the total length of the strings in the GCs is only polynomial in these size of the circuit.

Knowing the *encryption* $W_{e_1}^{b_{g_1}}, W_{e_2}^{b_{g_2}}$ and c_{g_1}, c_{g_2} of the input of an gate g, one can XOR each of the suitable halves of $W_{e_1}^{b_{g_1}}, W_{e_2}^{b_{g_2}}$ with the garbled table entry $Q_g^{c_{g_1}, c_{g_2}}$ and gets the encryption of b_g and b_{o_i} for all output wires o_i of g without learning anything about the hidden Boolean value. That is,

$$Q_g^{c_{g_1}, c_{g_2}} \oplus W_{e_1}^{b_{g_1}, c_{g_2}} \oplus W_{e_2}^{b_{g_2}, c_{g_1}} \;=\; W_g^{g(c_{g_1} \oplus r_{g_1}, c_{g_2} \oplus r_{g_2})} \circ (g(c_{g_1} \oplus r_{g_1}, c_{g_2} \oplus r_{g_2}) \oplus r_g).$$

Each expression $W_{o_i}^h$ for a Boolean variable $h \in \{0,1\}$ can be represented by $(h \cdot W_{o_i}^1) \oplus ((1-h) \cdot W_{o_i}^0)$. The output gates g of the circuit have the values $r_g \oplus b_g$.

This construction is closely related to the construction of Ishai and Kushilevitz [17]. In [17] the authors presented a construction of GCs based on Boolean formulas. Our construction above is an extension of this approach to Boolean circuits. The proof of correctness and privacy follows analogously to the case of Boolean formulas.

We will now present an alternative way to generate these garbled tables. This strategy will be used in our protocol to generate garbled tables t-privately. For every gate g and every wire e we define $\widehat{W}_e = (W_e^0, W_e^1)$ and

$$\widehat{Q}_g \;=\; \begin{pmatrix} \widehat{Q}_g^{0,0} & \widehat{Q}_g^{0,1} \\ \widehat{Q}_g^{1,0} & \widehat{Q}_g^{1,1} \end{pmatrix} \text{ with } \widehat{Q}_g^{a,b} \;=\; W_g^{g(a,b)} \circ (r_g \oplus g(a,b)) \,.$$

Let $$(W^0_{e_1}, W^1_{e_1}) \otimes (W^0_{e_2}, W^1_{e_2}) = \begin{pmatrix} W^{0,0}_{e_1} \oplus W^{0,0}_{e_2} & W^{0,1}_{e_1} \oplus W^{1,0}_{e_2} \\ W^{1,0}_{e_1} \oplus W^{0,1}_{e_2} & W^{1,1}_{e_1} \oplus W^{1,1}_{e_2} \end{pmatrix},$$

$$\Pi_{r_1,r_2} \begin{pmatrix} A^{0,0} & A^{0,1} \\ A^{1,0} & A^{1,1} \end{pmatrix} = \begin{pmatrix} A^{r_1,r_2} & A^{r_1,\bar{r}_2} \\ A^{\bar{r}_1,r_2} & A^{\bar{r}_1,\bar{r}_2} \end{pmatrix},$$

and $\Pi_r(W^0_e, W^1_e) = (W^0_e, W^1_e)$ for $r = 0$ and $\Pi_r(W^0_e, W^1_e) = (W^1_e, W^0_e)$ for $r = 1$. Then, we have $Q_g = (\Pi_{r_1}(\widehat{W}_{e_1}) \otimes \Pi_{r_2}(\widehat{W}_{e_2})) \oplus \Pi_{r_1,r_2}(\widehat{Q}_g)$. For a given circuit $\mathcal{C}$ a GC is denoted by $\Gamma_{\mathcal{C}}$. If $\mathcal{C}$ is known from the context we will omit the index.

3 t-Private Garbled Circuit Construction

In this section we will present two protocols for generating a garbled auction circuit Γ t-privately. We say that a protocol **generates a GC t-privately** if it generates a GC Γ and while constructing Γ no collusion U of up to t parties can deduce any information about the chosen random values of Γ that cannot be deduced from Γ directly.

Basically our protocol works as follows: For each gate of a circuit we independently generate $t+1$ collections of the random bits used in the construction of the corresponding garbled table — each random collection is generated by a separate auction issuer. Instead of computing the garbled auction circuit by the auction issuers we proceed as follows: we introduce a field of $(t+1)^2$ slaves that is divided into $t+1$ columns where the i-th column is used to permute all truth tables according to the random permutation chosen by the i-th auction issuer. The truth tables and the corresponding W-strings of each auction issuer and of each gate move through all columns of the field. At the end the tables of all auction issuers are permuted by the same permutation. XOR-ing all corresponding tables gives us the desired t-private GC.

Our protocols are designed such that every party can simultaneously take part in the construction of the GC, bidding and evaluation. Hence, every party can simultaneously be the auctioneer, one auction issuer, one slave, as well as an arbitrary number of bidders without gaining additional knowledge about the bids of the other bidders. Hence, if it is not necessary to distinguish between the different entities we call them just parties.

In the following, we use the term **a party P generates a ℓ-share $w_1, \ldots, w_\ell$ of w** to denote that party P generates ℓ random strings $w_1, \ldots, w_\ell$ such that $w = w_1 \oplus \ldots \oplus w_\ell$.

Constructing Garbled Circuits: In the first strategy we will use a set of $t+1$ auction issuers AI_k with $k \in [1, t+1]$ and a set of $(t+1)^2$ sub-workers $S_{i,j}$ with $i, j \in [1, t+1]$ called slaves. Every auction issuer generates an independent GC Γ_k for a globally given auction circuit C. The slaves are used to combine these GCs. More precisely, AI_k generates two random strings $W^0_{k,e}, W^1_{k,e}$ for every wire e, a random bit $r_{k,g}$ for every gate g, and a random bit $r_{k,i}$ for every input variable x_i. For easier notion we associate to every wire e the random bit $r_{k,g}$ of the gate g resp. $r_{k,i}$ of the input variable x_i where the wire is coming from. Let $r_{k,e}$ denote this random bit. After generating the random values, AI_k generates the tables $\widehat{Q}_{k,g}$ of every gate g and the vectors $\widehat{W}_{k,e}$ of every wire e of C. For technical reasons we assume that the tables $\widehat{Q}_{k,g}$ with $k > 1$ do not contain the result of the gate, i.e. $\widehat{Q}^{a,b}_{k,g} = W^{g(a,b)}_{k,g} \circ r_{k,g}$ for $k > 1$ and $\widehat{Q}^{a,b}_{1,g} = W^{g(a,b)}_{1,g} \circ (r_{1,g} \oplus g(a,b))$.

Our goal is to construct Γ and thus all tables

$$Q_g \;=\; \left(\Pi_{r_{e_1}}(\oplus_{k=1}^{t+1}\widehat{W}_{k,e_1}) \;\otimes\; \Pi_{r_{e_2}}(\oplus_{k=1}^{t+1}\widehat{W}_{k,e_2})\right) \;\oplus\; \Pi_{r_{e_1},r_{e_2}}(\oplus_{k=1}^{t+1}\widehat{Q}_{k,g})$$

t-privately where $r_{e_1} = \oplus_{k=1}^{t+1} r_{k,e_1}$ and $r_{e_2} = \oplus_{k=1}^{t+1} r_{k,e_2}$. The protocol performs the following steps for every wire e and every gate g:

1. Every AI_k generates a $t+1$-share $r_{k,1,e},\ldots,r_{k,t+1,e}$ of every random bit $r_{k,e}$ and sends $r_{k,i,e}$ to all slaves $S_{h,i}$ with $h \in [1,t+1]$. All $S_{h,i}$ compute $s_{i,e} = \oplus_{k=1}^{t+1} r_{k,i,e}$.
2. Every AI_k generates the $t+1$-shares $\widehat{W}^1_{k,0,e},\ldots,\widehat{W}^{t+1}_{k,0,e}$ and $\widehat{Q}^1_{k,0,g},\ldots,\widehat{Q}^{t+1}_{k,0,g}$ of the random values $\widehat{W}_{k,e}$ and $\widehat{Q}_{k,g}$. Afterwards, he sends $\widehat{W}^i_{k,0,e}$ and $\widehat{Q}^i_{k,0,g}$ to $S_{i,1}$.
3. After receiving the values $\widehat{Q}^k_{i,j-1,g}$ and $\widehat{Q}^k_{i,j-1,g}$ for every $i \in [1,t+1]$ slave $S_{k,j}$ computes $\widehat{W}_{k,j,e} = \Pi_{s_{j,e}}(\oplus_{i=1}^{t+1}\widehat{W}^k_{i,j-1,e})$ and $\widehat{Q}_{k,j,g} = \Pi_{s_{j,e_1},s_{j,e_2}}(\oplus_{i=1}^{t+1}\widehat{Q}^k_{i,j-1,g})$ where e_1, e_2 are the input wires of g. If $j < t+1$, $S_{k,j}$ generates $t+1$-shares $\widehat{W}^1_{k,j,e},\ldots,\widehat{W}^{t+1}_{k,j,e}$ and $\widehat{Q}^1_{k,j,g},\ldots,\widehat{Q}^{t+1}_{k,j,g}$ of $\widehat{W}_{k,j,e}$ and $\widehat{Q}_{k,j,g}$ and sends $\widehat{W}^i_{k,j,e}$ and $\widehat{Q}^i_{k,j,g}$ to $S_{i,j+1}$.
4. The slaves $S_{k,t+1}$ in the last column compute $Q_{k,g} = (\widehat{W}_{k,t+1,e_1} \otimes \widehat{W}_{k,t+1,e_2}) \oplus \widehat{Q}_{k,t+1,g}$ where e_1, e_2 are the input wires of g and send $Q_{k,g}$ to the auctioneer A.
5. Finally, A computes $Q_g = \oplus_{k=1}^{t+1} Q_{k,g}$.

We call this protocol the **field-protocol**. The field-protocol can be simulated on a complete network of $2t+1$ parties.

Theorem 1. *The GC Γ of an auction circuit can be constructed t-privately by using $t+1$ auction issuers, $(t+1)^2$ slaves, and $O(|\Gamma|t^3)$ random bits where $|\Gamma|$ denotes the length of the binary representation of Γ. Moreover, there is a t-private protocol to build Γ for $2t+1$ parties using $\mathcal{O}((t+1)^{t+3}|\Gamma|)$ random bits.*

***t*-Private Bidding:** Let $x_{i,j}$ denote the jth bit of the bid x_i of bidder B_i and let e be a wire of the circuit C connected to $x_{i,j}$. Our protocol should give the auctioneer A access to $W_e^{x_{i,j}} \circ (r_e \oplus x_{i,j})$ in a t-private way. To simplify our analysis we assume that A publishes Γ and $W_e^{x_{i,j}} \circ (r_e \oplus x_{i,j})$. For the field-protocol we proceed as follows:

1. B_i computes a $t+1$-share $x^1_{i,j},\ldots,x^{t+1}_{i,j}$ of $x_{i,j}$ and sends $x^k_{i,j}$ to AI_k.
2. AI_k computes a $t+1$-share $c^1_{k,e},\ldots,c^{t+1}_{k,e}$ of $x^k_{i,j} \oplus r_{k,e}$ and sends $c^i_{k,e}$ to $S_{1,i}$.
3. $S_{1,i}$ sends $c^i_e = \oplus_{k=1}^{t+1} c^i_{k,e}$ to all slaves in the last row $S_{j,t+1}$.
4. All $S_{j,t+1}$ compute $c_e = \oplus_{i=1}^{t+1} c^i_e$ and send $\widehat{W}^{c_e}_{k,t+1,e} \circ c_e$ to A.
5. Finally, A computes $W_e^{x_{i,j}} \circ c_e = (\oplus_{k=1}^{t+1}\widehat{W}^{c_e}_{k,t+1,e}) \circ c_e$.

Theorem 2. *There exists a t-private protocol for an EAS with $(t+1)^2$ auction issuers using $O(t^3|\Gamma|)$ random bits and for an EAS with $2t+1$ auction issuers using $O((t+1)^{t+3}|\Gamma|)$ random bits.*

Since the evaluation of the GC on the encrypted inputs gives an encrypted output the result of the auction remain hidden. To decrypted the output all auction issuers AI_i have to publish their random bits $r_{i,g}$ for every output gate g.

4 Security Against Active Attacks

If a node is malicious (Byzantine), then it does not need to follow the instructions of the protocol. It can arbitrarily drop, add or change messages. We define an active collusion as a collusion of t malicious nodes that are under control of a single active adversary. Thus, the active adversary knows all information gathered by the colluding nodes and he can arbitrarily instruct these nodes.

One can think of different aims the adversary tries to achieve when attacking our field protocol: (1) The adversary may try to change the outcome of the auction. (2) The adversary is destructive, i.e. he tries to sabotage the execution or change the result of the computation by random. To make our field protocol resistant against these types of attacks we can construct the GC redundantly. The aim of the third kind of adversary is to collect as much of information about the inputs as possible by manipulating messages. In the following we present an upper bound for the leakage for the field-protocol.

Since the field protocol uses $t + 1$ auction issuers and $t + 1$ columns of slaves one can show that for every collusion U of at most t parties there exists at least one honest $AI_i \notin U$ such that no slave of the column $S_{\cdot,i}$ is in the collusion U. Furthermore, one of the rows $S_{j,\cdot}$ does not include a party of U. Note that the slaves in the same column perform identical operations. Hence, if $j \neq i$ we can permute the rows such that the slaves in $S_{i,\cdot}$ do not belong to the collusion without effecting the protocol. In the following we call the set that consists of $AI_i, S_{\cdot,i}, S_{i,\cdot}$, and all bidders not in U the trusted skeleton (TS) and the set of all remaining parties the extended collusion (EC). Even if the adversary controls all parties in EC it holds that

Theorem 3. *Let l be the number of output bits of C. Then, an active adversary does not get more than l bits of information of the inputs of TS, i.e. the protocol is (t, l)-secure. Moreover, an active adversary that controls at most t parties and does not change the output of an auction does not get any information about the bids of the honest bidders that cannot be deduced by a passive adversary controlling the same parties.*

5 Dynamic t-Private Auctions

We call an EAS dynamic if it allows a bidder to change his bid and if bidders can join a running auction. Introducing dynamism into a EAS may lead to two scenarios: (1) A bidder gets feedback whether his bid changes the result of the auction. Hence, he gets additional information about the bids of the other bidders. (2) A bidder does not get any feedback. Then, dynamism can be reduced to the static case where only the last bid of each bidder is used.

Talking about a dynamic secure EAS we assume that the system works as follows: (1) The system generates an algorithm that allows to evaluate the auction without revealing any information about the bids. Our system will compute an encrypted result. (2) The bidders can deposit their bids in such a way that no party gets any information about the values of the bids (at this moment). (3) If a bidder changes his bid all other bidders can stay inactive. (4) If a bidder is paying some fee, he will get the information whether he is the winner of the auction at the moment. (5) A bidder can withdraw or change his bid such that no party gets any information about the value of his bid (at this moment). Note

that combining the last two requirements results in some leakage (one bit) of information about the bids of the remaining bidders. But, one can assume that the fee makes this bit of information very expensive.

Let us first investigate whether we can recycle a garbled table of a gate for evaluating a circuit on two different inputs. In general, we have to give a negative answer. For a Boolean circuit C and two binary inputs $x, x' \in \{0,1\}^n$ of C with $C(x) = C(x')$ let $\Delta(x, x')$ be the set of input variables with different values in x and x'. Furthermore, for a subset of input variables X let $\Lambda_C(x, X)$ be the set of gates g such that for some x, x' with $C(x) = C(x')$ and $\Delta(x, x') \subseteq X$ the value of g is different on x and x'.

Observation 1. *Let Γ and Γ' be two GCs and X be a subset of input variables that may change their values. For every input pair $x, x' \in \{0,1\}^n$ with $C(x) = C(x')$ and $\Delta(x, x') \subseteq X$ the random strings W_g^0, W_g^1, r_g of all $g \in \Lambda_C(x, X)$ have to be chosen independently in Γ and Γ' to preserve privacy when evaluating Γ on x and Γ' on x'.*

If the random strings W_g^0, W_g^1, r_g are not chosen independently in Γ and in Γ' for all gates $g \in \Lambda_C(x, X)$, then one can observe whether the value of g changes from x to x'. Let B_i be a bidder that has announced to change his bid, i.e. $X = \{x_{i,1}, x_{i,2}, \ldots\}$. Our goal is to find a protocol that allows all bidders $B_j \neq B_i$ to stay inactive. Let Γ be the GCs used on the old bids and let Γ' be the GCs used on the changed bids. From our observation above one can conclude that all random values W_g^0, W_g^1, r_g in Γ and in Γ' with $g \in \bigcup_{x \in \{0,1\}^n} \Lambda_C(x, X)$ have to be chosen independently. Hence, we can run our protocols for all gates $g \in \Lambda_C(X) = \bigcup_{x \in \{0,1\}^n} \Lambda_C(x, X)$ and all wires leaving these gates. Since the values of the gates $g' \notin \Lambda_C(X)$ do not change their values for every input x and x' with $\Delta(x, x') \in X$ we can recycle the random values $W_{g'}^0, W_{g'}^1, r_{g'}$ from Γ in Γ'. Note that for these GCs Γ and Γ' the intermediary strings $W_e^{b_e} \circ (b_e \oplus r_e)$ on the wires that are leaving gates $g' \notin \Lambda_C(X)$ are the same for every pair of input x and x' with $\Delta(x, x') \in X$. Hence, we have only to reevaluate the gates in $\Lambda_C(X)$. This can be done by running a new bidding process for B_i and running a new evaluation process of Γ' by A. We can use a similar protocol to extend a GC from n to $n+1$ bidders.

Theorem 4. *There exists a t-private protocol for a dynamic EAS for $(t+1)^2$ or more parties using $O(dt^3|\Gamma|)$ random bits and a t-private protocol for $2t+1$ or more parties using $O(d(t+1)^{t+3}|\Gamma|)$ random bits where d denotes the number of bid-changes.*

6 Conclusions

In this paper we present a t-private protocol for $2t+1$ parties and a more randomness efficient t-private protocol for $O(t^2)$ parties for sealed bid auctions. Our protocols are based on GCs for evaluating the auction and on strategies for constructing such a circuit in a distributed way. Usually, the bidder-to-auctioneer connection is the bottleneck of an EASs, e.g. the bidder has a limited bandwidth or he is not always available. Thus, we postulate that bidders have to be involved as infrequently as possible. In our system a bidder must only be online to bid and can stay passive during the remaining time. This feature also holds for dynamic auctions where bids are changed or bidders join a running auction. Finally, we analyze the information gain of Byzantine attackers on our EAS and present a strategy for introducing dynamism into an EAS.

References

1. A. Beimel, *On Private Computation in Incomplete Networks*, 12th SIROCCO, pp. 18–33, 2005.
2. M. Ben-Or, S. Goldwasser, A. Wigderson, *Completeness theorems for non-cryptographic fault-tolerant distributed computation*, 20th STOC, pp. 1–10, 1988.
3. M. Bläser, A. Jakoby, M. Liśkiewicz, B. Siebert, *Private Computation – k-Connected versus 1-Connected Networks*, 22nd CRYPTO, pp. 194–209, 2002.
4. F. Brandt, *Secure and Private Auctions without Auctioneers*, Technical Report FKI-245-02, Institut für Informatik, Technische Universität München, 2002.
5. F. Brandt, *Fully Private Auctions in a Constant Number of Rounds*, 7th Annual Conference on Financial Cryptography (FC), pp. 223–238, 2003.
6. C. Cachin, *Efficient Private Bidding and Auctions with an Oblivious Third Party*, 6th ACM Conference on Computer and Communications Security, pp. 120–127, 1999.
7. D. Chaum, C. Crépeau, I. Damgård, *Multiparty unconditionally secure protocols*, 20th STOC, pp. 11–19, 1988.
8. B. Chor, S. Goldwasser, S. Micali, B. Awerbuch, *Verifiable secret sharing and achieving simultaneity in the presence of faults*, 26th FOCS, pp. 383–395, 1985.
9. B. Chor, E. Kushilevitz, *A zero-one law for boolean privacy*, SIAM J. Disc. Math., 4(1):36–47, 1991.
10. K. Chui, R. Zwick, *Auction on the Internet - A Preliminary Study*, Manuscript, Technical report, Hong Kong University of Science and Technology, Department of Marketing, 1999.
11. I. Damgård, Y. Ishai, *Constant-Round Multiparty Computation Using a Black-Box Pseudo-random Generator*, 25th CRYPTO, pp. 378–394, 2005.
12. M. Franklin, M. Reiter, *The Design and Implementation of a Secure Auction Service*, IEEE Transactions on Software Engineering 22(5), pp. 302–312, 1996.
13. M. Franklin, M. Yung, *Secure hypergraphs: Privacy from partial broadcast*, 27th STOC, pp. 36–44, 1995.
14. O. Goldreich, S. Micali, A. Widgerson, *How to play any mental game or a completeness theorem for protocols with honest majority*, 19th STOC, pp. 218–229, 1987.
15. M. Harkavy, H. Kikuchi, J. Tygar, *Electronic Auctions with Private Bids*, 3rd USENIX Workshop on Electronic Commerce, pp. 61–74, 1998.
16. Y. Ishai, E. Kushilevitz, *Randomizing Polynomials: A New Representation with Application to Round-Efficient Secure Computation*, 41st FOCS, pp. 294–304, 2000.
17. Y. Ishai, E. Kushilevitz, *Perfect Constant-Round Secure Computation via Perfect Randomizing Polynomials*, 29th ICALP, pp. 244–256, 2002.
18. M. Jakobsson, A. Juels, *Mix and Match: Secure Function evaluation via Ciphertexts*, 6th ASIACRYPT, pp. 162–177, 2000.
19. A. Juels, M. Szydlo, *A Two-Server, Sealed-Bid Auction Protocol*, 6th Annual Conference on Financial Cryptography (FC), pp. 72–86, 2002.
20. H. Kikuchi, M. Harkavy, J. Tygar, *Multi-round Anonymous Auction Protocols*, 1st IEEE Workshop on Dependable and Real-Time E-Commerce Systems, pp. 62–69, 1998.
21. E. Kushilevitz, S. Micali, R. Ostrovsky, *Reducibility and Completeness In Multi-Party Private Computations*, 35th FOCS, pp. 478–489, 1994.
22. K. Kurosawa, W. Ogata, *Bit-Slice Auction Circuit*, 7th ESORICS, pp. 24–38, 2002.
23. E. Kushilevitz, R. Ostrovsky, A. Rosén, *Characterizing linear size circuits in terms of privacy*, JCSS, 58(1):129–136, 1999.
24. M. Naor, B. Pinkas, R. Sumner, *Privacy Preserving Auctions and Mechanism Design*, 1st ACM Conference on Electronic Commerce, pp. 129–139, 1999.

25. K. Omote, A. Miyaji, *A Second-price Sealed-bid Auction with Verifiable Discriminant of p_0-th Root*, 6th Financial Cryptography Conference (FC), pp. 57–71, 2002.
26. A. Shamir, *How to Share a Secret*, Communic. of the ACM 22(11), pp. 612–613, 1979.
27. A. Sadeghi, M. Schunter, S. Steinbrecher, *Private Auctions with Multiple Rounds and Multiple Items*, 13th IEEE DEXA, pp. 423–427, 2002.
28. P. Stechert, *Dynamic Private Auctions*, Diplomarbeit, Institut für Theoretische Infomatik, Universität zu Lübeck, Germany, January 2005.
29. A. C. Yao, *Protocols for secure computations*, 23rd FOCS, pp. 160–164, 1982.
30. A. C. Yao, *How to generate and exchange secrets*, 27th FOCS, pp. 162–167, 1986.

Secure Multiparty Computations Using a Dial Lock
(Extended Abstract)

Takaaki Mizuki[1], Yoshinori Kugimoto[2], and Hideaki Sone[1]

[1] Information Synergy Center, Tohoku University,
Aramaki-Aza-Aoba 6-3, Aoba-ku, Sendai 980-8578, Japan
tm-paper@rd.isc.tohoku.ac.jp
[2] Sone Lab., Graduate School of Information Sciences, Tohoku University,
Aramaki-Aza-Aoba 6-3, Aoba-ku, Sendai 980-8578, Japan

Abstract. This paper first explores the power of the dial locks (also called the combination locks), which are physical handy devices, in designing cryptographic protocols. Specifically, we design protocols for secure multiparty computations using the dial locks, and give some conditions for a Boolean function to be or not to be securely computable by a dial lock, i.e., to be or not to be "dial-computable." In particular, we exhibit simple necessary and sufficient conditions for a symmetric function to be dial-computable.

1 Introduction

It has been known that some cryptographic tasks can be implemented by several physical handy devices such as envelopes [6,12], cups [6], a deck of cards [1,3,5,7,10,13,17], a PEZ dispenser [2] and scratch-off cards [11,12]. For example, a deck of cards can be used for secure computations [3,5,13,17], zero-knowledge proofs [13,17] and secret key exchange (e.g. [1,7,10]). For another example, Balogh et al. [2] constructed a protocol which securely computes any function using a PEZ dispenser without any use of randomization. Recently, Moran and Naor [11] showed that bit-commitment and coin flipping can be implemented by scratch-off cards, but oblivious transfer cannot be. The research area above is often called *recreational cryptography* [2], *human-centric cryptography* [12] or *cryptography without computers* [13,14,15].

This paper also addresses designing a cryptographic protocol using such a physical handy device. Specifically, we consider the use of *dial locks* (also called *combination locks*) as illustrated in Fig. 1, and give protocols for secure multiparty computations using the dial locks.

1.1 Dial Locks

Any dial lock considered in this paper is like the following. (See again Fig. 1.) A dial lock has k dials, each of which has m numbers for some positive integers

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 499–510, 2007.
© Springer-Verlag Berlin Heidelberg 2007

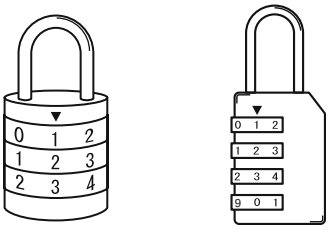


Fig. 1. Examples of dial locks

$k, m \in \mathbb{N}$. Thus, there are exactly m^k combinations. To open the dial lock, one has to set the k dials to the (unique) predetermined combination, which is called the *opening combination*. We assume that one can arbitrarily change the opening combination (provided that she knows the current opening combination). Such an "m-valued k-digit opening-combination-changeable" dial lock is called an (m, k)-*dial lock*. For example, the left dial lock in Fig. 1 is a $(10, 3)$-dial lock, and the right one in Fig. 1 is a $(10, 4)$-dial lock.

1.2 Secure Multiparty Computations

As mentioned before, the goal of this paper is to apply the physical property of the dial locks to achieving secure multiparty computations. (A comprehensive survey of the problems of secure multiparty computations appears in [8].) This paper considers the following simple scenario. Assume that there are n honest-but-curious players $P_1, P_2, \ldots, P_n$, who hold one-bit private inputs $x_1, x_2, \ldots, x_n \in \{0, 1\}$, respectively, where $n \geq 2$. All the n players want to learn the output $f(x_1, x_2, \ldots, x_n)$ of a predetermined (Boolean) function $f : \{0, 1\}^n \to \{0, 1\}$ without revealing more information about their inputs than necessary. (Hereafter, we use simply the term 'function' to refer to a Boolean function.) This paper deals with this simple type of secure multiparty computations (cf. [9]).

Example 1. Assume that three players P_1, P_2 and P_3 holding one-bit private inputs x_1, x_2 and x_3, respectively, want to securely compute the 3-variable AND function $\mathsf{AND}^3(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3$. After a secure multiparty computation (through some protocol), each of the three players learns the value of $x_1 \wedge x_2 \wedge x_3$. Furthermore, the secure computation asks the following for each player, say P_1: when $x_1 = 0$ (and hence $x_1 \wedge x_2 \wedge x_3 = 0$), P_1 must not gain any information about the inputs $x_2, x_3 \in \{0, 1\}$ of the other players; when $x_1 = 1$ and $x_1 \wedge x_2 \wedge x_3 = 0$, P_1 never gains any information other than the fact that $x_2 \wedge x_3 = 0$. Note that when $x_1 \wedge x_2 \wedge x_3 = 1$, all the players get to know the fact that $x_1 = x_2 = x_3 = 1$, of course.

1.3 Our Results

In this paper, we will first design *dial-lock-based cryptographic protocols*: we will give a class of protocols, each of which achieves a secure multiparty computation

(described in Section 1.2) using a dial lock (described in Section 1.1). Although there are numerous functions securely computable by a dial lock such as the AND function and the XOR function (as seen later), every function is not necessarily "dial-computable." For example, the OR function is not dial-computable as seen later. Therefore, we will give a necessary condition for a function to be dial-computable (as in Lemma 1). Furthermore, by restricting the considered functions to symmetric ones, we will obtain simple necessary and sufficient conditions for a symmetric function to be dial-computable (as in Theorem 1). The restriction to the symmetric functions is quite reasonable, because, when a demand for a secure multiparty computation arises, it is a natural setting that all players are "symmetric," i.e., all the players have the same circumstances.

The remainder of the paper is organized as follows. In Section 2, we construct protocols for secure multiparty computations using the dial locks, and provide a mathematical definition of "dial-computability," which abstracts away the concrete dial locks. In Section 3, we give a necessary condition for dial-computable functions. In Section 4, we completely characterize symmetric dial-computable functions. This paper concludes in Section 5 with some discussions and open problems.

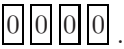
2 Secure Computations Using Dial Locks

In this section, we design a class of protocols, each of which achieves a secure multiparty computation using a dial lock. Before giving the complete description of our protocols, we first present an example in Section 2.1. We then give the description of our protocols in Section 2.2. We finally formalize "dial-computability" in Section 2.3, which abstracts away the concrete dial locks.

2.1 An Example

We now show an example of the execution of our protocols. Consider the case where three players P_1, P_2 and P_3 want to securely compute the 3-variable AND function $\mathsf{AND}^3(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3$ using a $(10, 4)$-dial lock.

The main idea behind our protocols is simple, as follows. Assume that the current opening combination for the $(10, 4)$-dial lock is '0-0-0-0':



Rotate the first dial (namely, the leftmost digit) so that the number on its face decreases by exactly 3 (modulo 10):

Let the three players P_1, P_2 and P_3 take turns, where each player P_i in her turn rotates the first dial (or does not rotate it) depending on her private input $x_i \in \{0, 1\}$ without being seen by any other player: P_i rotates the first dial so that

the number on its face increases by exactly x_i. For instance, if $x_1 = x_2 = x_3 = 1$, i.e., $\mathsf{AND}^3(x_1, x_2, x_3) = 1$, then the the number on the first dial's face varies as

$$\boxed{7} \to 1\left(\boxed{7}\right. \to \boxed{8} \to 1\left(\boxed{8}\right. \to \boxed{9} \to 1\left(\boxed{9}\right. \to \boxed{0}$$
$$x_1 = 1 \qquad\qquad x_2 = 1 \qquad\qquad x_3 = 1$$

and hence the dial lock is opened. On the other hand, if $\mathsf{AND}^3(x_1, x_2, x_3) = 0$, say $x_1 = x_2 = 1$ and $x_3 = 0$, then the the number on the first dial's face varies as

$$\boxed{7} \to 1\left(\boxed{7}\right. \to \boxed{8} \to 1\left(\boxed{8}\right. \to \boxed{9} \to 0\left(\boxed{9}\right. \to \boxed{9}$$
$$x_1 = 1 \qquad\qquad x_2 = 1 \qquad\qquad x_3 = 0$$

and hence the dial lock is not opened (note that P_3 never rotates the dial in this case). Thus, $\mathsf{AND}^3(x_1, x_2, x_3) = 1$ if and only if the dial lock is opened. Therefore, the function AND^3 can be computed by the $(10, 4)$-dial lock. However, this method is not secure; one of the simplest reasons is that player P_2 gets to know the input x_1 of player P_1 just by looking at the number on the first dial's face in her turn.

Slightly modifying the method above, we can obtain the following secure protocol, where N denotes the current number on the first dial's face during the execution of the protocol, and each player is assumed to operate the dial lock without being seen by any other player.

1. Player P_1 holds the $(10, 4)$-dial lock (whose opening combination is public). She randomly changes the opening combination, and sets the four dials to it. (Hence, we now have $N = r$ for a random number $r \in \mathbb{Z}_{10} = \{0, 1, \ldots, 9\}$, which is private only to P_1.)
2. Player P_1 rotates the first dial so that the number on its face decreases by exactly 3. (We now have $N = r - 3 \bmod 10$.)
3. Player P_1 rotates the first dial so that the number on its face increases by exactly x_1. (We now have $N = r - 3 + x_1 \bmod 10$.) She then hands the dial lock to player P_2.
4. Player P_2 rotates the first dial so that the number on its face increases by exactly x_2. (We now have $N = r - 3 + x_1 + x_2 \bmod 10$.) She then hands the dial lock to player P_3.
5. Player P_3 rotates the first dial so that the number on its face increases by exactly x_3. (We now have $N = r - 3 + x_1 + x_2 + x_3 \bmod 10$.)
6. Player P_3 announces whether the dial lock is opened or not. (The former case implies $\mathsf{AND}^3(x_1, x_2, x_3) = 1$, and the latter case implies $\mathsf{AND}^3(x_1, x_2, x_3) = 0$.)
7. Player P_3 randomly rotates the first dial, and then hands the dial lock back to player P_1.
8. Player P_1 sets the opening combination to the initial one (so that the current opening combination becomes public).

Note that the dial lock is opened in step 6 if and only if

$$r - 3 + x_1 + x_2 + x_3 \equiv r \pmod{10},$$

which is equivalent to $x_1 \wedge x_2 \wedge x_3 = 1$. Furthermore, since r is random and is known only to player P_1, each of players P_2 and P_3 cannot gain any information about the input of any other player when she looks at the number on the first dial's face in her turn. In addition, since player P_3 randomly rotates the first dial in step 7, the dial lock handed back to player P_1 gives player P_1 no information about the input of any other player. Thus, this protocol securely computes the function $\mathsf{AND}^3(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3$ using a $(10, 4)$-dial lock. (Remember that all the players have been assumed to be honest-but-curious.)

The protocol above can be represented by the following four vectors:

$$\boldsymbol{c}_1 = (1, 0, 0, 0),\ \boldsymbol{c}_2 = (1, 0, 0, 0),\ \boldsymbol{c}_3 = (1, 0, 0, 0) \text{ and } \boldsymbol{d} = (3, 0, 0, 0),$$

where $\boldsymbol{c}_i = (c_i^1, c_i^2, c_i^3, c_i^4)$, $1 \le i \le 3$, means that player P_i rotates the j-th dial for each j, $1 \le j \le 4$, so that the number on its face increases by exactly $c_i^j x_i$, and $\boldsymbol{d} = (d^1, d^2, d^3, d^4)$ means that player P_1 rotates the j-th dial for each j, $1 \le j \le 4$, in step 2 so that the number on its face decreases by exactly d^j. (Remember that the protocol above uses only the first dial, and hence $c_i^2 = c_i^3 = c_i^4 = d^2 = d^3 = d^4 = 0$ for every i, $1 \le i \le 3$.) Note that, for the four vectors above,

$$\mathsf{AND}^3(x_1, x_2, x_3) = 1 \iff \boldsymbol{c}_1 x_1 + \boldsymbol{c}_2 x_2 + \boldsymbol{c}_3 x_3 \equiv \boldsymbol{d} \pmod{10}. \tag{1}$$

Considering various sequences of four vectors $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$ and $\boldsymbol{d}$, one can obtain a class of protocols, each of which securely computes a certain (corresponding) function using a $(10, 4)$-dial lock. For example, one can easily observe that the following sequence of four vectors also achieves a secure multiparty computation of AND^3:

$$\boldsymbol{c}_1 = (1, 0, 0, 0),\ \boldsymbol{c}_2 = (0, 1, 0, 0),\ \boldsymbol{c}_3 = (0, 0, 1, 0) \text{ and } \boldsymbol{d} = (1, 1, 1, 0);$$

note that such a protocol uses the first three dials, and that this sequence of the four vectors also satisfies Eq. (1). For another example, one can easily notice that the 3-variable XOR function $\mathsf{XOR}^3(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ can be securely computed by the following sequence of four vectors:

$$\boldsymbol{c}_1 = (5, 0, 0, 0),\ \boldsymbol{c}_2 = (5, 0, 0, 0),\ \boldsymbol{c}_3 = (5, 0, 0, 0) \text{ and } \boldsymbol{d} = (5, 0, 0, 0),$$

because

$$x_1 \oplus x_2 \oplus x_3 = 1 \iff 5x_1 + 5x_2 + 5x_3 \equiv 5 \pmod{10}.$$

It should be noted that the $(10, 4)$-dial lock might be "automatically" and "unnecessarily" opened during the execution of the "XOR protocol" above depending on the features of the dial lock used in the protocol. For example, in the case of $x_1 = x_2 = 1$, the dial lock might be unnecessarily opened immediately after player P_2 rotates the first dial. When one owns a dial lock like this, in order to avoid unnecessarily opening the dial lock, one can utilize an unused dial, say the fourth dial, as a "stopper" by predeterminedly rotating such an unused dial.

2.2 Our Protocols

We are now ready to generalize a class of our dial-lock-based protocols.

Assume that there are n players and an (m, k)-dial lock whose opening combination is public. Then, a sequence of $n+1$ vectors $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n, \boldsymbol{d} \in (\mathbb{Z}_m)^k = \{0, 1, \ldots, m-1\}^k$ determines a unique protocol as follows, where

$$\begin{aligned} \boldsymbol{c}_1 &= (c_1^1, c_1^2, \ldots, c_1^k) \\ \boldsymbol{c}_2 &= (c_2^1, c_2^2, \ldots, c_2^k) \\ &\vdots \\ \boldsymbol{c}_n &= (c_n^1, c_n^2, \ldots, c_n^k) \\ \boldsymbol{d} &= (d^1, d^2, \ldots, d^k), \end{aligned}$$

and each player is assumed to operate the dial lock without being seen by any other player.

1. Player P_1 randomly changes the opening combination of the dial lock, and sets the k dials to it.
2. For each j, $1 \leq j \leq k$, player P_1 rotates the j-th dial so that the number on its face decreases by exactly d^j.
3. Set $i := 1$.
4. For each j, $1 \leq j \leq k$, player P_i rotates the j-th dial so that the number on its face increases by exactly $c_i^j x_i$.
5. Set $i := i + 1$. If $i \leq n$, then return to step 4.
6. Player P_n announces whether the dial lock is opened or not.
7. Player P_n randomly rotates all the dials, and then hands the dial lock back to player P_1.
8. Player P_1 sets the opening combination to the initial one.

Thus, the protocol above securely computes an n-variable function f such that

$$f(x_1, x_2, \ldots, x_n) = 1 \iff \text{the dial lock is opened in step 6.}$$

In other words, it securely computes an n-variable function f such that

$$f(x_1, x_2, \ldots, x_n) = 1 \iff \boldsymbol{c}_1 x_1 + \boldsymbol{c}_2 x_2 + \cdots + \boldsymbol{c}_n x_n \equiv \boldsymbol{d} \pmod{m}.$$

Therefore, given an (m, k)-dial lock, a sequence of $n + 1$ vectors

$$\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n, \boldsymbol{d} \in (\mathbb{Z}_m)^k$$

determines a unique function securely computable by the corresponding protocol. Hence, we can abstract away the concrete dial locks as in the next subsection.

2.3 Abstraction

From the discussion above, we immediately have the following Definition 1.

Definition 1. *We say that an* (m, k)*-dial lock* securely computes *an* n*-variable function* f *if there exist* $n+1$ *vectors* $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n, \boldsymbol{d} \in (\mathbb{Z}_m)^k$ *such that*

$$f(x_1, x_2, \ldots, x_n) = 1 \iff \boldsymbol{c}_1 x_1 + \boldsymbol{c}_2 x_2 + \cdots + \boldsymbol{c}_n x_n \equiv \boldsymbol{d} \pmod{m}.$$

For example, as seen in Section 2.1, a $(10, 4)$-dial lock securely computes the functions AND^3 and XOR^3. More generally, one can easily notice that an (m, k)-dial lock securely computes an n-variable AND function AND^n with $n \leq (m-1)^k$, and that a $(2, 1)$-dial lock securely computes any n-variable XOR (parity) function XOR^n.

On the other hand, unfortunately, not every function can be securely computed by a dial lock. For example, as shown later, there is no (m, k)-dial lock which securely computes the OR function

$$\mathsf{OR}^n(x_1, x_2, \ldots, x_n) = x_1 \vee x_2 \vee \cdots \vee x_n.$$

Therefore, in the next two sections, we will investigate conditions for a function to be or not to be "dial-computable," the formal definition of which is below.

Definition 2. *We say that a function* f *is* dial-computable *if there exists an* (m, k)*-dial lock which securely computes* f*.*

3 A Necessary Condition for Dial-Computable Functions

In this section, we give a necessary condition for a function to be dial-computable.

We first describe some notations. Given a function f, any vector $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = 1$ is called a *true vector* of f. For a function f, define the set $T(f)$ of all the true vectors as

$$T(f) \stackrel{\text{def}}{=} \{\boldsymbol{x} \mid f(\boldsymbol{x}) = 1\}.$$

As mentioned before, unfortunately, every function is not necessarily dial-computable. For example, the function OR^n (with $n \geq 2$) is not dial-computable, as follows. Suppose for a contradiction that OR^n is dial-computable. Then, by Definitions 1 and 2, there exist $k, m \in \mathbb{N}$ and $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n, \boldsymbol{d} \in (\mathbb{Z}_m)^k$ such that

$$\mathsf{OR}^n(x_1, x_2, \ldots, x_n) = 1 \iff \boldsymbol{c}_1 x_1 + \boldsymbol{c}_2 x_2 + \cdots + \boldsymbol{c}_n x_n \equiv \boldsymbol{d} \pmod{m}.$$

Now, consider the following three true vectors $\boldsymbol{x}_1^+, \boldsymbol{x}_2^+, \boldsymbol{x}_1^- \in T(\mathsf{OR}^n)$:

$$\boldsymbol{x}_1^+ = (1, 0, 0, \ldots, 0),\ \boldsymbol{x}_2^+ = (0, 1, 1, \ldots, 1) \text{ and } \boldsymbol{x}_1^- = (1, 1, \ldots, 1).$$

Since $\mathsf{OR}^n(\boldsymbol{x}_1^+) = 1$, we have

$$\boldsymbol{c}_1 \equiv \boldsymbol{d} \pmod{m}.$$

Similarly, since $\mathsf{OR}^n(\boldsymbol{x}_2^+) = \mathsf{OR}^n(\boldsymbol{x}_1^-) = 1$, we have

$$\boldsymbol{c}_2 + \boldsymbol{c}_3 + \cdots + \boldsymbol{c}_n \equiv \boldsymbol{d} \pmod{m}$$

and

$$\boldsymbol{c}_1 + \boldsymbol{c}_2 + \cdots + \boldsymbol{c}_n \equiv \boldsymbol{d} \pmod{m}.$$

Therefore,

$$\begin{array}{rr} & \boldsymbol{c}_1 \equiv \boldsymbol{d} \pmod{m} \\ +) & \boldsymbol{c}_2 + \boldsymbol{c}_3 + \cdots + \boldsymbol{c}_n \equiv \boldsymbol{d} \pmod{m} \\ -) & \boldsymbol{c}_1 + \boldsymbol{c}_2 + \boldsymbol{c}_3 + \cdots + \boldsymbol{c}_n \equiv \boldsymbol{d} \pmod{m} \\ \hline & 0 \equiv \boldsymbol{d} \pmod{m} \end{array}$$

and hence the last equation implies $\mathsf{OR}^n(0,0,\ldots,0) = 1$, a contradiction. Thus, the function OR^n is not dial-computable. Note that the three true vectors $\boldsymbol{x}_1^+$, $\boldsymbol{x}_2^+$ and $\boldsymbol{x}_1^-$ satisfy

$$(\boldsymbol{x}_1^+ + \boldsymbol{x}_2^+) - \boldsymbol{x}_1^- = (0,0,\ldots,0).$$

As in this example, given an n-variable dial-computable function f, if we can find two sets $\{\boldsymbol{x}_1^+, \boldsymbol{x}_2^+\}$ and $\{\boldsymbol{x}_1^-\}$ of true vectors of f such that $(\boldsymbol{x}_1^+ + \boldsymbol{x}_2^+) - \boldsymbol{x}_1^-$ is a binary vector, namely it is in $\{0,1\}^n$, then Definitions 1 and 2 imply that $(\boldsymbol{x}_1^+ + \boldsymbol{x}_2^+) - \boldsymbol{x}_1^-$ must be also a true vector of f, i.e., it must be also in $T(f)$. More generally, given a dial-computable function f, if we find a pair

$$\left(\{\boldsymbol{x}_1^+, \boldsymbol{x}_2^+, \ldots, \boldsymbol{x}_\ell^+, \boldsymbol{x}_{\ell+1}^+\}, \{\boldsymbol{x}_1^-, \boldsymbol{x}_2^-, \ldots, \boldsymbol{x}_\ell^-\}\right) \in 2^{T(f)} \times 2^{T(f)}$$

such that

$$(\boldsymbol{x}_1^+ + \boldsymbol{x}_2^+ + \cdots + \boldsymbol{x}_\ell^+ + \boldsymbol{x}_{\ell+1}^+) - (\boldsymbol{x}_1^- + \boldsymbol{x}_2^- + \cdots + \boldsymbol{x}_\ell^-)$$

is a binary vector, then the binary vector must be in $T(f)$, as will be shown in Lemma 1.

Before presenting Lemma 1, we formally define some terms as in the following Definitions 3 and 4.

Definition 3. *Let f be a function. We say that a pair $(X^+, X^-) \in 2^{T(f)} \times 2^{T(f)}$ such that $X^+ \cap X^- = \emptyset$ and $|X^+| = |X^-| + 1 \geq 2$ is a* successive-size-pair *of f if $\sum_{\boldsymbol{x} \in X^+} \boldsymbol{x} - \sum_{\boldsymbol{x} \in X^-} \boldsymbol{x}$ is a binary vector.*

Definition 4. *A successive-size-pair (X^+, X^-) of a function f is said to be* false *if $\sum_{\boldsymbol{x} \in X^+} \boldsymbol{x} - \sum_{\boldsymbol{x} \in X^-} \boldsymbol{x}$ is not a true vector of f, i.e., it is not in $T(f)$.*

We now have the following Lemma 1, which gives a necessary condition for a function to be dial-computable.

Lemma 1. *If a function f is dial-computable, then f has no false successive-size-pair.*

Proof. Let f be a dial-computable function. Suppose for a contradiction that f has a false successive-size-pair (X^+, X^-). Since f is dial-computable, Definitions 1 and 2 imply that there exist $k, m \in \mathbb{N}$ and $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n, \boldsymbol{d} \in (\mathbb{Z}_m)^k$ such that

$$\boldsymbol{x} \in T(f) \iff (\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n)\, {}^t\boldsymbol{x} \equiv \boldsymbol{d} \pmod{m}$$

where ${}^t\boldsymbol{x}$ is the transposed vector of $\boldsymbol{x}$. Since $X^+ \cup X^- \subseteq T(f)$ and $|X^+| = |X^-| + 1$, we have

$$(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n)\, {}^t\Big(\sum_{\boldsymbol{x}\in X^+} \boldsymbol{x} - \sum_{\boldsymbol{x}\in X^-} \boldsymbol{x}\Big) \equiv \boldsymbol{d} \pmod{m},$$

contradicting the assumption that the successive-size-pair (X^+, X^-) is false. □

One can use Lemma 1 to show the non-dial-computability of some function. For example, consider the following 9-variable function g:

$$g(x_1, x_2, \ldots, x_9) = x_1\overline{x}_2\overline{x}_3\overline{x}_4\overline{x}_5\overline{x}_6\overline{x}_7x_8x_9 \vee \overline{x}_1\overline{x}_2x_3x_4x_5\overline{x}_6\overline{x}_7\overline{x}_8\overline{x}_9$$
$$\vee\, \overline{x}_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6x_7\overline{x}_8\overline{x}_9 \vee \overline{x}_1\overline{x}_2\overline{x}_3x_4\overline{x}_5x_6\overline{x}_7x_8\overline{x}_9 \vee \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4x_5\overline{x}_6x_7\overline{x}_8x_9,$$

where the conjunction symbol $\wedge$ is omitted. Let

$$X^+ = \{(1,0,0,0,0,0,0,1,1), (0,0,1,1,1,0,0,0,0), (0,1,0,0,0,1,1,0,0)\},$$

and let

$$X^- = \{(0,0,0,1,0,1,0,1,0), (0,0,0,0,1,0,1,0,1)\}.$$

Then, (X^+, X^-) is a false successive-size-pair of g, because

$$\sum_{\boldsymbol{x}\in X^+} \boldsymbol{x} - \sum_{\boldsymbol{x}\in X^-} \boldsymbol{x} = (1,1,1,0,0,0,0,0,0)$$

is not in $T(g)$. Therefore, by Lemma 1, the function g is not dial-computable.

Although we have been unable to extend Lemma 1 to give a necessary and sufficient condition for dial-computable functions, we are able to show that, when we address only symmetric functions, the necessary condition in Lemma 1 becomes a sufficient one, as will be seen in Theorem 1.

4 Characterizing Symmetric Dial-Computable Functions

In this section, we restrict our attention to symmetric functions, and then completely characterize symmetric dial-computable functions. As mentioned before, when a demand for a secure multiparty computation arises, it is natural to assume that all players are "symmetric," i.e., all the players have the same circumstances.

We first review some terms. We say that an n-variable function f is *symmetric* if it is unchanged by any permutation of its variables, that is,

$$f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n)$$

for any variables x_i and x_j. It is well-known (e.g., refer to [16]) that any n-variable symmetric function $f(x_1, x_2, \ldots, x_n)$ can be represented by using the

$n+1$ *elementary symmetric functions* $S_0^n, S_1^n, \ldots, S_n^n$, that is, for any n-variable symmetric function f, there exists a set $A \subseteq \{0, 1, \ldots, n\}$ such that

$$f = \bigvee_{i \in A} S_i^n,$$

where

$$S_\ell^n(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i = \ell; \\ 0 & \text{otherwise} \end{cases}$$

for each ℓ, $0 \leq \ell \leq n$.

As will be seen below in Theorem 1, all the n-variable symmetric dial-computable functions can be characterized by the set $\mathcal{MOD}^n$ of the following "modular" functions. For integers s and p with $0 \leq s < p$, define

$$\mathsf{MOD}_{s,p}^n(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i \equiv s \pmod{p}; \\ 0 & \text{otherwise.} \end{cases}$$

We then define

$$\mathcal{MOD}^n \stackrel{\text{def}}{=} \{\mathsf{MOD}_{s,p}^n \mid 0 \leq s < p\}.$$

It should be noted that the functions $\mathsf{MOD}_{s,p}^n$ often appear in (logical or circuit) complexity theory (e.g. [4,18]).

We are now ready to present Theorem 1, which exhibits exact characterizations of symmetric dial-computable functions.

Theorem 1. *Let f be an n-variable symmetric function. Then, the following three assertions (a), (b) and (c) are equivalent:*

(a) f is dial-computable;
(b) f has no false successive-size-pair;
(c) $f \in \mathcal{MOD}^n$.

Note that, by Lemma 1, the condition (a) in Theorem 1 immediately implies the condition (b). Furthermore, '(c) implies (a)' is easily verified as follows: let $f = \mathsf{MOD}_{s,p}^n$ for some s and p, then a $(p,1)$-dial lock securely computes f by setting $c_1^1 = c_2^1 = \cdots = c_n^1 = 1$ and $d^1 = s$. Therefore, in order to complete the proof of Theorem 1, it suffices to show '(b) implies (c).'

Before proving '(b) implies (c),' we have the following Lemma 2.

Lemma 2. *Let $0 \leq a - \delta < a < a + \delta \leq n$, and let f be an n-variable symmetric function having no false successive-size-pair such that f contains S_a^n. Then, the following (a) and (b) hold.*

(a) If f contains $S_{a+\delta}^n$, then f contains $S_{a-\delta}^n$.
(b) If f contains $S_{a-\delta}^n$, then f contains $S_{a+\delta}^n$.

Proof. omitted in this extended abstract. □

Using Lemma 2, we can prove the following Lemma 3, which mentions that the condition (b) in Theorem 1 implies the condition (c).

Lemma 3. *If an n-variable symmetric function f has no false successive-size-pair, then $f \in \mathcal{MOD}^n$.*

Proof. omitted in this extended abstract. □

Thus, Lemma 3 has completed the proof of Theorem 1.

5 Conclusions

In this paper, we first proposed dial-lock-based cryptographic protocols: we designed a class of protocols, each of which achieves a secure multiparty computation using a dial lock, and hence we showed that the physical property of the dial locks can be applied to designing cryptographic protocols. We then gave a necessary condition for a function to be dial-computable. Finally, we obtained simple necessary and sufficient conditions for a symmetric function to be dial-computable.

A natural open problem is to find a simple necessary and sufficient condition for any (not necessarily symmetric) function to be dial-computable. Furthermore, characterizing "(m, k)-dial-computable" functions for some fixed positive integers m and k is a further task. For example, it is an interesting open problem to find the smallest m and/or k such that all dial-computable functions are also (m, k)-dial-computable.

This paper is the first attempt at constructing dial-lock-based cryptographic protocols, and our construction is somewhat straightforward. Therefore, one might be able to extend the class of protocols so that the set of "modified-dial-computable" functions would be interestingly larger. For a trivial example, if we slightly change the class of our protocols so that all the players recognize $f = 0$ (instead of $f = 1$) in the case where the dial lock is opened, then the negation $\bar{f}$ of a dial-computable function f becomes "modified dial-computable."

Acknowledgements

We thank the anonymous referees whose comments and suggestions helped us to improve the presentation of the paper. This research was partially supported by the Grant-in-Aid for Exploratory Research No. 17650002 from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. M. H. Albert, R. E. L. Aldred, M. D. Atkinson, H. P. van Ditmarsch, and C. C. Handley, "Safe communication for card players by combinatorial designs for two-step protocols," Australasian Journal of Combinatorics, vol. 33, pp. 33–46, 2005.
2. J. Balogh, J. A. Csirik, Y. Ishai, and E. Kushilevitz, "Private computation using a PEZ dispenser," Theoretical Computer Science, vol. 306, pp. 69–84, 2003.

3. B. den Boer, "More efficient match-making and satisfiability: the five card trick," Proc. EUROCRYPT '89, Lecture Notes in Computer Science, vol. 434, pp. 208–217, Springer-Verlag, 1990.
4. P. Clote and E. Kranakis, "Boolean Functions and Computation Models," Springer-Verlag, Berlin, Heidelberg, New York, 2002.
5. C. Crépeau and J. Kilian, "Discreet solitary games," Proc. CRYPTO '93, Lecture Notes in Computer Science, vol. 773, pp. 319–330, Springer-Verlag, 1994.
6. R. Fagin, M. Naor, and P. Winkler, "Comparing information without leaking it," Communications of the ACM, vol. 39, no. 5, pp. 77–85, 1996.
7. M. J. Fischer and R. N. Wright, "Bounds on secret key exchange using a random deal of cards," Journal of Cryptology, vol. 9, pp. 71–99, 1996.
8. O. Goldreich, "Foundations of Cryptography II: Basic Applications," Cambridge University Press, Cambridge, 2004.
9. T. Mizuki, T. Otagiri, and H. Sone, "Secure computations in a minimal model using multiple-valued ESOP expressions," Proc. TAMC 2006, Lecture Notes in Computer Science, vol. 3959, pp. 547–554, Springer-Verlag, 2006.
10. T. Mizuki, H. Shizuya, and T. Nishizeki, "Characterization of optimal key set protocols," Discrete Applied Mathematics, vol. 131, no. 1, pp. 213–236, 2003.
11. T. Moran and M. Naor, "Basing cryptographic protocols on tamper-evident seals," Proc. ICALP 2005, Lecture Notes in Computer Science, vol. 3580, pp. 285–297, Springer-Verlag, 2005.
12. T. Moran and M. Naor, "Polling with physical envelopes: a rigorous analysis of a human-centric protocol," Proc. EUROCRYPT 2006, Lecture Notes in Computer Science, vol. 4004, pp. 88–108, Springer-Verlag, 2006.
13. V. Niemi and A. Renvall, "Secure multiparty computations without computers," Theoretical Computer Science, vol. 191, pp. 173–183, 1998.
14. A. Salomaa, "Caesar and DNA. Views on cryptology," Proc. the 12th International Symposium on Fundamentals of Computation Theory (FCT '99), Lecture Notes in Computer Science, vol. 1684, pp. 39–53, Springer-Verlag, 1999.
15. A. Salomaa, "Public-Key Cryptography (Second, Enlarged Edition)," Springer-Verlag, Berlin, Heidelberg, New York, 1996.
16. T. Sasao, "Switching Theory for Logic Synthesis," Kluwer Academic Publishers, Boston, MA, 1999.
17. A. Stiglic, "Computations with a deck of cards," Theoretical Computer Science, vol. 259, pp. 671–678, 2001.
18. H. Vollmer, "Introduction to Circuit Complexity," Springer-Verlag, Berlin, Heidelberg, New York, 1999.

A Time Hierarchy Theorem for Nondeterministic Cellular Automata

Chuzo Iwamoto*, Harumasa Yoneda, Kenichi Morita, and Katsunobu Imai

Hiroshima University, Graduate School of Engineering
Higashi-Hiroshima, 739-8527 Japan
chuzo@hiroshima-u.ac.jp

Abstract. We present a tight time-hierarchy theorem for nondeterministic cellular automata by using a recursive padding argument. It is shown that, if $t_2(n)$ is a time-constructible function and $t_2(n)$ grows faster than $t_1(n+1)$, then there exists a language which can be accepted by a $t_2(n)$-time nondeterministic cellular automaton but not by any $t_1(n)$-time nondeterministic cellular automaton.

1 Introduction

One of the basic problems in complexity theory is to find the slightest enlarging of the complexity bound which allows new languages to be accepted. There is a huge amount of literature on time hierarchy theorems for various models of computation, such as Turing machines (TMs) [3,4,6,8,14,16], random access machines (RAMs) [2], parallel RAMs [7,13], and uniform circuit families [10].

In this paper, we investigate time-hierarchies of cellular automata (CA). The first result on CA-based hierarchies was given in [11]; it was shown that there is a language which can be accepted by a one-dimensional deterministic CA (1-DCA) in $t_2(n)$ time but not by any 1-DCA in $t_1(n)$ time. Here, $t_1(n)$ and $t_2(n)$ are arbitrary time-constructible functions such that $t_2(n)$ is not bounded by $O(t_1(n))$. (Time constructible functions on 1-DCA were also discussed in [11].)

Another result on CA-hierarchies is in the hyperbolic space [9]; it was shown that there is a language which can be accepted by two-dimensional hyperbolic CA (2-HCA) in $(t_2(n))^3$ time but not by any 2-HCA in $t_1(n)$ time. When $t_1(n) = n^r$, this hierarchy result can be improved as follows: For any rational constants $r \geq 1$ and $\epsilon > 0$, there is a language which can be accepted by an $n^{r+\epsilon}$-time 2-HCA but not by any n^r-time 2-HCA [12]. Interestingly, these time-hierarchy results in the hyperbolic space hold for both deterministic and nondeterministic cases.

On the other hand, no attempt has been made to present time-hierarchy results on one-dimensional nondeterministic CA (1-NCA). In this paper, it is shown that, if $t_2(n)$ is a time-constructible function and $t_2(n)$ grows faster than $t_1(n+1)$, then there exists a language which can be accepted by a $t_2(n)$-time 1-NCA but not by any $t_1(n)$-time 1-NCA.

* Corresponding author. This research was supported in part by Scientific Research Grant, Ministry of Japan.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 511–520, 2007.
© Springer-Verlag Berlin Heidelberg 2007

A lot of techniques have been known for separating complexity classes. For example, (i) the crossing-sequence argument for one-tape TMs [5], (ii) diagonal arguments for deterministic TMs [3,4,8], 1-DCA [11], PRAMs and DLOGTIME-uniform circuits [7], and (non)deterministic HCA [9], and (iii) padding arguments for nondeterministic TMs [6], alternating TMs and PRAMs [13], and (non)deterministic HCA [12]. In this paper, we will show a hierarchy theorem for one-dimensional NCA by using a recursive padding argument, which was firstly used in [1,19,20] for multi-tape nondeterministic TMs.

In Section 2, we give the definition of 1-NCA. The main result is also given in that section. The proof is given in Section 3.

2 Nondeterministic Cellular Automata

A cellular automaton (CA) is a synchronous parallel string acceptor, consisting of a semi-infinite one-dimensional array of identical finite-state automata, called *cells*, which are uniformly interconnected (see Fig. 1). Every cell operates synchronously at discrete time steps and changes its state depending on the previous states of itself and its neighbours.

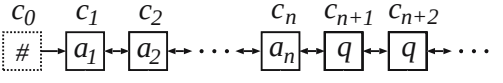


Fig. 1. Cellular automaton

A nondeterministic CA is a 7-tuple $M = (Q, \Sigma, \#, \delta, q, Q_A, Q_R)$, where

1. Q is the finite nonempty set of states,
2. $\Sigma \subseteq Q$ is the finite input alphabet,
3. $\#$ is the special boundary symbol not in Q,
4. $\delta : (Q \cup \{\#\}) \times Q \times Q \to 2^Q$ is the local transition function,
5. $q \in Q$ is the quiescent state such that $\delta(q, q, q) = \{q\}$, and if $\delta(p_1, p_2, p_3) \ni q$ then $(p_1, p_2, p_3) \in Q \times \{q\} \times \{q\}$ (i.e., any non-quiescent state does not become the quiescent state),
6. $Q_A \subseteq Q$ is the set of accepting states such that if $(p_1, p_2, p_3) \in \{\#\} \times Q_A \times Q$ then $\delta(p_1, p_2, p_3) \subseteq Q_A$,
7. $Q_R \subseteq Q$ is the set of rejecting states such that if $(p_1, p_2, p_3) \in \{\#\} \times Q_R \times Q$ then $\delta(p_1, p_2, p_3) \subseteq Q_R$.

The cell assigned to the integer $i \geq 1$ is denoted by c_i. The input string $a_1 a_2 \cdots a_n$ is applied to the array in parallel at step 0 by setting the states of cells $c_1, c_2, \ldots, c_n$ to $a_1, a_2, \ldots, a_n$, respectively. The remaining cells $c_{n+1}, c_{n+2}, \ldots$ are in the quiescent state. (Our CA is the so-called parallel-input model; however, the same hierarchy result holds for the sequential-input model, in which the input is fed serially to the leftmost cell.)

A configuration of a CA is represented by a string in $\{\#\}(Q-\{q\})^*$. A particular configuration is said to be *accepting* (resp. *rejecting*) if the first cell c_1 is in an accepting (resp. rejecting) state. The definition of a computation tree is mostly from [17]. For an input, computations of a nondeterministic CA are described as a tree T: All nodes are configurations, the root is the initial configuration of the CA for the given input, and the children of a configuration C are exactly those configurations reachable from C in one step allowed by the transition function. Leaves of T are final configurations, which may be accepting or rejecting. Certain paths in T may be infinite.

An interior node is defined to be accepting if at least one of its children is accepting. The CA accepts the input iff the root is accepting. A nondeterministic CA M is defined to be $t(n)$-*time bounded* if, for every accepted input w of length n, the computation tree T of M started with w stays accepting if it is pruned at depth $t(n)$. A CA is said to be *deterministic* if its transition function satisfies $|\delta(p_1,p_2,p_3)|=1$ for every $(p_1,p_2,p_3)\in(Q\cup\{\#\})\times Q\times Q$.

Let $M_1,M_2,\ldots,M_r$ be CA with state sets $Q_1,Q_2,\ldots,Q_r$, respectively. Consider a CA M such that the state set Q is a Cartesian product $Q=Q_1\times Q_2\times\cdots\times Q_r$ and each cell is partitioned into r *sub-cells*. The array of specific sub-cells in all cells is called a *track*. Then M can simulate $M_1,M_2,\ldots,M_r$ in r tracks simultaneously.

The language accepted by a CA M is denoted by $L(M)$. A function $t(n)$ is said to be *time-constructible* if, for each n, there is a deterministic CA such that the first cell c_1 enters an accepting state at step $t(n)$ for the first time on all inputs of length n. It is known that if a function $t(n)$ is computable by an $O(t(n)-n)$-time TM (to which value n is given as a binary string of length $\lfloor\log_2 n\rfloor+1$), then $t(n)$ is also time-constructible by a CA [11]. If two functions are time-constructible by CA, then the sum, product, and exponential functions of them are also time-constructible by CA. If $t(n)$ is time-constructible, then $t(n)$ is also *space-constructible*, i.e., there is a one-dimensional deterministic CA which places a "marker" at the $t(n)$th cell in $O(t(n))$ time (by emitting 1/2-speed and unit-speed pulses at steps 0 and $t(n)$, respectively).

Now we are ready to present our main theorem.

Theorem 1. *Suppose that $t_2(n)$ is an arbitrary time-constructible function and $\lim_{n\to\infty}\frac{t_1(n+1)}{t_2(n)}=0$. There exists a language which can be accepted by a $t_2(n)$-time nondeterministic CA but not by any $t_1(n)$-time nondeterministic CA.*

Since $t_1(n+1)=O(t_1(n))$ when $t_1(n)$ is a polynomial in n, we can say, for example, n^r-time nondeterministic CA are stronger than $(n^r/\log\log n)$-time nondeterministic CA, where $r\geq 1$ is an arbitrary rational constant. However, it is not known whether $(t(n))^2$-time nondeterministic CA are stronger than $t(n)$-time nondeterministic CA when $t(n)=2^{2^n}$.

3 Time Hierarchy for Nondeterministic CA

In this section, we will prove Theorem 1. In the rest of this paper, all CA are nondeterministic CA unless stated otherwise. We use the recursive padding

method [20]. The outline of the proof is as follows: We first define a universal CA in Section 3.1. In Section 3.2, we construct a recursively padding CA which lengthens the input string as many times as we want. In Section 3.3, we observe the relationship between our recursively padding CA and its language. In Section 3.4, we assume for contradiction that any $t_2(n)$-time computation is sped-up to $t_1(n)$ time. Under this assumption, we will show that any recursive language can be accepted by $t_1(n)$-time CA (i.e., any computation for recursive languages (with no time restriction) can be sped-up to $t_1(n)$ time), a contradiction.

3.1 Universal CA

All languages in this section are over $\{0,1\}$. We first define the encoding rule of CA, which is essentially from [11]. We denote the states of CA by $q_1, q_2, \ldots, q_m$, where q_1 is the special boundary state #, and q_2 is the quiescent state q. For simplicity, we assume that q_3 (resp. q_4) is the unique accepting (resp. rejecting) state. State q_i is encoded into string 10^i of length $i+1$. We encode each transition rule $\delta(q_i, q_j, q_k) = \{q_{l_1}, q_{l_2}, \ldots, q_{l_r}\}$ into string $1110^i10^j10^k10^{l_1}10^{l_2}\cdots10^{l_r}$ for every $(q_i, q_j, q_k) \in (Q \cup \{\#\}) \times Q \times Q$. The encoding sequence e of a CA is the concatenation of all transition rules in the lexicographical order, called the *encoding part*, followed by $1111\cdots10$ of length $2^l - l$, called the *padding part*, where l is the length of the encoding part. Let M_e denote the CA whose encoding sequence is e.

Let $L_U = \{ex \in \{0,1\}^* |\ e$ is the encoding sequence of some CA M_e, and M_e accepts $x\}$. We construct a universal CA U accepting L_U such that U accepts ex in $c_e t(|x|)$ time if M_e accepts x in $t(|x|)$ time, where c_e is a constant depending only on e.

It is not difficult to verify whether the syntax of the encoding sequence e is proper in time proportional to $|e|$. Note that any polynomial in l is much smaller than $|e|$ because $|e| = 2^l$. In order to verify whether M_e accepts x, U simulates M_e on input x as follows. We denote the ith cell of M_e by s_i. Each cell s_i of M_e is simulated by $|e|$ cells of U. Namely, the cellular space of U is divided into *blocks*, $B_1, B_2, \ldots$, of the same length $|e|$, and the ith block B_i corresponds to M_e's cell s_i. Each block is divided into two tracks in order to store e and the state of s_i. Therefore, every block has all transition rules of M_e.

We can generate blocks of the same length as follows (see Fig. 2). The first and $|e|$th cells emit unit-speed and 1/2-speed pulses p_0, p_1 at step 0 to the right, respectively. Then the $|e|$th cell emits a 1/2-speed pulse p_2 at step $|e|$. Similarly, at steps $3|e|, 5|e|, 7|e|, \cdots$, cells at positions $2|e|, 3|e|, 4|e|, \cdots$ emit unit-speed pulses p_0. Markers are placed at positions where a pulse p_0 catches up with p_1. The encoding sequence e in a block can be copied into the next block in time proportional to $|e|$ (by using the firing squad synchronization (FSS) algorithm [15]). The ith symbol x_i of M_e's input x is moved to the ith block in $2|e| \cdot |x|$ time (the detail of this procedure is left to the reader).

After the above procedures, U starts to simulate M_e on input x. In order that every cell starts the simulation simultaneously, we use the FSS algorithm. Since each block has length $|e|$, a single step of M_e's cell s_i can be simulated by U's

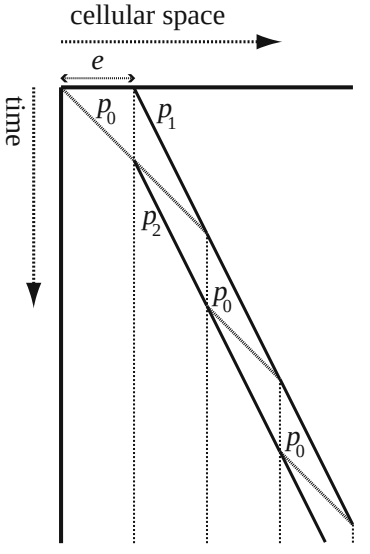


Fig. 2. Time-space diagram of CA. Markers are placed at regular intervals of length $|e|$.

block B_i in c'_e steps, where c'_e is a constant depending only on e. Therefore, there is a constant c_e such that U accepts ex in $c_e t(|x|)$ time if M_e accepts x in $t(|x|)$ time.

3.2 Recursive Padding

For a fixed encoding e, let $f(e)$ be the encoding of a CA $M_{f(e)}$ such that (i) $M_{f(e)}$ with the input string x changes x into ex and then (ii) $M_{f(e)}$ makes the computation of M_e on the input ex. (This definition is based on recursive function theory [18,21].)

For a fixed function h and a fixed CA M_1, let $g(h, M_1)$ be the encoding of a CA $M_{g(h,M_1)}$ which changes its input wx into $h(w)x$ and then makes the computation of M_1 on input $h(w)x$, where w is a valid encoding of some CA.

Now, consider CA $M_{f(g(f,M_1))}$ on input x. According to the definitions of f and g, (i) $M_{f(g(f,M_1))}$ on input x changes x into $g(f, M_1)x$, then (ii) $M_{f(g(f,M_1))}$ makes the computation of $M_{g(f,M_1)}$ on the input $g(f, M_1)x$ (i.e., $M_{g(f,M_1)}$ changes $g(f, M_1)x$ into $f(g(f, M_1))x$, and makes the computation of M_1 on input $f(g(f, M_1))x$). Therefore, $M_{f(g(f,M_1))}$ accepts x iff M_1 accepts $f(g(f, M_1))x$.

We analyse the relationship between the time complexities of $M_{f(g(f,M_1))}$ and M_1 as follows: We can convert the input x into $g(f, M_1)x$ and then into $f(g(f, M_1))x$ in time proportional to $|f(g(f, M_1))x|$. In order to start the computation of M_1 in every cell simultaneously, $M_{f(g(f,M_1))}$ performs the FSS algorithm, which can also be done in linear time. Hence, if M_1 accepts $f(g(f, M_1))x$ in $t(|f(g(f, M_1))x|)$ time, then $M_{f(g(f,M_1))}$ accepts x in $ct(|f(g(f, M_1))x|)$ time for some constant c.

Let $L \subseteq \{1\}^*$ be any recursive language, and let M be the deterministic CA which accepts L in $t(n)$ time. Now, we define (nondeterministic) CA M' which recursively pads the input string until the length becomes larger than $t(|x|)$.

The CA M' first verifies whether the input string is of the form $ex0^k$, where $x \in \{1\}^*$ and e is the encoding sequence of some CA M_e. So, M' verifies whether the input is an encoding sequence of a CA followed by an arbitrary number of 1's, which are further followed by an arbitrary number of 0's. Then, CA M' compares the values of $t(|x|)$ and $|x0^k|$ by (i) emitting a unit-speed pulse to the left from the rightmost 0 in $ex0^k$ and (ii) making the deterministic $t(|x|)$-step computation of M on x (see Fig. 3). If the computation of M halts before the pulse reaches the position of the first symbol of $x0^k$ (i.e., $t(|x|) < |x0^k|$), then M' halts with an accepting state iff M accepts x. If $t(|x|) \geq |x0^k|$, then M' pads $ex0^k$ to $ex0^{k'}$, where $k' > k$ is a nondeterministically chosen integer; M' performs the FSS algorithm in order to start the computation of the universal CA U on input $ex0^{k'}$.

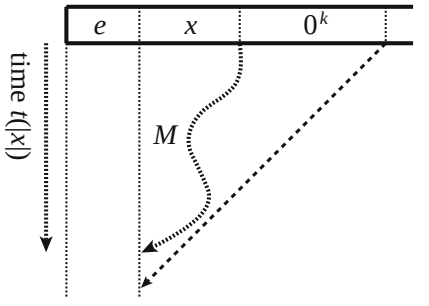


Fig. 3. M' makes the computation of M on x, and emits a unit-speed pulse

In the following, we analyse the properties of the above padding procedure. Consider the computation of M' on input $f(g(f, M'))x$. According to the definition, M' pads $f(g(f, M'))x$ to $f(g(f, M'))x0^k$ for nondeterministically chosen $k > 0$, and starts the computation of U on $f(g(f, M'))x0^k$. Since U is a universal CA, the computation of U on $f(g(f, M'))x0^k$ implies the computation of $M_{f(g(f,M'))}$ on input $x0^k$. According to the definition, $M_{f(g(f,M'))}$ first changes the input $x0^k$ to $g(f, M')x0^k$. The next task is to change $g(f, M')x0^k$ into $f(g(f, M'))x0^k$ and then to make the computation of M' on input $f(g(f, M'))x0^k$. According to the definition, M' compares the values of $t(|x|)$ and $|x0^k|$. If $t(|x|) < |x0^k|$, then M' halts with an accepting state iff M accepts x. If $|x0^k|$ has not yet been larger than $t(|x|)$, M' pads $x0^k$ to $x0^{k'}$ for nondeterministically chosen $k' > k$, and restarts the computation of U on $f(g(f, M'))x0^{k'}$. The CA M' recursively performs this procedure until the padding sequence becomes so long that $t(|x|) < |x0^{k'}|$.

3.3 Recursively Padding CA and Their Languages

Recall that $L \subseteq \{1\}^*$ is any recursive language, and M is the deterministic CA recognizing L in $t(n)$ time. We will prove

$$L(M_{f(g(f,M'))}) = \{x0^k \mid x \in L(M), k \geq 0\} \tag{1}$$

by induction on k running down from a sufficiently large k' to 0.

Let k' be a sufficiently large integer such that $t(|x|) < |x0^{k'}|$. As we mentioned in the last paragraph of Section 3.2, a computation of $M_{f(g(f,M'))}$ on $x0^{k'}$ implies a computation of M' on $f(g(f,M'))x0^{k'}$, and M' accepts it iff M accepts x. Therefore,

$$\begin{aligned} L(M_{f(g(f,M'))}) &= \{x0^{k'} \mid f(g(f,M'))x0^{k'} \in L(M')\} \\ &= \{x0^{k'} \mid x \in L(M)\}. \end{aligned} \tag{2}$$

From equation (2), one can see that $M_{f(g(f,M'))}$ accepts $x0^{k'}$ iff M accepts x, for any such large k'.

Consider an integer k such that $t(|x|) \geq |x0^k|$. Assume for induction that, for every $k' > k$, equation (2) holds. We observe the computation of $M_{f(g(f,M'))}$ on $x0^k$ (which implies the computation of M' on $f(g(f,M'))x0^k$.) Recall that the first task for M' was to pad $f(g(f,M'))x0^k$ to $f(g(f,M'))x0^{k'}$ for nondeterministically chosen $k' > k$, and the second task was to make the computation of U on $f(g(f,M'))x0^{k'}$. Thus,

$$L(M_{f(g(f,M'))}) = \{x0^k \mid f(g(f,M'))x0^{k'} \in L(U) \text{ for some } k' > k\}. \tag{3}$$

Since U is a universal CA, the computation of U on input $f(g(f,M'))x0^{k'}$ implies the computation of $M_{f(g(f,M'))}$ on input $x0^{k'}$. Therefore, the right-hand side of (3) can be rewritten as

$$L(M_{f(g(f,M'))}) = \{x0^k \mid x0^{k'} \in L(M_{f(g(f,M'))}) \text{ for some } k' > k\}. \tag{4}$$

From the induction hypothesis (see equation (2)), $M_{f(g(f,M'))}$ accepts $x0^{k'}$ iff M accepts x. Hence, the right-hand side of (4) is further rewritten as

$$L(M_{f(g(f,M'))}) = \{x0^k \mid x \in L(M)\}.$$

Thus, if equation (2) holds for every $k' > k$, then the same equation holds also for k. Hence, equation (1) holds for every $k \geq 0$.

3.4 Proof of Theorem 1

Let $t_2(n)$ be an arbitrary time-constructible function which is not bounded by $O(t_1(n+1))$. Assume for contradiction that there does not exist any language which can be accepted by $t_2(n)$-time CA but not by any $t_1(n)$-time CA. This

assumption implies that any $t_2(n)$-time computation in CA can be sped-up to $t_1(n)$ time.

Recall that $L \subseteq \{1\}^*$ is any recursive language, and M is the deterministic CA recognizing L in $t(n)$ time. Note that $t(n)$ can be taken as rapidly as we want by choice of L. In the following paragraphs, we will prove by induction on n that, *for each sufficiently long x accepted by M, U accepts $f(g(f, M'))x0^k$ of length n in $t_1(n)$ time for every $n \geq |f(g(f, M'))x|$*. Here, the computation of U on $f(g(f, M'))x0^k$ in $t_1(n)$ time implies the computation of $M_{f(g(f,M'))}$ on $x0^k$ in $O(t_1(n))$ time, since U is a universal CA. It is not difficult to show that there is an $O(t_1(n))$-time CA, say, $M'_{f(g(f,M'))}$, such that

$$\begin{aligned} L(M'_{f(g(f,M'))}) &= \{x \mid x0^k \in L(M_{f(g(f,M'))})\} \\ &= \{x \mid x \in L(M)\} = L(M) = L. \end{aligned} \tag{5}$$

The second equation holds because of equation (1). Equation (5) implies that any language L (recognized by the deterministic CA M with no time restriction) can be accepted by an $O(t_1(n))$-time CA $M'_{f(g(f,M'))}$, a contradiction.

It remains to show that U accepts $f(g(f, M'))x0^k$ of length n in $t_1(n)$ time. Consider a sufficiently large n such that $n > |f(g(f, M'))x| + t(|x|)$. In this case, since the padding sequence is sufficiently long, the computation of $M_{f(g(f,M'))}$ on $x0^k$ can be done in linear time (which is less than $t_2(n)$). From the assumption, any $t_2(n)$-time computation (of $M_{f(g(f,M'))}$ on $x0^k$) can be sped-up to $t_1(n)$ time. Therefore, U can accept $f(g(f, M'))x0^k$ in $t_1(n)$ time.

Consider an integer n such that $|f(g(f, M'))x| \leq n \leq |f(g(f, M'))x| + t(|x|)$. Assume for induction that, for each sufficiently long x accepted by M, U accepts $f(g(f, M'))x0^{k+1}$ of length $n' = n + 1$ in $t_1(n')$ time. Under this assumption, we will show that U accepts $f(g(f, M'))x0^k$ of length n in $t_1(n)$ time as follows.

Consider the nondeterministic computation tree of U on input $f(g(f, M'))x0^k$. The computation of U on $f(g(f, M'))x0^k$ implies the computation of $M_{f(g(f,M'))}$ on $x0^k$, which further implies the computation of M' on $f(g(f, M'))x0^k$ (see the last paragraph of Section 3.2). According to the definition, M' pads $f(g(f, M'))x0^k$ to $f(g(f, M'))x0^{k'}$ for nondeterministically chosen $k' > k$.

Consider the nondeterministic choice which pads $f(g(f, M'))x0^k$ to $f(g(f, M'))x0^{k+1}$. (We will analyse the height of the subtree rooted at $f(g(f, M'))x0^{k+1}$.) The time complexity for lengthening the padding sequence is $O(n)$. After lengthening the padding sequence, M' makes the computation of U on the padded input $f(g(f, M'))x0^{k+1}$. The computation of U on $f(g(f, M'))x0^{k+1}$ of length $n + 1$ can be done in $t_1(n + 1)$ time (because of the induction hypothesis). The total complexity is $t_1(n + 1) + O(n)$. From the assumption of the theorem, $t_2(n)$ is not bounded by $O(t_1(n + 1))$. Thus, U can accepts $f(g(f, M'))x0^k$ of length n in $t_2(n)$ time. From the assumption (of this section), any $t_2(n)$-time computation can be sped-up to $t_1(n)$ time. Therefore, U accepts $f(g(f, M'))x0^k$ of length n in $t_1(n)$ time. This completes the proof of Theorem 1.

4 Conclusion and Final Observations

In this paper, we presented a tight time-hierarchy theorem for nondeterministic cellular automata (NCA). It was shown that, for any time-constructible function $t_2(n)$ not bounded by $O(t_1(n+1))$, $t_2(n)$-time NCA are stronger than $t_1(n)$-time NCA.

Finally, we intuitively observe a recursive padding for the separation between nondeterministic classes of $t_1(N) = N^2/\log\log N$ and $t_2(N) = N^2$. Let L be any recursive language recognized by a deterministic CA (DCA) M in time $t(n) = 2^{2^n}$. Assume for contradiction that any $t_2(N)$-time (nondeterministic) computation is sped-up to $t_1(N)$ time.

We recursively pad the input x of length n until the length of $x00\cdots0$ becomes $N = 2^{2^n}$. Let $L_N = \{x0^{N-n} \mid x \in L\}$. Then, there is a CA accepting L_N in $t_2(N)$ time, since $t_2(N) = 2^{2^{n+1}}$ is larger than $t(n) = 2^{2^n}$. From the assumption, the $t_2(N)$-time computation for L_N is sped-up to $t_1(N)$ time. If such a $t_1(N)$-time computation exists, then there is a $t_2(N-1)$-time computation for L_{N-1} (see Section 3.4). Again, the $t_2(N-1)$-time computation is sped-up to $t_1(N-1)$ time, and thus there is a $t_2(N-2)$-time computation for L_{N-2}. By continuing this observation, one can see that there is a $t_2(n)$-time computation for $L_n = L$. Therefore, any recursive language L, which can be accepted by a 2^{2^n}-time DCA, can also be accepted by an n^2-time NCA. This contradicts the following simulation and separation results: (i) every n^2-time NCA can be simulated by a $2^{2^{n-1}}$-time DCA, and (ii) there is a language which can be accepted by a 2^{2^n}-time DCA but not by any $2^{2^{n-1}}$-time DCA [11].

References

1. S.A. Cook, A hierarchy for nondeterministic time complexity, *J. Comput. System Sci.* **7** (1973) 343–353.
2. S.A. Cook and R.A. Reckhow, Time bounded random access machines, *J. Comput. System Sci.* **7** (1973) 354–375.
3. M. Fürer, The tight deterministic time hierarchy, Proc. ACM Symp. on Theory of Computing, 8–16, 1982.
4. J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965) 285–306.
5. F.C. Hennie, One-tape off-line Turing machine computations, *Inform. Contr.* **8** (1965) 553–578.
6. O.H. Ibarra, A note concerning nondeterministic tape complexity, *J. Assoc. Comput. Mach.*, **19** (1972) 608–612.
7. K. Iwama and C. Iwamoto, Parallel complexity hierarchies based on PRAMs and DLOGTIME-uniform circuits, Proc. 11th IEEE Conf. on Computational Complexity, 1996, 24–32.
8. K. Iwama and C. Iwamoto, Improved time and space hierarchies of one-tape off-line TMs, Proc. 23rd Int'l Symp. on Mathematical Foundations of Computer Science, LNCS 1450, Springer, 1998, 580–588.
9. C. Iwamoto, T. Andou, K. Morita, and K. Imai, Computational complexity in the hyperbolic plane, Proc. 27th Int'l Symp. on Mathematical Foundations of Computer Science, LNCS 2420, Springer, 2002, 365–374.

10. C. Iwamoto, N. Hatayama, K. Morita, K. Imai, and D. Wakamatsu, Hierarchies of DLOGTIME-uniform circuits, in M. Margenstern (ed.): Machines, Computations and Universality (Proc. MCU 2004, Saint-Petersburg, Sep. 21–26, 2004), LNCS 3354, Springer, 2005, 211–222.
11. C. Iwamoto, T. Hatsuyama, K. Morita, and K. Imai, Constructible functions in cellular automata and their applications to hierarchy results, *Theoret. Comput. Sci.* **270** (2002) 797–809.
12. C. Iwamoto and M. Margenstern, Time and space complexity classes of hyperbolic cellular automata, *IEICE Trans. on Information and Systems*, **E87-D** 3 (2004) 700–707.
13. C. Iwamoto, Y. Nakashiba, K. Morita, and K. Imai, Translational lemmas for alternating TMs and PRAMs, Proc. 15th Int'l Symp. on Fundamentals of Computation Theory, LNCS 3623, Springer, 2005, 126–137.
14. K. Loryś, New time hierarchy results for deterministic TMs, Proc. 9th Symp. on Theoretical Aspects of Computer Science, LNCS 577, Springer, 1992, 329–336.
15. J. Mazoyer, A 6-state minimal time solution to the firing squad synchronization problem, *Theoret. Comput. Sci.*, **50** (1987) 183–238.
16. W.J. Paul, On time hierarchies, *J. Comput. System Sci.* **19** (1979) 197–202.
17. W.J. Paul, E.J. Prauß, and R. Reischuk, On alternation, *Acta Inform.* **14** (1980) 243–255.
18. H. Rogers Jr., Theory of recursive functions and effective computability, McGraw-Hill, New York, 1967.
19. J.I. Seiferas, Nondeterministic time and space complexity classes, MIT-LCS-TR-137, Proj. MAC, MIT, Cambridge, Mass., Sept. 1974.
20. J.I. Seiferas, M.J. Fischer, and A.R. Meyer, Separating nondeterministic time complexity classes, *J. Assoc. Comput. Mach.*, **25** 1 (1978) 146–167.
21. M. Sipser, Introduction to the theory of computation, PWS Publishing, Boston, Mass., 1997.

Decidability of Propositional Projection Temporal Logic with Infinite Models*

Zhenhua Duan and Cong Tian

Institute of Computing Theory and Technology
Xidian University, Xi'an, 710071, P.R. China
{zhhduan, ctian}@xidian.edu.cn

Abstract. This paper investigates the satisfiability of Propositional Projection Temporal Logic (PPTL) with infinite models. A decision procedure for PPTL formulas is formalized. To this end, Normal Form (NF) and Normal Form Graph (NFG) for PPTL formulas are defined and an algorithm constructing NFG for PPTL formulas is presented. Further, examples are also given to illustrate how the decision algorithm works.

Keywords: interval temporal logic; satisfiability; decidability; infinite model; model checking.

1 Introduction

Interval Temporal Logic (ITL) [3,4] is a useful formalism for the specification and verification of concurrent systems. In the past two decades, a number of axiom systems have been proposed [2, 4, 5, 13, 14, 18] to verify properties of concurrent systems. Although verification of programs can be managed by complete deductive systems of ITL, Model Checking approach which is an automatic method based on model checking algorithms has not extensively been studied in ITL.

Satisfiability and validity of formulas are fundamental issues in the model theory of a logic. Moreover, satisfiability plays an important role in the model checking approach. Within ITL community, several researchers have looked at decision procedures. Halpern and Moszkowski [3] proved the decidability of Quantifier Propositional ITL (QPITL) over finite time. Kono presented a tableaux-based decision procedure for QPITL with projection [12]. Bowman and Thompson [13] presented the first tableaux-based decision procedure for quantifier-free propositional ITL (PITL) over finite intervals with projection. To the best of our knowledge, all the existing decision procedures for ITL are confined in finite intervals. However, many reactive systems are designed not to terminate. So, to verify those systems using model checking, a decision procedure with infinite models is required.

Projection Temporal Logic (PTL) [6,8,11,19] is an extension of ITL. It extends ITL to include infinite models and a new projection construct, $(P_1, ..., P_m)\ prj\ Q$. The new projection construction can be treated as a combination of the parallel

* This research is supported by the NSFC Grant No. 60373103 and 60433010.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 521–532, 2007.
© Springer-Verlag Berlin Heidelberg 2007

$(P \parallel Q)$ and original projection construct, $(P\ proj\ Q)$ [3]. Formulas P_1, ..., P_m and Q are autonomous; each formula has the right to specify its own interval over which it is interpreted. Although formula Q is interpreted in a parallel way with formula $(P_1\ ;\ ...\ ;\ P_m)$ (; denotes the chop operator, see Section 2), the communication between them is only at rendezvous states, and the formulas may terminate at different time points.

Compared with the *proj* construct $(P\ proj\ Q)$ defined in [3], *prj* is more powerful. Firstly, $(P_1, ..., P_m)\ prj\ Q$ is able to handle terminal formulas (see Section 2) while $(P\ proj\ Q)$ construct cannot. Within $(P\ proj\ Q)$, the formula P is interpreted repeatedly over a series of consecutive subintervals whose endpoints form the interval over which Q is interpreted. This may result in repeating the same global state in the interpretation of Q several times if P is interpreted over a subinterval of zero length. In [13], Bowman and Thompson changed this by confining P not to be a terminal formula. However, this restriction causes the loss of flexibility since the formula P is necessary to be interpreted in a point state in some circumstances. Secondly, in $(P\ proj\ Q)$, the series of Ps and Q terminate at the same time. In practice, the formulas Ps and Q are not so regularly interpreted. In our definition, formulas P_1, ..., P_m and Q are autonomous; they are interpreted independently and may terminate at different time points. Thirdly, $(P\ proj\ Q)$ is only defined over finite intervals and hard to be extended to infinite intervals. In contrast, our projection construct is defined over both finite and infinite intervals. Finally, *prj* can subsume the central operator chop in ITL, $P; Q \equiv (P, Q)\ prj\ \varepsilon$, but *proj* cannot.

ITL has also been extended into infinite time by Moszkowski [4]. However, the chop construct is yet different from the one in PTL in the case of infinite models. Within ITL, $(P; Q)$ holds over an interval if and only if either the interval can be split into two parts and P holds on the first part and Q holds on the second part (the interval can be finite or infinite), or P holds over the whole interval and the interval is infinite; whereas within PTL, $(P; Q)$ is true only for the first case. That is, in PTL, P is only interpreted over a finite interval, while in ITL, P can be interpreted over an infinite interval. $(P; Q)$ is not satisfiable in PTL if P has only infinite models. However, the two constructs can express each other directly. Let $;_m$ and $;_d$ denote the chop operators in ITL and PTL respectively. Then we have,

$$P\ ;_m Q \equiv (P\ ;_d Q) \vee P \wedge \Box\ \bar{\varepsilon}$$
$$P\ ;_d Q \equiv (P \wedge \Diamond\varepsilon)\ ;_m Q$$

where $\Box$ (always) and $\Diamond$ (sometimes) are modal operators and $\bar{\varepsilon}$ means the current interval is not over. In the sense of *Until* construct in Linear Temporal Logic (LTL) [15], $P;_d Q$ can be viewed as the strong version while $P;_m Q$ can be thought of as the weak version of the chop construction.

Within PTL, plenty of logic laws have been formalized and proved [6, 11], and a decision procedure for checking satisfiability of Propositional Projection Temporal Logic (PPTL) formulas with finite models has been given in [9]. Nevertheless, to check the satisfiability of the underlying logic with infinite models, a decision procedure is also required.

Therefore, we are motivated to investigate the decidability of PPTL formulas with infinite models. To this end, Normal Form (NF) and Normal Form Graph (NFG) for PPTL formulas are defined. Further, an algorithm constructing NFGs for PPTL formulas is also formalized. Basically, the normal form is the same as we gave in [6, 8] for Tempura programs. An NFG is a useful formalism for constructing models of a PPTL formula.

Accordingly, a decision procedure based on NFGs for checking the satisfiability of PPTL formulas with infinite models is formalized in the paper. With this method, for a given formula, an NFG can be constructed by means of its normal form. A finite path from the root node to the ε node (i.e., terminating node, see Section 2) in the NFG of the formula corresponds to a finite model of the formula while an infinite path emanating from the root corresponds to an infinite model of the formula. It is clear that a formula is satisfiable if and only if there exists a finite or infinite path in its NFG. This decision procedure can also be used to check the satisfiability of PITL with minor changes [10].

The paper is organized as follows. The next section briefly presents the syntax, semantics and some logic laws of the underlying logic. Section 3 gives the definition of the normal form of PPTL formulas. In Section 4, the normal form graph is defined; further, an algorithm constructing NFGs is formalized, and the finiteness of NFGs is proved. A decision algorithm for checking the satisfiability of PPTL formulas with infinite models is demonstrated in Section 5. Conclusions are drawn in Section 6.

2 Propositional Projection Temporal Logic

Our underlying logic is Propositional Projection Temporal Logic (PPTL) [6, 11, 7, 19]. It is an extension of Propositional Interval Temporal Logic (PITL) [3].

Syntax. Let *Prop* be a countable set of atomic propositions. The formula P of PPTL is given by the following grammar:

$$P ::= p \mid \bigcirc P \mid \neg P \mid P_1 \vee P_2 \mid (P_1, ..., P_m)\ prj\ P$$

where $p \in Prop$, P_1 , ..., P_m and P are all well-formed PPTL formulas. A formula is called a state formula if it contains no temporal operators. The abbreviations $true$, $false$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $true \stackrel{\text{def}}{=} P \vee \neg P$ and $false \stackrel{\text{def}}{=} P \wedge \neg P$. Also we have the following derived formulas:

$$\begin{array}{llll}
A_1\ \varepsilon & \stackrel{\text{def}}{=} \neg \bigcirc true & A_2\ \bar{\varepsilon} & \stackrel{\text{def}}{=} \neg\varepsilon \\
A_3\ \bigcirc^0 P & \stackrel{\text{def}}{=} P & A_4\ \bigcirc^n P & \stackrel{\text{def}}{=} \bigcirc(\bigcirc^{n-1} P) \\
A_5\ len\ n & \stackrel{\text{def}}{=} \bigcirc^n \varepsilon & A_6\ skip & \stackrel{\text{def}}{=} len\ 1 \\
A_7\ \odot P & \stackrel{\text{def}}{=} \varepsilon \vee \bigcirc P & A_8\ \Diamond P & \stackrel{\text{def}}{=} true\ ;\ P \\
A_9\ \Box P & \stackrel{\text{def}}{=} \neg\Diamond\neg P & A_{10}\ P\ ;\ Q & \stackrel{\text{def}}{=} (P,\ Q)\ prj\ \varepsilon
\end{array}$$

Semantics. Following the definition of Kripke's structure [1], we define a state s over *Prop* to be a mapping from *Prop* to $B = \{true, false\}$, $s : Prop \longrightarrow B$. We will use $s[p]$ to denote the valuation of p at the state s.

An interval σ is a non-empty sequence of states, which can be finite or infinite. The length, $|\sigma|$, of σ is ω if σ is infinite, and the number of states minus 1 if σ is finite. To have a uniform notation for both finite and infinite intervals, we will use extended integers as indices. That is, we consider the set N_0 of non-negative integers and ω, $N_\omega = N_0 \cup \{\omega\}$, and extend the comparison operators, $=, <, \leq$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Moreover, we define $\preceq$ as $\leq -\{(\omega, \omega)\}$. To simplify definitions, we will denote σ as $< s_0, ..., s_{|\sigma|} >$, where $s_{|\sigma|}$ is undefined if σ is infinite. With such a notation, $\sigma_{(i..j)}$ $(0 \leq i \preceq j \leq |\sigma|)$ denotes the sub-interval $< s_i, ..., s_j >$ and $\sigma^{(k)}$ $(0 \leq k \preceq |\sigma|)$ denotes $< s_k, ..., s_{|\sigma|} >$. The concatenation of a finite σ with another interval (or empty string) σ' is denoted by $\sigma \cdot \sigma'$.

Let $\sigma =< s_0, s_1, \ldots, s_{|\sigma|} >$ be an interval and $r_1, \ldots, r_h$ be integers $(h \geq 1)$ such that $0 \leq r_1 \leq r_2 \leq \ldots \leq r_h \preceq |\sigma|$. The projection of σ onto $r_1, \ldots, r_h$ is the interval (namely projected interval)

$$\sigma \downarrow (r_1, \ldots, r_h) =< s_{t_1}, s_{t_2}, \ldots, s_{t_l} >$$

where $t_1, \ldots, t_l$ is obtained from $r_1, \ldots, r_h$ by deleting all duplicates. That is, $t_1, \ldots, t_l$ is the longest strictly increasing subsequence of $r_1, \ldots, r_h$. For instance,

$$< s_0, s_1, s_2, s_3, s_4 > \downarrow (0, 0, 2, 2, 2, 3) =< s_0, s_2, s_3 >$$

An interpretation is a quadruple $\mathcal{I} = (\sigma, i, k, j)$[1] , where σ is an interval, i, k are integers, and j an integer or ω such that $i \leq k \preceq j \leq |\sigma|$. We use the notation $(\sigma, i, k, j) \models P$ to denote that formula P is interpreted and satisfied over the subinterval $< s_i, ..., s_j >$ of σ with the current state being s_k. The satisfaction relation $(\models)$ is inductively defined as follows:

$I - prop$ $\mathcal{I} \models p$ iff $s_k[p] = true$, for any given atomic proposition p
$I - not$ $\mathcal{I} \models \neg P$ iff $\mathcal{I} \not\models P$
$I - or$ $\mathcal{I} \models P \vee Q$ iff $\mathcal{I} \models P$ or $\mathcal{I} \models Q$
$I - next$ $\mathcal{I} \models \bigcirc P$ iff $k < j$ and $(\sigma, i, k+1, j) \models P$
$I - prj$ $\mathcal{I} \models (P_1, ..., P_m)\ prj\ Q$ if there exist integers $k = r_0 \leq r_1 \leq ...$
$\leq r_m \leq j$ such that $(\sigma, 0, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_{l-1}, r_l) \models P_l$
$1 < l \leq m$, and $(\sigma', 0, 0, |\sigma'|) \models Q$ for one of the following σ' :
(a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, ..., r_m) \cdot \sigma_{(r_m+1..j)}$ or
(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, ..., r_h)$ for some $0 \leq h \leq m$

Satisfaction and Validity. A formula P is satisfied by an interval σ, denoted by $\sigma \models P$, if $(\sigma, 0, 0, |\sigma|) \models P$. A formula P is called satisfiable if $\sigma \models P$ for some σ. A formula P is valid, denoted by $\models P$, if $\sigma \models P$ for all σ.

Two formulas, P and Q, are equivalent, denoted by $P \equiv Q$, if $\models \Box(P \leftrightarrow Q)$. A formula P is called a terminal formula if $P \equiv P \wedge \varepsilon$, a non-local formula if $P \equiv P \wedge \bar{\varepsilon}$, and a local formula if P is a state or terminal formula.

Precedence Rules. In order to avoid an excessive number of parentheses, the following precedence rules are used:

[1] Parameter i is used to handle past operators and redundant with the current version of the underlying logic. However, to keep the consistency with our previous research, it is kept in the interpretation.

$$1\ \neg \quad 2\ \bigcirc, \odot, \Diamond, \Box \quad 3\ \wedge, \vee \quad 4\ \rightarrow, \leftrightarrow \quad 5\ \ prj,\ ;$$

where 1=highest and 5=lowest.

3 Normal Form of PPTL

Let Q be a PPTL formula and Q_p denote the set of atomic propositions appearing in Q. The normal form of Q can be defined as follows.

$$Q \equiv \bigvee_{j=1}^{n_0} (Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{n} (Q_{ci} \wedge \bigcirc Q'_i) \tag{1}$$

where $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$, $l = |Q_p|$, $1 \leq n$ (also n_0) $\leq 3^l$, $1 \leq m$ (also m_0) $\leq l$; $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, $\dot{r}$ denotes r or $\neg r$; Q'_i is a general PPTL formula[2]. In some circumstances, for convenience, we write $Q_e \wedge \varepsilon$ instead of $\bigvee_{j=1}^{n_0} (Q_{ej} \wedge \varepsilon)$ and $\bigvee_{i=1}^{r} (Q_i \wedge \bigcirc Q'_i)$ instead of $\bigvee_{i=1}^{n} (Q_{ci} \wedge \bigcirc Q'_i)$. Thus,

$$Q \equiv (Q_e \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (Q_i \wedge \bigcirc Q'_i) \tag{2}$$

where Q_e and Q_i are state formulas or $true$. Further, in a normal form, if $\bigvee_i Q_i \equiv true$ and $\bigvee_{i \neq j} (Q_i \wedge Q_j) \equiv false$, then this normal form is called a complete normal form. The complete normal form plays an important role in transforming the negation of a PPTL formula into its normal form. For example, if P has been written to its complete normal form:

$$P \equiv P_e \wedge \varepsilon \vee \bigvee_{i=1}^{r} (P_i \wedge \bigcirc P'_i) \tag{3}$$

then we have,

$$\neg P \equiv \neg P_e \wedge \varepsilon \vee \bigvee_{i=1}^{r} (P_i \wedge \bigcirc \neg P'_i)$$

For any PPTL formula P, P can be rewritten to its normal form and complete normal form. The details of the proofs and the algorithms transforming PPTL formulas into normal forms and complete normal forms can be found in [9].

Obviously, the normal form of a formula enables us to rewrite the formula into two parts: the present and future ones. The present part is a state formula while the future part is either ε or a next formula. This normal form inspires us to construct a graph for describing the models of the formula.

4 Normal Form Graph

Our decision algorithm is based on a so called Normal Form Graph (NFG). It is constructed according to the normal form.

4.1 Definition of NFG

For a PPTL formula P, the NFG of P is a directed graph, $G = (CL(P), EL(P))$, where $CL(P)$ denotes the set of nodes and $EL(P)$ denotes the set of edges in

[2] It is an exercise to prove $n, n_0 \leq 3^l$.

the graph. In $CL(P)$, each node is specified by a formula in PPTL, while in $EL(P)$, each edge is identified by a triple (Q, Q_e, R). Where Q and R are nodes and Q_e is labeling of the directed arc from Q to R. $CL(P)$ and $EL(P)$ of G can be inductively defined as in Definition 1. Note that the normal form employed in this definition is normal form (1).

Definition 1. For a PPTL formula P, set of nodes, $CL(P)$, and set of of edges, $EL(P)$, connecting nodes in $CL(P)$ are inductively defined as follows:

1. $P \in CL(P)$;
2. For all $Q \in CL(P) \setminus \{\varepsilon, false\}$, if $Q \equiv \bigvee_{j=1}^{h} (Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{k} (Q_{ci} \wedge \bigcirc Q'_i)$, then $\varepsilon \in CL(P)$, $(Q, Q_{ej}, \varepsilon) \in EL(P)$ for each j, $1 \leq j \leq h$; $Q'_i \in CL(P)$, $(Q, Q_{ci}, Q'_i) \in EL(P)$ for all i, $1 \leq i \leq k$;
3. $CL(P)$ and $EL(P)$ are only generated by 1 and 2; thus, the NFG of P can be defined as a directed graph G given by
4. $G = (CL(P), EL(P))$.

In the NFG of P, the root node P is denoted by a double circle, ε node by a small black dot, and each of other nodes by a single circle. If the main operator of a node (formula) is chop (;), a letter F is placed in the circle or double circle. Each of the edges is denoted by a directed arc connecting two nodes. Examples of NFGs are depicted in Fig.1.

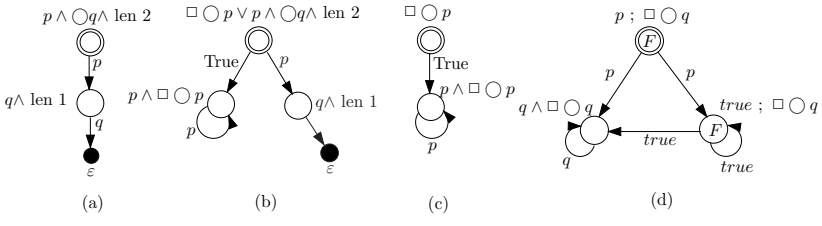


Fig. 1. Examples of NFGs

Intuitively, the NFG of formula P describes models of formula P since it is constructed according to the normal form. However, the NFG of formula P generated by Definition 1 might wrongly describe models of P with the chop construct, $Q_1; Q_2$, when Q_1 has infinite models.

To solve the problem, two cases need to be considered for chop construct $Q_1; Q_2$. 1) Q_1 has only infinite models. In this case, $Q_1; Q_2$ has no models, so edges might not depart from it. 2) Q_1 has both finite and infinite models. In this case, we need to eliminate all infinite models of Q_1. Based on the above analysis, we have formalized two algorithms, *NFG* and *Simplify* to generate a subgraph of the NFG for a given formula so that models of the formula can correctly be generated. In algorithm *NFG*, a label F is employed to indicate that a node in a cycle can only be repeated for finitely many times. Therefore, whenever the chop operator is the main operator of a formula, F is placed in the node as a mark.

4.2 Algorithm for Constructing NFG

In the following, algorithm *NFG* for constructing the NFG of a PPTL formula is presented. It is merely a sketch of the implementation of Definition 1. The algorithm uses *mark*[] to indicate whether or not a formula needs to be decomposed. If $mark[P] = 0$ (unmarked), then P needs further to be decomposed, otherwise $mark[P] = 1$ (marked), thus P has been decomposed or needs not to be done. Note that algorithm *NFG* employs algorithm *NF* [9] to transform a formula into its normal form. In the algorithm, we consider the chop construct in the form of $R \equiv Q_1; Q_2$ carefully since Q_1 may only have infinite models and cause R to be $false$. Therefore, if Q_1 has no finite models it needs not further to be decomposed and marked with 1 ($mark[R] = 1$) otherwise it needs further to be dealt with and marked with 0 ($mark[R] = 0$). Also label F is employed to denote the chop formula cannot be repeated infinitely many times. Further, in the algorithm, two global boolean variables AddE and AddN are employed to indicate whether or not ε and the next formulas in the normal form are encountered respectively. Note also that the algorithm only deals with formulas in a pre-prepared form in which only $\vee, \wedge$ and $\neg$, as well as temporal operators $\bigcirc$, ;, $\Box$ and prj are contained. Others such as $\rightarrow$, $\leftrightarrow$, $\Diamond$, $\neg\neg$ etc. can be eliminated since they can be expressed by the basic operators.

function *NFG*(P)
/* precondition: P is a PPTL formula in pre-prepared form*/
/* postcondition: *NFG*(P) computes NFG of P, $G = (CL(P), EL(P))$*/
begin function
/*initialization*/
$CL(P) = \{P\}$; $EL(P) = \phi$; $mark[P] = 0$; AddE = AddN = 0;
while there exists $R \in CL(P) \setminus \{\varepsilon, false\}$, and $mark[R] == 0$
do $mark[R] = 1$; /*marking R is decomposed*/
if R *is* $Q_1 ; Q_2$ /*computing NFG of Q_1*/
then add label F in node R; $G' = (CL(Q_1), EL(Q_1)) = NFG(Q_1)$;
if $\varepsilon \notin CL(Q_1)$ **then continue**;
/*R needs not decomposed, jump to while*/
$Q = NF(R)$;
case
Q is $\bigvee_{j=1}^{h} Q_{ej} \wedge \varepsilon$: AddE=1; /*first part of NF needs added*/
Q is $\bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$: AddN=1; /*second part of NF needs added*/
Q is $\bigvee_{j=1}^{h} Q_{ej} \wedge \varepsilon \vee \bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$: AddE=AddN=1;
/*both parts of NF needs added*/
end case
if AddE == 1 **then** /*add first part of NF*/
$CL(P) = CL(P) \cup \{\varepsilon\}$;

$EL(P) = EL(P) \cup \bigcup_{j=1}^{h} \{(R, Q_{ej}, \varepsilon)\}$;
AddE=0;
if AddN == 1 **then** /*add second part of NF*/
for $i = 1$ **to** k **do if** Q'_i is $false$
then $mark[Q'_i]$=1;
/*Q'_i needs not decomposed*/
else if $Q'_i \notin CL(P)$
then $mark[Q'_i]$=0;
/*Q'_i needs decomposed*/
$CL(P) = CL(P) \cup \bigcup_{i=1}^{k} \{Q'_i\}$;
$EL(P) = EL(P) \cup \bigcup_{i=1}^{k} \{(R, Q_i, Q'_i)\}$;
AddN=0;
end while
return G;
end function

4.3 Finiteness of NFG

In the NFG of PPTL formula P generated by algorithm *NFG*, the set $CL(P)$ of nodes and the set $EL(P)$ of edges are inductively produced by repeatedly rewriting the unmarked nodes into their normal form. So one question we have to answer is whether or not the rewriting process terminates. Fortunately, we can prove that, for any PPTL formula P, the number of nodes in $CL(P)$ are finite, that is, $|CL(P)| = k \in N_0$. To prove this, Lemma 1 $\sim$ 6 are needed. The proofs of these lemmas and theorems can be found in [10].

Lemma 1. For any PPTL formula P, $|CL(\bigcirc P)| \leq |CL(P)| + 1$.

The lemma tells us that the number of nodes in the NFG of $\bigcirc P$ is at most the number of nodes in the NFG of P plus one.

Lemma 2. If Q is a PPTL formula and P_e is a state formula, then $|CL(P_e \wedge Q)| \leq |CL(Q)| + 1$.

The lemma indicates that the number of nodes in the NFG of a formula being conjunctive with a state formula is no more than the number of nodes in its own NFG plus one.

Lemma 3. For PPTL formulas P and Q, $|CL(P \vee Q)| \leq |CL(P)| + |CL(Q)| + 1$.

This lemma shows us that the number of nodes in the NFG of formula $P \vee Q$ is no more than the sum of numbers of nodes in NFGs of formulas P and Q plus one.

Lemma 4. For a PPTL formula P, $|CL(\neg P)| \leq |CL(P)| + 1$.

This lemma indicates the relationship between NFGs of P and $\neg P$.

Lemma 5. If P and Q are PPTL formulas, then $|CL(P\ ;Q)| \leq |CL(P)| + |CL(Q)| + 1$.

Similar to Lemma 3, this lemma shows us that the number of nodes in NFG of $P\ ;Q$ is no more than the sum of numbers of nodes in NFGs of P and Q plus one.

Lemma 6. If $|CL(P_i)| = k_i \in N_0$ $(1 \leq i \leq m)$ and $|CL(Q)| = k_0 \in N_0$, then $|CL((P_1, ..., P_m)\ prj\ Q)| = k \in N_0$.

This lemma predicates only that if the numbers of nodes in the NFGs of $P_1, ..., P_m$ and Q are finite then the number of nodes in the NFG of $(P_1, ..., P_m)\ prj\ Q$ is also finite.

Theorem 1. For any PPTL formula P, $|CL(P)| = k \in N_0$.

The theorem can be proved by Lemma 1 $\sim$ Lemma 6. It convinces us that for any PPTL formula P, the number of nodes in the NFG of P is finite. This is critical since it guarantees that algorithm *NFG* terminates. Furthermore, this enables us to develop a decision procedure for checking the satisfiability of PPTL formulas based on algorithm *NFG*. In addition, the NFG of a formula P contains all models of P, including both finite and infinite ones. Hence, our decision procedure not only gives an algorithm for checking the satisfiability of a formula P but also constructs all models of formula P.

5 Decision Procedure for PPTL Formulas

5.1 Path and Satisfiability

In the NFG of formula Q, a finite path, $\Pi = \langle Q, Q_e, Q_1, Q_{1e}, ..., \varepsilon \rangle$, is an alternating sequence of nodes and edges from the root to ε node, while an infinite path, $\Pi = \langle Q, Q_e, Q_1, Q_{1e}, ... \rangle$, is an infinite alternating sequence of nodes and edges emanating from the root. In fact, a path (finite or infinite) in the NFG of a formula Q corresponds to a model of Q. The fact is concluded in Theorem 2 and 3.

Theorem 2. A formula Q can be satisfied by finite models if and only if there exist finite paths in the NFG of Q.

Theorem 3. A formula Q can be satisfied by infinite models if and only if there exist infinite paths in the NFG of Q.

The proof of the Theorem 2 can be found in [9]. To prove Theorem 3, Lemma 7 and Lemma 8 are needed. The two lemmas are useful since they tell us the relationship between the models and paths in the NFG of a PPTL formula.

Lemma 7. Given a PPTL formula Q, if there exists an infinite path in NFG of Q, a corresponding infinite model of Q can be constructed.

Lemma 8. Given a PPTL formula Q, if there exists an infinite model of Q, a corresponding infinite path in NFG of Q can be found.

The proofs of Lemma 7 and Lemma 8 can be found in [10]. Theorem 2 and 3 confirm that a PPTL formula is satisfiable if and only if there exist finite or infinite paths in its NFG.

5.2 Decision Procedure

In an NFG, some nodes might have no successors (e.g. $\Box \bigcirc p \,;q$). These nodes are redundant and can be removed. Algorithm *Simplify* is useful for eliminating redundant nodes of an NFG.

function *Simplify*(G)
/* precondition: $G = (CL(P), EL(P))$ is an NFG of PPTL formula P*/
/* postcondition: *Simplify*(G) computes an NFG of P, $G' = (CL'(P), EL'(P))$, which contains no redundant nodes*/
begin function
$CL'(P) = CL(P)$; $EL'(P) = EL(P)$;
while $\exists\ R \in CL'(P)$ and R is not ε and has no edges departing from
do $CL'(P) = CL'(P) \setminus R$; $EL'(P) = EL'(P) \setminus \bigcup_i (R_i, R_e, R)$;
/* $\bigcup_i (R_i, R_e, R)$ denotes the set of edges connecting to node R*/
end while
return G';
end function

The algorithm is terminable since the nodes of an NFG is finite. In the simplified NFG of P, a finite or infinite path can readily be constructed. Obviously, P is satisfiable if and only if there exist finite or infinite paths in the NFG of P. Consequently, a decision procedure for checking the satisfiability of a PPTL formula P can be constructed based on the NFG of P. In the following, a skeleton of the procedure, algorithm *Check* in pseudo code, is given.

function *Check*(P)
/* precondition: P is a PPTL formula*/
/* postcondition: *Check*(P) checks whether formula P is satisfiable or not.*/
begin function
$G = NFG(P)$;
$G' = Simplify(G)$;
if G' is empty, **return** unsatisfiable;
else return satisfiable;
end function

Example. Check the satisfiability of formula $Q \equiv (skip,\ p \wedge \Box \bigcirc p,\ \varepsilon)\ prj\ \bigcirc^2 q$.

To check the satisfiability of formula $Q \equiv (skip,\ p \wedge \Box \bigcirc p,\ \varepsilon)\ prj\ \bigcirc^2 q$, NFG of formula Q can be constructed according to **function** *NFG*, and then simplified by **function** *Simplify*. The details of the procedure are shown as follows.

Step 1. Build NFG of formula Q (see Fig.2);

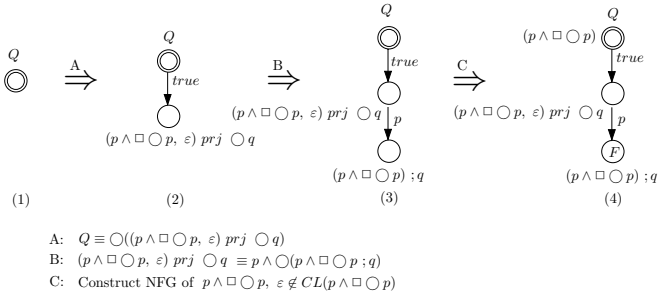


Fig. 2. Constructing the NFG of Q

Step 2. Obtain simplified NFG of formula Q (see Fig.3);

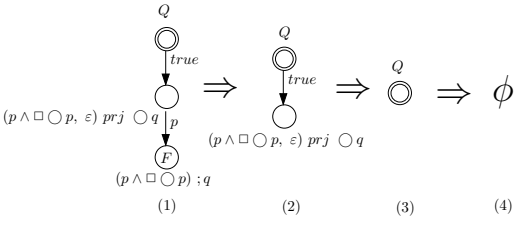


Fig. 3. Simplifying the NFG of Q

Step 3. The simplified NFG of Q is empty, so formula Q is unsatisfiable.

6 Conclusion

In this paper, we have given a decision procedure for PPTL formulas with infinite models. A modified version of the decision algorithm can also be applied to PITL with infinite models [10]. This enables us to verify properties of concurrent systems with PPTL and PITL by means of model checking. However, the existing model checkers such as SPIN [16] and SMV [17] cannot directly be used to check PPTL or PITL formulas since PPTL and PITL are involved both finite and infinite models due to the chop operator while SPIN and SMV are only dealt with infinite models for PLTL and CTL formulas. Therefore, to check PPTL and PITL formulas, a model checker for PPTL and PITL is required. We believe that the decision procedures we give in this paper are useful in this respect. At the present, we have developed a model checker based on SPIN for PPTL. Further, we are also motivated to develop a practical verification environment with a set of supporting tools in the near future.

References

1. S.A. Kripke. Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9, 67-96, 1963.
2. R. Rosner and A. Pnueli. A choppy logic. First Annual IEEE Symposium on Logic In Computer Science, LICS, 306-314, 1986.
3. B.C. Moszkowski. *Reasoning about digital circuits.* Ph.D Thesis, Department of Computer Science, Stanford University. TRSTAN-CS-83-970,1983.
4. B. Moszkowski. A Complete Axiomatization of Interval Temporal Logic with Infinite Time. lics, p. 241, 15 th Annual IEEE Symposium on Logic in Computer Science (LICS'00), 2000.
5. Zhou Chaochen , C.A.R. Hoare and A.P. Ravn. A calculus of duration. Information Processing Letters 40(5): 269-275, 1991.
6. Z. Duan. Temporal Logic and Temporal Logic Programming. Science Press of China, 2006.
7. Z. Duan and M. Koutny. A Framed Temporal Logic Programming Language. Journal of Computer Science and Technology, Vol,19, No.3, pp.341-351, May, 2004.
8. Z. Duan, X. Yang and M. Koutny. Semantics of Framed Temporal Logic Programs. Proceedings of ICLP 2005, Barcelona, Spain, LNCS 3668, pp256-270, Oct. 2005.
9. Z. Duan and L. Zhang. A Decision Procedure for Propositional Projection Temporal Logic. Technical Report No.1, Institute of computing Theory and Technology, Xidian University, Xi'an P.R.China, 2005.
   http://www.paper.edu.cn/process/download.jsp?file=200611-427
10. Z. Duan and C. Tian. Decison Prodedure for Propositional Projection Temporal Logic with Infinite Models. Technical Report No.1, Institute of computing Theory and Technology, Xidian University, Xi'an P.R.China, 2006.
    http://www.paper.edu.cn/process/download.jsp?file=200611-444
11. Z. Duan, M. Koutny and C. Holt. Projection in temporal logic programming. In F. Pfenning (ed.), Proceedings of Logic Programming and Automatic Reasoning, LNAI, Springer-Verlag, vol 822, 333-344, 1994.
12. S. Kono. A combination of clausal and non-clausal temporal logic programs. In Lecture Notes in Artificial Intelligence, vol. 897, pages 40-57. SpringerVerlag, 1993.
13. H. Bowman and S. Thompson. A decision procedure and complete axiomatization of interval temporal logic with projection. Journal of logic and Computation 13(2),195-239, 2003.
14. Dutertre B. Complete proof systems for first order interval temporal logic. In Proceedings of LICS'95, (1995)36-43.
15. Z. Manna and A. Pnueli. The temporal logic of reactive and concurrent systems. Springer-Verlag, 1992.
16. Gerard J. Holzmann. The Model Checker Spin, IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.
17. K.L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers, 1993.
18. Hanpin Wang and Qiwen Xu, Completeness of Temporal Logics over Infinite Models, Discrete Applied Mathematics 136 (2004) 87-103, Elsevier B.V.
19. X. Yang Z. Duan, Operational Semantics of Framed Temporal Logic Programs, Proceedings of SOFSEM 2007, Harrachov, Czech, LNCS 4362,pp.566-578,Jan. 2007.

Separation of Data Via Concurrently Determined Discriminant Functions*

Hong Seo Ryoo** and Kwangsoo Kim

Division of Information Management Engineering, Korea University
1, 5-Ka, Anam-Dong, Seongbuk-Ku, Seoul, 136-713, Korea
Phone: +82-2-3290-3394; Fax: +82-2-929-5888
{hsryoo,kksoo}@korea.ac.kr

Abstract. This paper presents a mixed $0-1$ integer and linear programming (MILP) model for separation of data via a finite number of nonlinear and nonconvex discriminant functions. The MILP model concurrently optimizes the parameters of the user-provided individual discriminant functions and implements a decision boundary for an optimal separation of data under analysis.

The MILP model is extensively tested on six well-studied datasets in data mining research. The comparison of numerical results by the MILP-based classification of data with those produced by the multisurface method and the support vector machine in these experiments and the best from the literature illustrates the efficacy and the usefulness of the new MILP-based classification of data for supervised learning.

Keywords: data classification, machine learning, mixed integer and linear programming.

1 Introduction

Separation/classification of data deals with discrimination of different types of data and has wide applications in cancer diagnosis and prognosis [1,2], the scoring of credit card applications [3], the prediction of defects during the assembly of disk drives [4], face detection [5] and the scouting of professional athletes [6], to name a few.

Let us consider optimal separation of two types of data $\mathcal{A}$ and $\mathcal{B}$, without loss of generality (e.g., [7]), by a finite number of discriminant functions. When the l_1-norm distance based error metric is used, the optimal separation of data by a single discriminant function can be accomplished via linear programming (LP) techniques. The optimal separation of data via two or more concurrently determined discriminant functions, however, is a difficult problem from the optimization point of view (e.g., [8,9,10]). For example, [11] addressed separation

* This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2005-003-D00445).

** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 533–541, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of data by two concurrently determined hyperplanes as solving three different bilinear programs to global optimality in succession, and the global optimization of bilinear programs is well-known to be $\mathcal{NP}-$hard (e.g., [12]).

The difficulty associated with the optimal separation of data by more than a single discriminant function has invited the development of greedy, local optimization, or successive LP-based techniques in the literature (e.g., [2,13,14,15]). In brief, these heuristic approaches make a non-optimal use of the individual discriminant functions and, as a result, train decision rules that are more complex than necessary or implement decision rules that perform differently in training and testing.

In this paper, we present a general-purpose mixed integer and linear programming (MILP) model for concurrent optimization of the unknown parameters of the user-provided discriminant functions for optimal separation of data under analysis. Any function can be used with the proposed MILP model as long as it is linear in the parameters, and the proposed model implements a highly nonlinear, nonconvex and/or disjoint decision function (refer to Figure 1) for the data at hand by exploiting advances in solution techniques for MILP and powerful commercial MILP solvers available nowadays (e.g., [16].) Furthermore, in terms of performance, the new MILP-based learning compares favorably with well-established learning methodologies for supervised learning.

The organization of this paper is as follows. In Section 2, we develop a new l_1-norm based error metric and use it to develop the MILP classification model. In Section 3, we apply the MILP-based classification methodology to analyze six well-studied datasets in machine learning research from [17] and compare classification results with those produced by the multisurface method [18] and the support vector machines [19] in the identical experimental setting and also with the best results from the literature to the best of our knowledge. In summary, the proposed MILP-based learning produced the best results on four of the six datasets, and the comparative study in this section illustrates well the efficacy and the usefulness of the new MILP-based classification methodology in supervised learning. Concluding remarks are provided in Section 4.

2 Model Development

Denote by $\bullet_i$ an observation/data of type $\bullet$, where $\bullet \in \{\mathcal{A}, \mathcal{B}\}$. Let $\overline{\bullet}$ denote the complement of type $\bullet$ with respect to set $\{\mathcal{A}, \mathcal{B}\}$. Suppose that there are $m_\bullet$ observations of type $\bullet$ data for $\bullet \in \{\mathcal{A}, \mathcal{B}\}$. Let K denote the index set of k user-provided discriminant functions f_k, $k \in K$, whose parameters ω_k are to be determined via training.

To aid in understanding, refer to Figure 2, where two types of data $\mathcal{A}$ and $\mathcal{B}$, indicated by dots and crosses, respectively, are separated by a decision surface composed of a quadratic surface f_1 (recall that quadratic functions are linear in parameters.) and a linear function f_2. With regard to the goal of placing class $\mathcal{B}$ data in the region above the surface indicated by the arrows, the data $\mathcal{A}_i$ is misclassified if it is placed above both of the two discriminant functions while

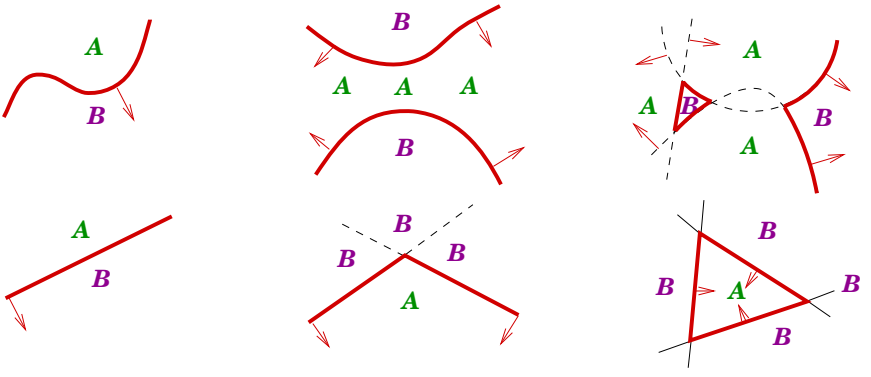


Fig. 1. Example of k piecewise nonlinear, nonconvex decision surfaces for $k = 1, 2, 3$

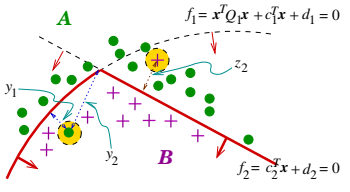


Fig. 2. Illustration of data separation via 2 discriminant functions

the data $\mathcal{B}_j$ is misclassified if it is placed below any of the two discriminant functions. Using the l_1-norm distance metric, the amount of misclassification associated with $\mathcal{A}_i$ and $\mathcal{B}_j$ is measured by $y_{ik} = \max\{f_k(\mathcal{A}_i, \omega_k), 0\}$ and $z_{jk} = \max\{-f_k(\mathcal{B}_j, \omega_k), 0\}$, respectively. Based on this, we develop an optimization model

$$\min_{\omega, \mathbf{y}, \mathbf{z}} \quad \sum_{i=1}^{m_\bullet} \prod_{k \in K} y_{ik} + \sum_{j=1}^{m_{\overline{\bullet}}} \sum_{k \in K} z_{jk}$$

$$\text{s.t.} \quad -f(\bullet_i, \omega_k) + y_{ik} \geq 0, \qquad i = 1, \ldots, m_\bullet,\ k \in K \tag{1}$$

$$f(\overline{\bullet}_j, \omega_k) + z_{jk} \geq 0, \qquad j = 1, \ldots, m_{\overline{\bullet}},\ k \in K \tag{2}$$

$$\mathbf{y} \in I\!R_+^{m_\bullet \times |K|}, \mathbf{z} \in I\!R_+^{m_{\overline{\bullet}} \times |K|}, \tag{3}$$

for $\bullet \in \{\mathcal{A}, \mathcal{B}\}$ for the optimal separation of data with $\bullet$ type data placed below the decision surface made up of k discriminant functions f_k, $k \in K$.

Refer back to Figure 2 and note that $\mathcal{A}_i$ is misclassified if the minimum of y_{ik}'s is greater than 0, hence that $r_i := \min_{k \in K}\{y_{ik}\}$ is a valid estimator of the misclassification error associated with $\mathcal{A}_i$, $i = 1, \ldots, m_\mathcal{A}$. This allows us to develop an alternative classification model for $\bullet \in \{\mathcal{A}, \mathcal{B}\}$ as follows:

$$\text{(PNC)} \quad \left| \begin{array}{ll} \min\limits_{\omega,\mathbf{y},\mathbf{z},\mathbf{r}} & \sum\limits_{i=1}^{m_\bullet} r_i + \sum\limits_{j=1}^{m_{\overline{\bullet}}} \sum\limits_{k\in K} z_{jk} \\ \text{s.t.} & (1),(2),(3) \\ & \mathbf{r} \in I\!R_+^{m_\bullet}, \end{array} \right.$$

where $r_i = \min_{k\in K}\{y_{ik}\}$.

The objective function of (PNC) above is nonlinear and nonconvex, and we linearize this function as follows. First, let $u_{ik} = y_{ik}x_{ik}$, $i = 1,\ldots,m_\bullet$, $k \in K$, and let $\sum_{k\in K} x_{ik} = 1$, where $x_{ik} \in \{0,1\}$, $i = 1,\ldots,m_\bullet$, $k \in K$. Next, underestimate $u_{ik}(= y_{ik}x_{ik})$ by its piecewise linear and convex envelope (e.g., [12,20])

$$u_{ik} \geq \max\{y_{ik} + Mx_{ik} - M,\, 0\},\ \forall k,$$

where M is an upper bound on y_{ik} (that can be normalized to 1.) This yields the following MILP model that concurrently determines the parameters of k discriminant functions for the best separation of data under analysis:

$$\text{(PNC}_{01}\text{)} \quad \left| \begin{array}{lll} \min\limits_{\omega,\mathbf{y},\mathbf{z},\mathbf{u},\mathbf{x}} & \sum\limits_{i=1}^{m_\bullet} \sum\limits_{k\in K} u_{ik} + \sum\limits_{j=1}^{m_{\overline{\bullet}}} \sum\limits_{k\in K} z_{jk} & \\ \text{s.t.} & (1),(2),(3) & \\ & \sum\limits_{k\in K} x_{ik} = 1, & i = 1,\ldots,m_\bullet \\ & u_{ik} - y_{ik} - x_{ik} \geq -1, & i = 1,\ldots,m_\bullet,\ k \in K \\ & \mathbf{u} \in I\!R_+^{m_\bullet\times|K|}, \mathbf{x} \in \{0,1\}^{m_\bullet\times|K|},\ \mathbf{y} \leq \mathbf{1} & \end{array} \right.$$

for $\bullet \in \{\mathcal{A},\mathcal{B}\}$.

Theorem 1. *(PNC) and (PNC$_{01}$) are equivalent.*

Proof. We establish the equivalence by showing that an optimal feasible solution of any one of the two programs can be used for constructing an optimal and feasible solution of the other.

Suppose that $(\omega^*, \mathbf{y}^*, \mathbf{z}^*, \mathbf{u}^*, \mathbf{x}^*)$ is an optimal feasible solution of (PNC$_{01}$). First, note that constraints (1), (2) and (3) in (PNC) are trivially satisfied by $\omega^*, \mathbf{y}^*$, and $\mathbf{z}^*$. Next, note for each i $(i = 1,\ldots,m_\bullet)$ that $u_{ik}^* > 0$ and $x_{ik} = 1$ implies that $y_{il}^* \geq y_{ik}^*$ for every $l \in K \setminus \{k\}$. This owes to the optimality of the solution at hand: the nature of minimization selects the minimum u_{ik} by letting $x_{ik}^* = 1$ and $x_{il}^* = 0$ for all $l \in K \setminus \{k\}$. Let $r_i = u_{ik}$ for $i = 1,\ldots,m_\bullet$ and use it along with $\omega^*, \mathbf{y}^*$ and $\mathbf{z}^*$ to obtain an optimal feasible solution of (PNC).

For the reverse direction, suppose that $(\omega^*, \mathbf{y}^*, \mathbf{z}^*, \mathbf{r}^*)$ is an optimal feasible solution of (PNC). For each $i \in \{1,\ldots,m_\bullet\}$, let $x_{ik} = 1$ for $k \in K$ corresponding to the minimum of y_{ik}^*'s and let $u_{ik} = y_{ik}^*$ for all $k \in K$. Now, $\mathbf{x}$ and $\mathbf{u}$ along with $\omega^*, \mathbf{y}^*$ and $\mathbf{z}^*$ form an optimal and feasible solution of (PNC$_{01}$).

Note in the above that the convexification of the objective function of (PNC) introduced additional $m_\bullet|K|$ $0-1$ and $m_\bullet|K|$ real variables and $m_\bullet(|K|+1)$

constraints in the formulation of (PNC_{01}). Although larger in size than (PNC) and the first classification model we developed in this section, (PNC_{01}) can exploit for its solution rich advances in theory, techniques and algorithms for MILP and powerful commercial MILP solvers like ILOG CPLEX [16], hence can readily be used for nonlinear and nonconvex separation of data.

3 Experiments with Six Well-Studied Datasets

For testing the usefulness of the new MILP-based classification of data, henceforth referred to as PNC, we used six databases that are commonly used in machine learning research from [17]. The six datasets contain incomplete data, and we used only those data with complete information in these experiments. We summarize information on the six datasets as tested in Table 1 and refer interested readers to [17] for additional information.

Table 1. Datasets studied

Database (abbreviation)	Number of Observations
	Total (Class $\mathcal{A}$, Class $\mathcal{B}$)*
Australian credit card (`credit`)	653 (296 approvals, 357 denials)
Boston housing (`housing`)	506 (260 with income $\geq$ \$21,000, 246 else)
Cleveland heart disease (`chd`)	297 (137 disease, 160 no disease)
congressional voting (`voting`)	232 (108 democrats, 124 republicans)
Pima Indian diabetes (`pid`)	768 (268 diabetes, 500 no disease)
Wisconsin breast cancer (`wbc`)	683 (239 malignant, 444 benign)

*Class assignments are arbitrarily made.

For validation, 30 hold-out experiments were performed independently for each of the six datasets in Table 1. In each of the 30 experiments, we used randomly selected P% (with $P = 50$ and $P = 80$) of the dataset for training a decision rule by PNC and then used the remaining $(100 - P)$% of the dataset for the testing. In experimenting with PNC, we used up to 4 linear and quadratic discriminant functions with the number and the complexity of the individual functions varied from 1 and linear to 4 and all quadratic or until the 100% correct separation of the training data was achieved in all 30 independent experiments. We used ILOG CPLEX 9.0 [16] for solving the MILP's, and used the optimal or the best solution found in 10,000 branch-and-bound nodes to implement the PNC classifiers. To compare PNC with other well-established learning methods, we implemented the multisurface method (MSM) (e.g., [18]) and the support vector machine (SVM) (e.g., [19,21]) according to their presentations from [18] and [21,22], respectively, and used them to analyze the six datasets in the identical 30 independent hold-out experimental setting. In experimenting with SVM, we used the 60 combinations of SVM kernels and parameters summarized in Table 2 for each dataset and selected the combination yielding the best testing result.

Table 2. SVM kernels & parameters tested

Kernel	ν	λ	ρ	μ	Degree
Polynomial	1×10^a with $a \in \{0, 1, 2, 3\}$	1	0	1	2, 3
	1×10^a with $a \in \{0, 1, 2, 3\}$	1	0	-1	2, 3
	1×10^a with $a \in \{4, 5\}$	2	0	-1	2, 4, 6
	1×10^a with $a \in \{4, 5\}$	1	1	-1	2, 4, 5
	1×10^a with $a \in \{4, 5\}$	2	1	-1	2, 4, 5
Neural	1×10^a with $a \in \{4, 5\}$	2	0	-1	1
		1	1	-1	1
		2	1	-1	1
Sinusoidal	1×10^a with $a \in \{1, 2, 3, 4, 5\}$	$\frac{50}{\pi}$	2π	1	2, 3, 4
Gaussian radial	1×10^a with $a \in \{1, 2, 3, 4, 5\}$	n/a	n/a	1	n/a

n/a: Not applicable.

Table 3. Comparison of testing results by PNC, MSM, SVM and other approaches from the literature

Database	P^*	Testing Performance† by					Best from Literature		
		MSM	SVM	Parameter§	PNC	Form‡	P^*	Accuracy†	Ref.
credit	50	77.8±3.8	81.3±2.5	$s(2, 1)$	**86.0±1.7**	1	71	85.5±?	[3]
	80	77.6±4.5	84.5±3.2	$s(2, 1)$	**85.8±2.5**	1	80	85.5±2.6	[23]
housing	50	83.0±2.9	81.9±2.0	$s(2, 1)$	**86.1±2.0**	$2, a$	80	85.2±3.0	[23]
	80	83.1±3.7	84.2±3.3	$s(2, 2)$	**87.7±2.6**	$3, a$	80	83.2±3.1	[24]
chd	50	76.1±2.7	76.7±3.0	$s(2, 1)$	**81.8±2.2**	1	80	83.8±5.2	[23]
	80	77.0±5.2	78.7±4.6	$s(2, 2)$	**83.4±3.4**	1	?	80.6±3.1	[25]
voting	50	92.0±2.2	94.3±1.7	$p(2, 1)$	**94.5±1.8**	1	80	96.6±1.8	[23]
	80	93.2±3.7	**96.1±3.1**	$s(4, 1)$	93.9±3.7	1	67	95.6±?	[26]
pid	50	68.2±2.7	64.0±1.3	$s(2, 1)$	**76.5±1.6**	1	75	76±?	[27]
	80	67.5±3.6	65.5±2.8	$s(2, 1)$	**77.4±3.3**	$2, b$	80	72.3±2.4	[23]
wbc	50	94.9±1.1	**97.0±0.8**	$s(4, 1)$	96.3±0.8	1	80	97.2±1.3	[23]
	80	94.2±2.1	**96.7±1.3**	$s(3, 1)$	96.0±1.5	$2, c$	80	96.2±0.3	[24]

MSM, SVM & PNC results are obtained in identical experimental setting.

*: %-age of dataset used for training.

†: Correct classification %-age (in average ± 1 standard deviation)

§: SVM kernel & parameter yielding best testing results (refer to Table 2):
- $s(d, n)$: sinusoidal, degree d, $\nu = 1 \times 10^n, \lambda = \frac{50}{\pi}, \rho = 2\pi$ & $\mu = 1$
- $p(l, r)$: polynomial, degree 2, $\nu = 1 \times 10^4, \lambda = l, \rho = r$ & $\mu = -1$

‡: PNC complexity & parameter yielding best testing results:
- 1: composed of a single hyperplane
- 2: composed of two hyperplanes
- 3: composed of four hyperplanes
- a: trained with 'income < \$21K' as • class
- b: trained with 'disease' as • class
- c: trained with 'malignant' as • class

?: Not reported.

An enormous amount of results were produced from these experiments and we summarized the testing performances by MSM, SVM and PNC in Table 3 in the format the 'average ± 1 standard deviation' of the 30 correct prediction rates. To aid in comparison, we highlighted the best of the prediction rates by the three learning methodologies in this table. For reference, we also provided, to the best of our knowledge, the two best predictions rates for the datasets from the literature.

Recall that MSM, SVM and PNC are all tested in the identical hold-out experimental setting. Therefore, the comparison of classification results by the three learning methodologies in Table 3 clearly demonstrates that PNC, the proposed MILP-based classification of data, is an effective supervised learning methodology that compares favorably with the two popular methods for supervised learning.

Experimental results in the literature are often produced in different experimental settings or reported without specifying important parameters of the experiments. In some experiments, furthermore, heuristic tools are used for handling noisy data to avoid overfitting and improve testing results (e.g., [23].) Although the difference in experiments like the aforementioned does not allow a rigorous comparative statement to be made, the comparison of PNC results with the previous best in Table 3 illustrates that the MILP-based classification methodology is competitive with well-established learning methods.

4 Conclusion

This paper presented a general-purpose MILP classification model for separation of data via multiple nonlinear and nonconvex discriminant functions. The MILP model is meritorious in that it can exploit powerful MILP solvers available nowadays to concurrently optimize the parameters of discriminant functions to implement a highly nonlinear and nonconvex decision boundary for an optimal separation of data under analysis. Extensive experiments with benchmark machine learning datasets demonstrated the efficacy and the usefulness of the new MILP-based classification of data.

References

1. Lee, Y.J., Mangasarian, O., Wolberg, W.: Breast cancer survival and chemotherapy: A support vector machine analysis. In Du, D., Pardalos, P., Wang, J., eds.: Discrete Mathematical Problems with Medical Applications. Volume 55 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematics Society (2000) 1–20
2. Mangasarian, O., Wolberg, W.: Cancer diagnosis via linear programming. SIAM Review **23**(5) (1990) 1–18
3. Carter, C., Catlett, S.: Assessing credit card applications using machine learning. IEEE Expert (1987) 71–79
4. Apté, C., Weiss, S., Grout, G.: Predicting defects in disk drive manufacturing: A case study in high-dimensional classification. In: Proceedings of the 9th Conference on Artificial Intelligence for Applications, Orlando, Florida (1993) 212–218

5. Osuna, E., Freund, R., Girosi, F.: Training support vector machines: an application to face detection. In: IEEE Conference on Computer Vision and Pattern Recognition, Puerto Rico (1997) 130–136
6. Bhandari, I., Colet, E., Parker, J., Pines, Z., Pratap, R., Ramanujam, K.: Advanced scout: Data mining and knowledge discovery in nba. Data Mining and Knowledge Discovery **1** (1997) 121–125
7. Cortes, C., Vapnik, V.: Support vector networks. Machine Learning **20** (1995) 273–297
8. Megiddo, N.: On the complexity of polyhedral separability. Discrete and Computational Geometry **3** (1988) 325–337
9. Bennett, K., Mangasarian, O.: Robust linear programming discrimination of two linearly inseparable sets. Optimization Methods and Software **1** (1992) 23–34
10. Falk, J., Lopez-Cardona, E.: The surgical separation of sets. Journal of Global Optimization **11** (1997) 433–462
11. Bennett, K., Mangasarian, O.: Bilinear separation of two sets in $n-$space. Computational Optimization and Applications **2** (1994) 207–227
12. Al-Khayyal, F., Falk, J.: Jointly constrained biconvex programming. Mathematics of Operations Research **8**(2) (1983) 273–286
13. Bennett, K.: Global tree optimization: A non-greedy decision tree algorithm. Computing Sciences and Statistics **26** (1994) 156–160
14. Duda, R., Fossum, H.: Pattern classification by iteratively determined linear and piecewise linear discriminant functions. IEEE Transactions on Electronic Computers **EC-15** (1966) 220–232
15. Wolberg, W., Mangasarian, O.: Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Proceedings of the National Academy of Sciences **87** (1990) 9193–9196
16. ILOG CPLEX Division: CPLEX 9.0 User's Manual, Incline, Nevada. (2003)
17. Murphy, P., Aha, D.: Uci repository of machine learning databases: Readable data repository. Department of Computer Science, University of California at Irvine, CA (1994) Available from World Wide Web: http://www.ics.uci.edu/˜mlearn/MLRepository.html.
18. Mangasarian, O.: Multisurface method of pattern separation. IEEE Transactions on Information Theory **14**(6) (1968) 801–807
19. Vapnik, V.: The Nature of Statistical Learning Theory. 2nd edn. Springer-Verlag (2000)
20. Ryoo, H., Sahinidis, N.: Analysis of bounds for multilinear functions. Journal of Global Optimization **19**(4) (2001) 403–424
21. Mangasarian, O.: Generalized support vector machines. In Smola, A., Bartlet, P., Schölkopf, B., eds.: Advances in Large Margin Classifiers. MIT Press (2000) 135–146
22. Mangasarian, O., Musicant, D.: Data discrimination via nonlinear generalized support machines. In Ferris, M., Mangasarian, O., Pang, J.S., eds.: Complementarity: Applications, Algorithms and Extensions. Kluwer Academic Publishers (2000)
23. Boros, E., Hammer, P., Ibaraki, T., Kogan, A., Mayoraz, E., Muchnik, I.: An implementation of logical analysis of data. IEEE Transactions on Knowledge and Data Engineering **12** (2000) 292–306
24. Murthy, S., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Journal of Artificial Intelligence Research **2** (1994) 1–32
25. Shavlik, J., Mooney, R., Towell, G.: Symbolic and neural learning algorithms: an experimental comparison. Machine Learning **6** (1991) 111–143

26. Holte, R.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **11** (1993) 63–91
27. Smith, J., Evelhart, J., Dickinson, W., Knowler, W., Johannes, R.: Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. Proceedings of the Twelfth Symposium on Computer Applications and Medical Care (1988) 261–265

The Undecidability of the Generalized Collatz Problem

Stuart A. Kurtz and Janos Simon

Department of Computer Science
The University of Chicago

Abstract. The Collatz problem, widely known as the $3x + 1$ problem, asks whether or not a certain simple iterative process halts on all inputs. In this paper, we build on work of J. H. Conway to show that a natural generalization of the Collatz problem is Π_2^0 complete.

1 Introduction

Let ω denote the set $\{0, 1, \ldots\}$ of natural numbers. Let ω^+ denote the set $\{1, 2, 3, \ldots\}$ of positive natural numbers. Define the function $k : \omega \to \omega$ as:

$$k(x) = \begin{cases} x/2, & \text{if } x \text{ is even;} \\ 3x + 1, & \text{if } x \text{ is odd.} \end{cases}$$

Let $k^{(i)}$ denote the i-th iterate of k, i.e.,

$$k^{(0)}(x) = x$$
$$k^{(i+1)}(x) = k(k^{(i)}(x)).$$

Problem 1 (The Collatz Problem). For all $x \in \omega^+$, does there exist an $i \in \omega$ such that $k^{(i)}(x) = 1$?

Because of its tantalizingly elementary form, and our inability to settle it, the Collatz problem has received substantial attention. Collatz started working on the problem in 1928, but, since he felt he made little progress, only published a history of its origin in 1986 [1]. There is a very extensive literature on the many attempts to settle the conjecture, as well as related questions, using an arsenal of technologies from Number Theory, to Dynamical Systems, and Markov Chains, including a 47-page annotated bibliography [2], excellent surveys [3], even a monograph dedicated to the Dynamical Systems generalization [4]. Erdös offered $500.00 for a solution. As of this writing, the Collatz problem remains open, and mathematicians continue their attacks to try to settle it, and the diverse related questions inspired by it [5].

We provide a heuristic explanation for the apparent difficulty of the problem. We consider a generalization, defined by Conway [6], [7].

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 542–553, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Definition 1 (Generalized Collatz Functions). *A function g is called a* Collatz function *if there is an integer m together with non-negative rational numbers $\{a_i, b_i : i < m\}$, such that whenever $x \equiv i \bmod m$, then $g(x) = a_i x + b_i$ is integral. Let k_e be a Gödel numbering of the Collatz functions.*

This is a natural generalization: the function $k(x)$ of the Collatz Problem has this form for $m = 2$, $a_0 = \frac{1}{2}$, $b_0 = 0$, $a_1 = 3$, and $b_1 = 1$. The corresponding problem is:

Problem 2 (GCP: The Generalized Collatz Problem). Given a representation of a Collatz function g, can it can be decided whether for all $x \in \omega^+$ there exists an $i \in \omega$ such that $g^{(i)}(x) = 1$?

To simplify our presentation, we will choose a specific acceptable numbering of the computable partial functions—counter machines—so the e-th computable partial function φ_e is identified with the e-th counter machine M_e subsequently. Counter machines are discussed in detail in Sec. 2.

We build on beautiful results of Conway, who proved

Theorem 1 ([7]). *From the index of a computable partial function M_e, one can compute a representation of a Collatz function g_e, such that $M_e(x)$ converges iff there exists an i such that $g_e^{(i)}(2^{x+1}) = 1$.*

Conway's result already implies an undecidability result for specific Collatz functions, if not the GCP itself: There exists a Collatz function g (e.g., the function that arises from transforming a counter machine that computes the halting problem) for which the problem, given x to determine whether there exists an i such that $g^{(i)}(x) = 1$, is undecidable. Conway's result also implies the Π_2^0 completeness of the problem, given e, of whether for all $x > 0$ there exists an i such that $k_e^{(i)}(2^{x+1}) = 1$.

We improve Conway's result by eliminating the technical restriction on the relevant part of the domain of k_e.

Theorem 2. *From the index of computable partial function M_e, one can compute a representation of a Collatz function g_e such M_e is total iff for every x there exists an i such that $g_e^{(i)}(x) = 1$.*

In the previous definition, and in the sequel, we use standard concepts and notations from Computability Theory–see for example [8]. Our reduction of $\mathrm{TOT} = \{e : M_e \text{ is total}\}$ to GCP classifies precisely the complexity of GCP:

Theorem 3. *GCP is Π_2^0-complete.*

The strengthening of Conway's results that we needed to do is substantial. Section 3 has a detailed discussion of the difficulties to overcome, and our strategies for doing so. In short, Conway simulates computation steps of M_e via the application of g_e to an integer that encodes a configuration M_e. The initial configuration of M_e on input x is encoded as 2^{x+1}.

The problem is that the action of Conway's g_e on integers not of the form 2^{x+1} has nothing to do with the totality of M_e. Arbitrary integers may not encode valid configurations of M_e, and even if they do, those configurations may not be reachable from a valid starting configuration.

We use two different strategies to overcome these difficulties: first, we devise a transformation that maps a counting machine M into a machine $U(M)$, where $U(M)$ computes the same partial function as M, but has the additional property that if M is total, then $U(M)$ halts from any configuration. We call these *universally total* machines. An elegant coding trick is our second technique. It allows us to ensure termination on invalid codewords. These techniques are detailed in Sec. 3.

The next section, Sec. 2, presents Conway's technique. Section 3 contains our contribution.

2 Conway's Reduction

In this section, we provide our presentation of Conway's reduction from counter machines to GCP.

Conway defines two separate reductions, the first from counter machines to FracTran (a novel and amusing model of computation), and the second from FracTran to GCP.

A *counter machine* is a simple mathematical model of a digital computer introduced by Minsky [9]. Counter machines are equipped with finitely many counters, each of which can contain any natural number. Control is handled by a finite sequence of instructions. There are three kinds of instructions: increment a counter, decrement a counter, and halt.

An increment instruction indicates which counter is to be incremented, and the index of the index of next instruction to be executed. A decrement instruction indicates which counter is to be decremented, and two indices, one which will be the next instruction to be executed if the decrement succeeded, i.e., if the counter to be decremented was originally nonzero, and the other which specifies the next instruction if the decrement failed, i.e., if the counter was originally zero.

Each counter machine M defines a partial function ψ_M, where $\psi_M(i) = j$ if and only if when i is placed in counter 0, and the machine is started at instruction 0, it eventually halts with j in counter 0. No confusion should arise from identifying M with the partial function it computes (ψ_M), and we will do so.

Minsky proved the following two theorems:

Theorem 4. *(The Acceptability of Counter Machines) Let TM_e denote the e-th Turing machine. There exists a recursive function r such that for all natural numbers e, $TM_e = M_{r(e)}$.*

Theorem 5. *(Two Counters Suffice) It suffices to consider counter machines having only two counters, i.e., every counter machine can be effectively converted into a counter machine computing the same partial function which has only two counters.*

We will not use Theorem 5 in this paper, although it would slightly simply our presentation of Conway's reduction.

A FracTran program is a finite sequence of non-negative rational numbers $f = \{q_0, q_1, ..., q_{n-1}\}$. The transition function F_f defined by the FracTran program f maps a natural number s to $q_i s$, where i is minimal such that $q_i s$ is integral. If no such i exists, then s is a *halting state* for f, and we will establish the convention that $F_f(s) = s$.

The FracTran program f computes a partial function $F_f(x)$ by starting with state $s_0 = 2^{x+1}$, and halting in state $3^{F_f(x)+1}$. Note that if f halts in a state that is not a power of 3, by definition, F_f is not in a halting state, and is deemed to have diverged. [Note that the partial function F_f is quite different from the (total) function f, and this prevents us from identifying f and F_f in the same way we identified M and ψ_M earlier.]

2.1 Reducing a Counter Machine to a FracTran Program

We begin by making a few technical modifications to Minsky's definition of a counter machine. They do not affect the power of the underlying machine, or the acceptability of its enumeration.

- We assume that counter machines take their input in counter 0, but produce output in counter 1.
- We assume that no instruction can branch to itself.
- We assume that the last instruction of M is a halting instruction, and moreover it is the only halting instruction of M.
- We assume that just before M enters its halting state, it zeros all its counters, except for the output counter 1.

It is easy to ensure these assumptions via a reduction from Minsky's counter machines to our restricted counter machines.[1]

Note also that in this paper all sequences start with the 0-th element.

The key to the reduction is a simple process for encoding the state of a counter machine M as a natural number. Assume that M uses k counters, and it has

[1] Here is a quick sketch of how to do this. First, add no-op and unrestricted go-to statements to the instruction set of M. For every instruction that is the target of a branch, add a preceding no-op, and point the branch to the no-op. Add a "clean-up routine" to the end of M's instructions, which zeros all counters except for counter 0, then moves the contents of counter 0 to counter 1, and halts. Replace all of the original halt instructions by a go-to instruction that targets the clean-up routine. Finally, we get rid of the new instructions, first by replacing no-op's with "go-to the next instruction," and then replacing go-to's with an increment followed by a decrement, adjusting branch labels as needed.

n instructions. Let p_i denote the i-th prime. We will use the first k primes to encode the contents of the k counters. (This explains why we want our machines to take input in counter 0, which corresponds to $p_0 = 2$, and to produce output in counter 1, which corresponds to $p_1 = 3$.) For each instruction i of M, we assign the prime p_{k+i}.

For example, let's suppose we have a three counter machine, with counters 0, 1 and 2 containing the values of 2, 0, and 1 respectively, and that it is currently about to execute its fourth instruction. This would be represented by the natural number $2^2 3^0 5^1 7^0 11^0 13^0 17^0 19^1 = 380$, where $19 = p_7 = p_{3+4}$.

We can describe the state transitions of M in by multiplying the integer encoding the state by an appropriate fraction. For example, assume that instruction 4 of the machine is to increment counter 1, and branch to instruction 3. We can encode this action via the fraction $\frac{p_1 p_{3+3}}{p_{3+4}} = \frac{3 \cdot 17}{19} = \frac{51}{19}$.

Encoding a decrement instruction is a bit trickier. For example, let us assume that instruction 3 is "decrement counter 0, branching to instruction 1 on success, and instruction 2 on failure". We encode this instruction by two fractions, the first $\frac{p_{3+1}}{p_{3+3} p_0} = \frac{11}{17 \cdot 2} = \frac{11}{34}$ representing a successful decrement, and the second $\frac{p_{3+2}}{p_{3+3}} = \frac{13}{17}$ representing failure. Note that the ordering of the FracTran fractions ensure that the failure step will be taken only if the success step can't be taken, i.e., only if counter 0 contains zero, and cannot be decremented.

The restriction that no instruction can branch to itself has the effect of preventing unintended cancellations in the fractions used in encoding the instructions of M.

Note how this approach essentially encodes finite tuples of natural numbers into a single integer, and it uses multiplication by fractions to effect multiple simultaneous increment/decrement actions on those tuples.

It should be clear now that, given a counter machine M with k counters and n instructions, we can effectively construct a FracTran program f, such that if $M(x)$ converges, and f started in state $2^x p_{k+0}$, it will halt in state $3^{M(x)} p_{k+n-1}$ where instruction $n-1$ is the (necessarily unique) halting instruction of M (recall that all indexing starts with 0). Note that our final assumption is used here ensure that we don't have "spurious" powers of primes corresponding to non-output counters. If $M(x)$ does not converge, then the FracTran program f will not halt.

This is (modulo the occurences of p_{k+0} and p_{k+n-1}) the claimed reduction from counter machines to FracTran. To deal with these, we append two additional fractions to the FracTran program f. The first fraction is $p_{k+0}/2$. This will be used by the FracTran program only in an initial step, and it has the effect of discarding the "excess" power of 2, while introducing the encoding of the start state. The second is $\frac{p_1}{p_{k+n-1}} = \frac{3}{p_{k+n-1}}$, which has the effect of "stripping off" the encoding of the halting state and introducing the required "excess" power of 3. This completes the reduction from counter machines to FracTran.

2.2 Reducing FracTran to GCP

Let $f = \{q_0, q_1, \cdots, q_{n-1}\}$ be a FracTran program. It turns out that the transition function of f *is* a Collatz function k_f.

Let m be the least common multiple of the denominators of the q_i's. We will use m as the modulus for the Collatz function that we construct. We start by letting all of the a_i's be undefined, and defining all of the b_i's to be zero.

Consider each fraction q_i in order of increasing i. Let d_i be the denominator of q_i. For each residue r mod m, if r is congruent to 0 mod d_i, and a_r is undefined, define $a_r = q_i$. Finally, define $a_r = 1$ for any a_r that remain undefined after all of the q_i's have been considered.

We claim k_f has the same action on ω as f. To that end, let x be given, and let q_i be the first fraction in f such that $q_i x$ is integral. Clearly x is congruent to zero mod d_i, because the numerator n_i of q_i must be relatively prime to d_i. Also, it is clear that x cannot be congruent to zero mod d_j for any $j < i$ without contradicting the choice of j. Let x' be the congruence class of x mod m. By the foregoing, we must have defined $a_{x'} = q_i$, and $b_{x'} = 0$. Therefore $k_f(x) = q_{x'}x = f(x)$ as required.

Alternatively, $q_i x$ is not integral for any i. In this case, we must not have set $a_{x'}$ during the stages in the construction of k_f that corresponded to consideration of the q_i's, and therefore $a_{x'} = 1$, $b_{x'} = 0$, and we have $k_f(x) = 1 \cdot x + 0 = x = f(x)$ as required.

Unfortunately this does not yet do what we want.

The resulting Collatz function has the following property: $F_f(x)$ is defined iff there exists an i such that $k_f^{(i)}(2^{x+1}) = 3^{z+1}$ for some z. What we want is a Collatz function k such that $F_f(x)$ is defined iff there exists an i such that $k_f^{(i)}(2^{x+1}) = 1$. To achieve this, we modify k_f so that $a_r = 1/3$ for $r = 3^k$ mod m for all $k > 0$, in effect throwing out our output and remembering only that we have converged.

A separate concern is that we will have an "inadvertent accept," i.e., that for some x and i, our Collatz function k will have the property that $k^{(i)}(x) = 1$, even though there is no i' such that $k^{(i')}(x) = 3^{z+1}$ for any $z \in \omega$. This is a legitimate issue, even though it is easily seen that this cannot happen for FracTran programs that arise via reduction from counter machines. In this case, the repair is quite simple. Let p be a prime that does not divide m. Let $m' = mp$. Construct a Collatz function so that it works modulo m', but start the construction by defining $a_r = p$ for all r which are not divisible by p, except for $r = 1$. Finally, add rules such that $a_r = 1/r$ if $r = 3^{z+1}p$ for some z. In effect, we are using p as a "safety prime." This safety prime will be applied to all input states 2^{x+1}, and will only be stripped from inputs that correspond to output states of the form $3^{z+1}p$.

This concludes the reduction of FracTran to GCP.

3 Improvements to Conway's Reduction

In the previous section we showed how to obtain, for every counter machine M a GCP k_M that for inputs 2^{x+1} reaches 1 iff $M(x)$ halts. However, we want to

have the much stronger property that k_M reaches 1 for *every* input y. As we pointed out in the Introduction, this is difficult for two reasons:

Unreachable Configurations. Some y may represent configurations of the counter machine M, but these configurations do not arise with any computation of M starting with any input y. Such configurations have arbitrary values ("garbage") in the counters. There is no reason to believe that there is any relationship between the halting of M when started from a legal configuration and its halting when started in one of these states.

Invalid Representations. Some y may not be a valid representation of any M. The behavior of k_M on such y is not related to the reductions of the previous section.

We solve each of these problems separately. The first is taken care of by manipulating the counter machines, the second by appropriate manipulation of the GCP.

In Sec. 3.1 we define the property of *universal totality* for counter machines, as the property that started in any configuration they eventually halt. We then show how to obtain for each total counter machine M, a new machine $U(M)$ that computes the same partial function, and if M is total, $U(M)$ is universally total.

The second difficulty can be overcome with a different technique. A careful look at the Conway encoding shows that incorrect encodings are essentially integers with factors that are prime powers of "illegal" primes—primes not used in the encoding of the counter machine. Slick modular arithmetic can eliminate these: details can be found in Sec. 3.2.

3.1 Universal Totality

Definition 2. *A counter machine M is* universally total *if M halts from all of its configurations.*

The key result of this section is that universal totality is, in a uniform sense, an achievable consequence of totality.

Definition 3. *There exists a computable reduction U from counter machines to counter machines such that for all M, M and $U(M)$ compute the same partial functions. Moreover, if M is total, then $U(M)$ is universally total.*

Our approach is to simulate M in a way that involves considerable introspection.

It is useful to think of the machine $U(M)$ that we will construct somewhat more abstractly. We will think of configurations of $U(M)$ as consisting of 7-tuples of the form (a, b, c, A, B, C, x) where a, b, and c represent step counts; A, B, and C represent configurations of independent copies of M, including, for each copy, both the contents of the counters and an instruction counter which holds the index of the pending instruction; x represents an input. Note that these 7-tuples actually map to $3k + 7$ counters, assuming that M had k counters.

It is also useful to think of $U(M)$ as starting in a particular state (described in terms of a 7-tuple), and as making "big steps" which are also described in terms of the 7-tuples above. These big steps correspond to comparisons and single steps of its three copies of M. The big steps are implemented by runtime subroutines, which consist of many low-level counter machine instructions, a.k.a. "small steps." Thus, our analysis of universal totality works at two levels—we must show that $U(M)$ makes only finitely many big steps, and that each of the runtime subroutines that implement a big step of $U(M)$ is guaranteed to take only finitely many small steps (from any configuration). These runtime subroutines may make use of a finite number of additional "scratch" counters. We will ensure that $U(M)$'s big steps are oblivious to the state of the scratch counters, and that our runtime subroutines zero out the scratch counters they use on entry.

We simplify our simulation by stipulating that we use counter 0 to hold x, and counter 1 to hold the output register of C.

We start $U(M)$ in configuration $(0, 0, 0, S_M(x), S_M(x), S_M(x), x)$, where $S_M(x)$ is the starting configuration of M on input x. Note that the runtime code required to get to this configuration amounts to moving the contents of x to a scratch counter, and then moving the scratch counter back (simultaneously) to x and the start counters of A, B, and C. This runtime code is guaranteed to exit, because counter-to-counter moves are limited by the initial contents of the source counter.

This is a guiding principle in the runtime subroutines—they can be thought of as consisting of a few (non-nested) loops, where each loop is controlled by decrementing a counter (which is not incremented in the body of that loop). We can easily see that such a routine must terminate—the loop bodies are straight-line code, and each loop will execute a number of times that is bounded by the (necessarily finite) contents of its control counter on entry.

To define the big steps of $U(M)$, let T_M denote the single step transition function of the underlying machine M. Assume $U(M)$ is in configuration (a, b, c, A, B, C, x). Our intention is that $A = T_M^{(a)}(S_M(x))$, i.e., that A is the configuration of M after a steps, starting from input x, and likewise for B and b, and C and c. Similarly, each of the three copes of M in $U(M)$ (which we will refer to as the A, B, and C copies) has an intended role. Copy C is our "main copy." Copy A is intended to be a checkpointed copy of C—it should hold the configuration of C at an earlier time step. Copy B represents a restart of M, which is used to verify the checkpoint.

We define various types of "big steps" of $U(M)$ as follows. Assume $U(M)$ is in configuration (a, b, c, A, B, C, x).

- If C is in a halting configuration, we halt.
- Our intentions regarding the various elements of the configuration of $U(M)$ imply $b \leq a \leq c$ if we are in a reachable configuration. If this is not the case, we know that we are not in a reachable configuration, and we halt.

- If $b < a$, we make a *basic* big step to $(a, b+1, c+1, A, T_M(B), T_M(C), x)$.
- If $b = a$, we verify that $B = A$. If verification fails, we know that we are not in a reachable configuration, and halt. If the verification succeeds, we make a *reflection* big step to $(c, 0, c+1, C, S_M(x), T_M(C), x)$.

There is some trickiness in writing the runtime subroutines that implement the big steps of $U(M)$. The equality and inequality checks are relatively straightforward—we decrement two counters together, while incrementing a scratch counter that we can use to restore the original states of the counters that are being compared. Dispatching instructions to an individual copy of M is a bit trickier. Basically, we decrement our way through that copy's instruction counter, using our state to hold the (bounded) number of decrements. When we hit zero, we branch to a copy of the instruction we are simulating, followed by a sequence of increment statements that properly set the value of the instruction counter for the next step. While the dispatch code is cumbersome, its termination analysis is trivial because it is forward-branching.

Because our small step runtime subroutines are guaranteed to terminate, our analysis of $U(M)$ can focus on its big step behavior.

If we start $U(M)$ with input x, we set up the big step start configuration $(0, 0, 0, S_M(x), S_M(x), S_M(x), x)$. It is easily seen that big step start configuration established the invariants that

- A is the configuration M on input x after a steps (respectively, B and b, and C and c), and
- $b \leq a \leq c$.

and that these invariants are preserved by basic and reflection big steps. Each basic or reflection big step incorporates making a step with copy C. If $M(x)$ terminates in c steps, then $U(M)(x)$ will terminate after c basic + reflection steps. Because the C copy of M uses counter 1 as its output counter, if $U(M)(x)$ will terminate with the same output as $M(x)$, and therefore the two machines have the same I/O behavior, i.e., they implement the same partial functions.

The crucial additional property we intend for $U(M)$ is that it is universally total if M is total. To this end, let's assume that we start $U(M)$ in an unreachable state. If we start in a small step subroutine, we know by our prior analysis that it must exist, resulting in a big step configuration (a, b, c, A, B, C, x). If M eventually halts in configuration C, then $U(M)$ must also halt by our earlier analysis. Therefore, assume that M does not halt from configuration C. We must show that $U(M)$ halts anyway.

We can assume that $b \leq a \leq c$, otherwise $U(M)$ halts immediately. After $a - b$ basic steps, $U(M)$ will be ready to execute a reflection big step. At this point, we might detect an inconsistency in the form $A \neq B$, in which case we would halt. Therefore, we can assume that no inconsistency is detected. The reflection step will take us from a state of the form (a, a, c, A, A, C, x) to $(c, 0, c+1, C, S_M(x), T_M(C), x)$. At this point, we've established the invariant that $B = S_M(x)$ holds the configuration of M on input x after $b = 0$ steps, which we preserve through subsequent basic and reflection steps. On our assumption

that the original state was unreachable, we know that $A = C$ does *not* hold the state of M after $a = c$ steps. After $a = c$ additional basic steps, we will encounter a second reflection step. At this point we will discover $A \neq B$, and halt.

3.2 Unintended Factors

The final issue involved in giving a completely satisfactory reduction of TOT to GCP is that not all integers encode valid states of M'. What is required is something like a universal totality result for GCP. This is not so easy to obtain!

Instead, it suffices to focus on the particular Collatz functions that are produced by the reductions from (possibly universally total) counter machines. These Collatz functions all work mod m, where m is a product of distinct primes, and were intended to manipulate integers that took one of following forms:

- Integers of the form 2^{x+1}, i.e., initial input states.
- Integers representing a state of M. These integers would have prime factorizations that consisted of powers of a few small primes (corresponding to the contents of counters), and exactly one other prime (corresponding to the state of M).
- Integers of the form 3^{k+1}, i.e., final output states, in the process of being reduced to 1.

If we have integers of these forms, the Collatz functions produced by modifying Conway's transformation works. The problem comes from integers that have other forms.

Now the Collatz function k is working modulo m. Some of the congruence classes correspond to numbers of the preceding form, other do not. Suppose r is a congruence class that does not contain an integer of one of the three forms above. Then we set $a_r = \frac{1}{m}$, and $b_r = \frac{m-r}{m}$. This rule has the effect of mapping an "impossible" residue to the next largest multiple of m, and then casting out the factor of m. Suppose $x = r \bmod m$. The result of this rule is that $k(x) < x$, unless $x = 1$, in which case $k(1) = 1$.

Another possible problem is that we are working with a number that has the form xz, where x takes one of the three forms above, and $z \neq 1$, but $z = 1 \bmod m$. In this case, if our original machine was total, our Collatz function arose from reduction from a universally total machine. In this case, after finitely many simulation steps, the Collatz function will reduce xz to z. In this case, our rule for dealing with the residue of 1 from the preceeding paragraph applies. The result of this step either an impossible number, or a number of the form $x'z'$, where x' and z' as as x and z were before, but $z' < z$.

As in the preceeding section, a "big step," "little step" analysis demonstrates totality (assuming the original machine M was total). Our "big steps" will be impossible residue steps, and the steps from xz to $x'z'$ as above. We see that each big step may involve a number of little steps (corresponding to individual iterations of the Collatz function), but this number must be finite (because we are simulating a universally total counter machine). As we work through the big steps, the state number we are dealing with is strictly decreasing, and therefore there can be only finitely many big steps.

3.3 Our Main Result

Our main theorem, that GCP is Π_2^0-complete follows from putting all previous techniques together. More precisely, we show that TOT reduces to GCP as follows:

- start with a counter machine M_e,
- transform it into a counter machine M'_e of the special form indicated in Section 2
- apply the transformation described in Section 3.1 to get $U(M'_e)$, which will be universally total if M'_e is total
- use the Conway transformations of Section 2 to obtain a GCP $k_{U(M'_e)}$
- modify this GCP using the techniques of Section 3.2, obtaining the final GCP $k'_{U(M'_e)}$

After these transformations, the GCP $k'_{U(M'_e)}$ has the property that for all y, $k'_{U(M'_e)}(y)$ reaches 1 iff M_e is total. This establishes the Π_2^0 hardness of GCP.

On the other hand, the statement "$\forall y\ \exists i.\ {k'_{U(M'_e)}}^{(i)}(y) = 1$" is manifestly Π_2^0, which establishes that GCP is Π_2^0. Thus, GCP is Π_2^0 complete, concluding the proof of the main theorem.

As suggested above, the strong undecidability of the generalized problem is a good heuristic explanation for the initially unexpected hardness of the Collatz problem. We believe that it is interesting in its own right.

Acknowledgements

The authors gratefully acknowledge Carl Jockusch, Jr., whose elegant presentation of J. H. Conway's work interested us in this question, and who sustained us with encouragement and many thoughtful comments, and Warren Goldfarb, whose timely questions encouraged us that there was more general interest in this question.

References

1. Collatz, L.: On the origin of the (3n+1) problem. Journal of Qufu Normal University, Natural Science Edition **12**(3) (1986) 9–11
2. Lagarias, J.C.: The 3x+1 problem: An annotated bibliography (1963–2000). ArXiv math (NT0608208) (2006)
3. Lagarias, J.C.: The 3x+1 problem and its generalizations. American Mathematics Monthly **92**(1) (1985) 3–23
4. Wirsching, G.J.: The Dynamical System Generated by the 3n+1 Function. Volume 1681 of Lecture Notes in Mathematics. Springer Verlag, Berlin (1981)
5. Matthews, K.R.: The generalized 3x + 1 mapping (2006)
6. Conway, J.H.: Unpredictable iterations. In: 1972 Number Theory Conference, University of Colorado, Boulder (1972) 49–52

7. Conway, J.H.: Fractran, a simple universal computing language for arithmetic. In Clover, T.M., Gopinath, B., eds.: Open Problems in Communication and Computation. Springer Verlag (1987) 3–27
8. Rogers, H.: Theory of Recursive Functions and Effective Computability. MIT Press, Cambridge, MA (1987)
9. Minsky, M.L.: Recursive unsolvability of Post's problem of "tag" and other topics in the theory of Turing machines. Annals of Mathematics **74**(3) (November 1961) 437–455

Combinatorial and Spectral Aspects of Nearest Neighbor Graphs in Doubling Dimensional and Nearly-Euclidean Spaces

Yingchao Zhao[1,⋆] and Shang-Hua Teng[2,⋆⋆]

[1] Department of Computer Science
Tsinghua University
yczhao@mails.tsinghua.edu.cn
[2] Department of Computer Science
Boston University
steng@cs.bu.edu

Abstract. Miller, Teng, Thurston, and Vavasis proved that every k-nearest neighbor graph (k-NNG) in $\mathbb{R}^d$ has a balanced vertex separator of size $O(n^{1-1/d}k^{1/d})$. Later, Spielman and Teng proved that the Fiedler value — the second smallest eigenvalue of the graph — of the Laplacian matrix of a k-NNG in $\mathbb{R}^d$ is at $O(\frac{1}{n^{2/d}})$. In this paper, we extend these two results to nearest neighbor graphs in a metric space with doubling dimension γ and in nearly-Euclidean spaces. We prove that for every $l > 0$, each k-NNG in a metric space with doubling dimension γ has a vertex separator of size $O(k^2 l(32l+8)^{2\gamma}\log^2\frac{L}{S}\log n + \frac{n}{l})$, where L and S are respectively the maximum and minimum distances between any two points in P, and P is the point set that constitutes the metric space. We show how to use the singular value decomposition method to approximate a k-NNG in a nearly-Euclidean space by an Euclidean k-NNG. This approximation enables us to obtain an upper bound on the Fiedler value of the k-NNG in a nearly-Euclidean space.

Keywords: Doubling dimension, shallow minor, neighborhood system, metric embedding, Fiedler value.

1 Introduction

Graph partitioning is an important combinatorial optimization problem that is widely used in applications that include parallel processing, VLSI design, and data mining. There are several versions of this problem. Perhaps the simplest

⋆ Part of this work was done while visiting Computer Science Department at Boston University. In part supported by the National Grand Fundamental Research 973 Program of China under Grant (2004CB318108, 2004CB318110, 2003CB317007), the National Natural Science Foundation of China Grant (60553001, 60321002) and the National Basic Research Program of China Grant (2007CB807900, 2007CB807901).
⋆⋆ Part of this work was done while visiting Tsinghua University and Microsoft Research Asia Lab.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 554–565, 2007.
© Springer-Verlag Berlin Heidelberg 2007

version is to divide a graph into two equal-sized clusters and minimize the number of edges between these two clusters. In general, we may want to divide a graph into multiple clusters and minimize some objective functions such as the total number of inter-cluster edges or the maximum among the ratios defined by the number of edges leaving a cluster to the number of vertices in that cluster [9,10]. Graph partitioning is usually a hard problem if an optimal solution is desired [5]. But, because of its importance in practice, various partitioning heuristics and approximation algorithms are designed and implemented. The spectral method, which uses the eigenvectors of a graph matrix, has been among the most popular ones used in practice [1,12].

In this paper, we study combinatorial and spectral aspects relating with partitioning nearest neighbor graphs defined in Euclidean-like metric spaces. Our study is inspired by two early work on Euclidean nearest neighbor graphs. The first one is by Miller *et al* [11] who shows that every k nearest neighbor graph (k-NNG) of n points in $\mathbb{R}^d$ has a vertex separator of size $O(n^{1-1/d}k^{1/d})$ that $1/(d+2)$ splits the graph. Here, for a parameter $f : 0 < f < 1$, a vertex separator that f-splits a graph is a subset of its vertices whose removal divides the rest of the graph into at least two disconnected components such that the sizes of all components are no more than $f \cdot n$. If f is a constant, independent of n, then we refer to the vertex separator that f-splits as a *balanced* separator. The second one is by Spielman and Teng [17]. It shows that the Fiedler value — the second smallest eigenvalue of the graph — of the Laplacian matrix of a k-NNG in $\mathbb{R}^d$ is at $O(\frac{1}{n^{2/d}})$.

We first consider the k-NNG for points in a metric space of a finite doubling dimension. This family of metric spaces (see Section 2 for a formal definition) is introduced by Karger and Ruhl [8] with the motivation to extend efficient nearest-neighbor-search data structures from Euclidean spaces to other growth-constrained metric spaces arising in internet applications.

As one of the main results of this paper, we prove that for every $l > 0$, each k-NNG in a metric space with doubling dimension γ has a balanced vertex separator of size $O(k^2 l(32l+24)^{2\gamma} \log^2 \frac{L}{S} \cdot \log n + \frac{n}{l})$, where L and S are respectively the maximum and minimum distances between any two points in P. By choosing $l = n^{1/(2\gamma+2)}(k^2 \log^2 \frac{L}{S} \cdot \log n)^{-1/(2\gamma+2)}$, we prove that every k-nearest neighbor graph of n points in a metric space with doubling dimension γ has a balanced vertex separator of size

$$O\left(n^{1-1/(2\gamma+2)} k^{1/(\gamma+1)} \log^{1/(\gamma+1)}(L/S) \cdot \log^{1/(2\gamma+2)} n\right)$$

We can also show that the maximum degree of these k-NNG is at most $O(k \log(L/S))$. Thus, this separator bound also implies that the Fiedler value of a k-nearest neighbor graph of n points in a metric space with doubling dimension γ is at most

$$O\left(\frac{n^{\frac{-1}{2\gamma+2}} k^{1+\frac{1}{\gamma+1}} \log^{\frac{1}{2\gamma+2}}(L/S) \log^{\frac{1}{2\gamma+2}} n}{1 - 2n^{\frac{-1}{2\gamma+2}} k^{\frac{1}{\gamma+1}} \log^{\frac{1}{\gamma+1}}(L/S) \log^{\frac{2}{\gamma+1}} n + n^{\frac{-1}{\gamma+1}} k^{\frac{2}{\gamma+1}} \log^{\frac{2}{\gamma+1}}(L/S) \log^{\frac{1}{\gamma+1}} n}\right)$$

Key to our proof, we characterize the family of minors excluded by these nearest neighbor graphs: For any given depth t, we show that these graphs can not contain a minor of size $O(kt^{\gamma}\log(L/S))$. With this graph-theoretic property, we can use the separator theorem of Plotkin, Rao, and Smith [13] to prove our separator bound.

For each k-NNG in nearly Euclidean spaces (see Section 4 for formal definition), we can apply the singular value decomposition method to find an approximate Euclidean k-NNG. This approximation enables us to obtain a better separator and Fiedler value bound than those that can be derived from doubling-dimensional framework.

We organize our paper as following. In Section 2, we introduce the notation and definitions which will be used in the paper. In particular, we will introduce doubling dimensional spaces, nearest neighbor graphs, the Fiedler value of a graph, and Singular Value Decomposition. We will prove the separator theorem for k-NNG in a finite doubling dimensional space in Section 3. For the k-nearest neighbor graphs in nearly-Euclidean space, we discuss their spectra in section 4. Finally, we conclude our work in Section 5.

2 Graphs and Geometry

In this paper, we consider graphs that are geometric defined. We first introduce some notation and definitions that will be used later. Given a graph $G = (V, E)$, we assume V is the point set from a metric space.

2.1 Metric Spaces and Doubling Dimension

Given a set X of points and a distance function d which is defined as d: $X \times X \longrightarrow [0, \infty)$, we call the pair (X, d) a *metric* space if it satisfies the following axioms.

- $\forall x, y \in X$, $d(x, y) = 0$ iff $x = y$.
- $\forall x, y \in X$, $d(x, y) = d(y, x)$.
- $\forall x, y, z \in X$, $d(x, y) + d(y, z) \geq d(x, z)$.

If (X, d) only satisfies the last two axioms and $d(x, x) = 0$ for all $x \in X$ instead of the first item, we call it a *semimetric* (or *pseudometric*).

There are various metric spaces with different dimensions, for example, the Euclidean space and the Hamming space. Not all the problems in practice can be modeled as graphs in an Euclidean space or a Hamming space. Although these metric spaces are simple and more familiar to us, practical problems may not satisfy all those geometric terms. The doubling dimensional space, which has less constraints, is introduced by Karger and Ruhl [8] and becomes useful in several research areas, such as graph partitioning and network routing. One objective of this paper is to design efficient algorithms for graphs in a metric space with finite doubling dimension.

Denote the space within a distance r to a point $v \in X$ as a ball $B_r(v)$ where r is the radius and v is the center. The metric (X, d) has a doubling dimension γ if any ball of radius r could be covered by 2^{γ} balls of radius $\frac{r}{2}$. Euclidean

space could be considered as a special doubling dimension space. Different from general Euclidean spaces, doubling dimensional space has no such definitions as volume and parallelization. However, for those continuous doubling dimensional spaces, they could also have some useful properties, such as segment property as follows.

Definition 1. *A metric doubling dimension space (X, d) has segment property if for each pair of points $x, y \in X$, there exists a continuous curve $\gamma = \gamma(t)$ connecting x and y such that $d(\gamma(t), \gamma(s)) = |t - s|$ for all t and s.*

Segment property appears naturally in many applications and in this paper, we will mainly focus on those instances where segment property holds. For more details about segment property, please refer to [14]. We will give more properties in Section 3.

2.2 Nearest Neighbor Graphs

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in a metric space. For each $p_i \in P$, let $N_k(p_i)$ be the set of k points closest to p_i in P (if there are ties, break them arbitrarily). Let $R(p_i)$ be the distance between p_i and its k-th closest neighbor, hence $\forall p_i, p_j$, if $p_i \in N_k(p_j)$ then $\|p_i p_j\| \quad R(p_j)$. Let $B_R(p_i)$ be the ball centered at p_i with radius R, and we denote $\alpha B_R(p_i)$ to be the ball centered at p_i with radius αR. With $B_R(p_i)$, we can define k-ply systems, k-nearest neighbor graphs and intersection graphs in general metric spaces.

Definition 2. *Let $P = \{p_1, \ldots, p_n\}$ be points in a metric space, then a k-ply neighborhood system for P is a set of closed balls, $B = \{B_1, \ldots, B_n\}$, such that B_i centers at p_i and no point p in this metric space is contained in the interior of more than k balls from B.*

Definition 3. *A k-Nearest Neighbor Graph (k-NNG) of a set of n vertices is a graph with vertex set $P = \{p_1, \ldots, p_n\}$ and edge set $E = \{(p_i, p_j)$: $p_i \in N_k(p_j)$ or $p_j \in N_k(p_i)\}$, where $N_k(p_i)$ represents the set of k points closest to p_i in V. We denote the k-Nearest neighbor Graph of P as $N_k(P)$.*

Definition 4. *Given a set S, and a family of nonempty subsets of S, the corresponding intersection graph has a vertex for each subset, and a connecting edge whenever two subsets intersect.*

Definition 5. *Given a k-ply neighborhood system $\Gamma = \{B_1, B_2, \ldots, B_n\}$. The intersection graph of Γ is the undirected graph with vertices $V = \{1, \ldots, n\}$ and edges $E = \{(B_i, B_j) : (B_i \cap B_j \neq \emptyset)\}$.*

Definition 6. *Given a k-ply neighborhood system $\Gamma = \{B_1, B_2, \ldots, B_n\}$ and $\alpha \geq 1$. The α-overlap graph of Γ is the undirected graph with vertices $V = \{1, \ldots, n\}$ and edges $E = \{(B_i, B_j) : (B_i \cap \alpha \cdot B_j \neq \emptyset)$ and $(B_j \cap \alpha \cdot B_i \neq \emptyset)\}$.*

In this paper, the subsets are balls in some metric space. Therefore, we can let the vertex for each subset be the corresponding center of the ball. In this way, we can bound the degree of the intersection graph with the help of the ply bound in the original ball system.

2.3 Graph Partitioning and Vertex Separators

A *partition* of a graph $G = (V, E)$ is a division of its vertices into several specified number of subsets. Generally we focus on two objectives when we use graph partitioning : one objective is to minimize the number of the edges cut by the partition; the other one is to balance the computational load, i.e, to limit the size of each subset to within a tolerance. We call E_s, a subset of E, the *edge separator* of G, if removing E_s from E leaves two or more disconnected components of V. We call V_s, a subset of V, the *vertex separator* of G, if removing V_s and all incident edges leaves two or more disconnected components of V.

2.4 Laplacian and the Fiedler Value

Suppose $G = (V, E)$ is an undirected, connected graph, then its adjacent matrix $A(G) = (a_{ij})_{n\times n}$, where

$$a_{ij} = \begin{cases} 1 \text{ if } (i,j) \in E \\ 0 \text{ otherwise} \end{cases}$$

Let $D(G) = (d_{ij})_{n\times n}$ be a diagonal matrix where d_{ii} is the degree of the vertex v_i in the graph G. The Laplacian matrix of G is denoted as $L(G) = D(G) - A(G) = (l_{ij})_{n\times n}$. Hence

$$l_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } (i,j) \in E \\ 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ degree(v_i) & \text{if i=j} \end{cases}$$

Because $L(G)$ is real and symmetric, its eigenvalues are all non-negative and its smallest eigenvalue is zero, with $(1, \ldots, 1)^T$ being its corresponding eigenvector. Fiedler [6] associated the second smallest eigenvalue of the Laplacian matrix of the graph with its connectivity. Thus, we call the second smallest eigenvalue of $L(G)$ the *Fiedler value* and call the corresponding eigenvector the *Fiedler vector*. Because G is connected, we know that the Fiedler value is non-zero and can be expressed as following.

$$\lambda_2 = \min_{x \perp (1,\ldots,1)^T} \frac{x^T L(G) x}{x^T x} = \min_{x \perp (1,\ldots,1)^T} \frac{\sum_{(i,j)\in E} (x_i - x_j)^2}{\sum_{i=1}^{n} x_i^2}$$

From the definition, we can get the following property.

Corollary 1. *The Fiedler value of the edge subgraph is no more than the Fiedler value of the original graph.*

2.5 Singular Value Decomposition

To learn more about Laplacian matrix and its Fiedler value, we review a useful technique called *singular value decomposition* (SVD). We give its formal definition below.

Definition 7. *A singular value decomposition of an $m \times n$ matrix A with $m \geq n$ is any factorization of the form*

$$A = UDV^T = \left[u_1, u_2, \ldots, u_n\right] \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{pmatrix} \tag{1}$$

where U is an $m \times n$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and D is an $n \times n$ diagonal matrix with $s_{ij} = 0$ if $i \neq j$ and $s_{ii} = \sigma_i \geq 0$.

In SVD, the quantity σ_i is a singular value of A. Without loss of generality, we assume that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ in this paper. We usually use two norms to describe the matrix. Given a matrix $A = (a_{ij})_{m \times n}$, the *Frobenius* norm ($F$ norm) of A is defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} {a_{ij}}^2} = \sqrt{\sum_{i=1}^{n} \sigma_i^2}$$

while the *Euclidean* norm (2-norm) of A is defined as

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2 = 1} \|Ax\|_2$$

where x is an n dimensional vector and $\|x\|_2 = (x^T x)^{\frac{1}{2}} = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$.

In 1907, Erhard Schmidt [15] introduced the infinite dimensional analogue of the singular value decomposition. Eckart and Young [3,4] showed that if we replace the smallest $m - s$ singular values with zeros in D, then the new multiplication of UDV^T is the least square approximation in s dimensions of the original matrix A.

Theorem 1 (Eckart-Young). *Let the SVD of A given by (1) with rank $r = rank(A) \leq p = \min\{m, n\}$ and define*

$$A_k = U_k D_k V_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

then A_k is the optimal approximation of A in the view of

$$\min_{rank(B)=k} \|A - B\|_F = \|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{p} \sigma_i^2}$$

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

Hence we can find a proper low-rank matrix A_k to approximate the original graph and Eckart-Young Theorem guarantees that this approximation will not cause much difference. For more properties of SVD, please refer to [2] and [7].

3 A Separator Theorem for Doubling Dimensional Spaces

In this section we prove a separator theorem for k-NNG in a metric space with finite doubling dimension.

Theorem 2. *For every $l > 0$, each k-NNG in a metric space with doubling dimension γ, we can find a separator of size $O(k^2 l(32l+8)^{2\gamma} \log^2 \frac{L}{S} \log n + \frac{n}{l})$, where L and S are respectively the maximum and minimum distances between any two points in P.*

We start with the following lemma that will be useful to obtain a degree bound for k-NNG in a doubling dimensional metric space.

Lemma 1. *For any ball of radius r in metric space with doubling dimension γ, it contains at most 2^γ disjoint balls of radius $\frac{r}{2}$.*

Similarly, we can get the following corollary.

Corollary 2. *For any ball of radius r in a metric space with doubling dimension γ, it contains at most $2^{t\gamma}$ disjoint balls of radius $\frac{r}{2^t}$.*

In this section, we will show an extended version of separator theorem in doubling dimensional spaces.

3.1 Shallow Minors

Key to our analysis is to show that k-NNG in finite doubling dimensional metric space excludes certain type of minors.

Definition 8. *A minor of a graph G is a graph obtained from G by a series of edge contractions and edge deletions.*

Teng [18] showed that for those balls in k-ply neighborhood system in Euclidean space, there could not be too many balls of large radius intersecting the same ball. We can get a similar result for graphs in doubling dimensional space, i.e, for any ball of radius r, there could not exist many balls of radius at least βr which intersect it.

Lemma 2. *Suppose $\{B_1, \ldots, B_n\}$ is a k-ply neighborhood system in a metric space with doubling dimension γ. For each ball B with radius r, for all constant $\beta > 0$, we have*

$$|\{i : B_i \cap B \neq \phi \quad \text{and} \quad r_i \geq \beta r\}| \leq k \left(\frac{2(1+2\beta)}{\beta}\right)^\gamma$$

where r_i is the radius of B_i.

Although there is no such definition as volume in Euclidean space, the doubling dimensional space does have similar shallow minor properties. We'll show that the intersection graph of a k-ply neighborhood system in doubling dimensional space does exclude shallow minors of a certain size.

Because there are not many intersecting balls, the intersection graph of k-ply neighborhood system does not have a large minor either.

Theorem 3. *Suppose Γ is a k-ply neighborhood system in a metric space with doubling dimension γ and G is the intersection graph of Γ. Then $\forall l$, G excludes K_h as a depth[1] l minor for $h \geq k(8l+2)^\gamma$.*

Proof. Suppose G has a K_h minor of depth l. We claim that there must exist h sets of balls, $\Gamma_1, \ldots, \Gamma_h \subset \Gamma$, such that:

- The intersection graph of each Γ_i is connected with diameter at most l.
- For each pair $i, j \in \{1, \ldots, h\}$, there's a ball in Γ_i that intersects a ball in Γ_j.

Let B_i be the ball of the largest radius in Γ_i. Without loss of generality, assuming that B_1 is the ball of the smallest radius among $\{B_1, \ldots, B_h\}$ and its radius is r. Hence, all the balls in Γ_1 are contained in a ball $B' = (2l+1)B_1$, because the intersection graph of Γ_1 is connected. According to the second condition, $\forall i > 1$, there is a ball from Γ_i that intersects B'.

We claim that for each $i > 1$, there is a ball in Γ_i of radius at least r that intersects the ball $(4l-1)B_1$.

As we know, the diameter of the intersection graph of Γ_i is at most l and there is a ball from Γ_i that intersects B'. If that intersecting ball has radius at least r, then we are done with Γ_i. If not, we can enlarge the radius of B' by $2r$, at that time, the enlarged B' will completely contain the intersecting ball in Γ_i and meet other balls in Γ_i because of the connectivity of Γ_i. Then we judge whether one of the intersecting balls has radius at least r. If not, we repeat the augment process above. Because B_1 is the ball of the smallest radius among $\{B_1, \ldots, B_h\}$, the process will surely terminate. This process is like a breadth-first-search. The number of iterations is less than $l-1$, since we will surely meet either B_i (the maximum-radius ball in Γ_i, whose radius is at least r) or some other balls in Γ_i that has radius at least r.

Namely, the ball B^* of radius $R = (4l-1)r$ intersects h balls of radius at least βR where $\beta = 1/(4l-1)$. Applying Lemma 2, we have $h \leq k(8l+2)^\gamma$. □

Theorem 4. *Suppose Γ is a k-ply neighborhood system in a metric space with doubling dimension γ and G is the α-overlap graph of Γ. Then $\forall l$, G excludes K_h as a depth l minor for $h \geq k(8\alpha l+2)^\gamma$.*

3.2 Proof of Theorem 2

In this subsection, we give the proof of Theorem 2. First, let's bound the max degree of nearest neighbor graphs and the ply of neighborhood system in metric spaces with doubling dimension γ.

Lemma 3. *Let $P = \{p_1, \ldots, p_n\}$ be a point set in a metric space with doubling dimension γ. Then the ply of $N_k(P)$ is bounded by $k4^\gamma \log_{\frac{3}{2}} \frac{2L}{S}$, where L is the maximum distance between any two points in P, and S is the smallest one.*

[1] The maximum number of edges in each simple path.

The k-nearest neighbor graph has no more ply than $N_k(P)$, therefore we can bound the ply of k-NNG in doubling dimension space.

Corollary 3. *The ply for any k nearest neighbor graph in a metric space with doubling dimension γ is at most $k4^\gamma \log_{3/2} \frac{2L}{S}$, where L is the longest distance in the graph and S is the shortest.*

Plotkin, Rao and Smith [13] gave the following theorem and showed that we can find a small size separator for the graph which excludes shallow minors.

Theorem 5. *For any graph that excludes K_h as a depth l minor, we can find a separator of size $O(lh^2 \log n + n/l)$, where n is the number of vertices of the graph.*

Because every k-NNG in a metric space with doubling dimension γ has ply bound of $k4^\gamma \log_{3/2} \frac{2L}{S}$ and it excludes K_h minor with depth l, where $h > k4^\gamma \log_{3/2} \frac{2L}{S}(8l+2)^\gamma = k(32l+8)^\gamma \log_{\frac{3}{2}}(\frac{2L}{S})$. Applying Theorem 5 gives the separator bound of k-NNG in a metric space with doubling dimension γ. Therefore, Theorem 2 holds.

To minimize the separator size, we choose $l = n^{1/(2\gamma+2)}(k^2 \log^2 \frac{L}{S} \cdot \log n)^{-1/(2\gamma+2)}$ such that the two terms are equal and get that every k-nearest neighbor graph of n points in a metric space with doubling dimension γ has a balanced vertex separator of size

$$O(n^{1-1/(2\gamma+2)} k^{1/(\gamma+1)} \log^{1/(\gamma+1)}(L/S) \cdot \log^{1/(2\gamma+2)} n)$$

Since we have showed that the maximum degree of these k-NNG is at most $O(k \log(L/S))$, the above separator bound could also give an upper bound of the Fiedler value of a k-nearest neighbor graph of n points in a metric space with doubling dimension γ. Assign 1 to each vertex in the vertex separator and $|separator|/(|separator| - n)$ to the remaining vertices, then we have the following inequality.

$$\begin{aligned}
\lambda_2 &\leq \frac{\sum_{cut\ edge(i,j)}(x_i - x_j)^2}{\sum_{\forall i} x_i^2} \\
&\leq \frac{(\frac{n}{|separator|-n})^2 \times |separator| \times |max\ degree|}{n} \\
&= O(\frac{n^{\frac{-1}{2\gamma+2}} k^{1+\frac{1}{\gamma+1}} \log^{\frac{1}{2\gamma+2}}(L/S) \log^{\frac{1}{2\gamma+2}} n}{1 - 2n^{\frac{-1}{2\gamma+2}} k^{\frac{1}{\gamma+1}} \log^{\frac{1}{\gamma+1}}(L/S) \log^{\frac{2}{\gamma+1}} n + n^{\frac{-1}{\gamma+1}} k^{\frac{2}{\gamma+1}} \log^{\frac{2}{\gamma+1}}(L/S) \log^{\frac{1}{\gamma+1}} n})
\end{aligned}$$

4 A Spectral Theorem for Nearly-Euclidean Spaces

Since Fiedler [6] discovered that the second smallest eigenvalue is closely related to the connectivity of the graph, a large amount of work has been done on spectra analysis of graphs. In 1996, Spielman and Teng [17] proved that the Fiedler value of a k-nearest neighbor graph with n vertices in $\mathbb{R}^d$ is bounded by $O(k^{1+2/d}/n^{2/d})$.

In this session, we consider a point set P of n vertices in $\mathbb{R}^m$ space. $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^m$. We can get an $m \times n$ matrix $\mathbf{P}$ with column vectors $(p_1, \ldots, p_n)$. An upper bound of Fiedler value of the Laplacian matrix $L(\mathbf{P})$, given by Spielman and Teng, is as following.

Theorem 6. (Spielman-Teng) *If G is a subgraph of an α-overlap graph of a k-ply neighborhood system in $\mathbb{R}^m$ and the maximum degree of G is Δ, then the Fiedler value of $L(G)$ is bounded by $\gamma_m \Delta \alpha^2 (\frac{k}{n})^{2/m}$, where $\gamma_m = 2(\pi + 1 + \frac{\pi}{\alpha})^2 (\frac{A_{m+1}}{V_m})^{2/m}$.*

A_m is the surface volume of a unit m-dimensional ball, and V_m is the volume of a unit m-dimensional ball. In general case, the numbers k and α are two constants, and the item γ_m can be considered as a constant if the dimension m is fixed. Therefore, the bound can be expressed by $O(\frac{1}{n^{2/m}})$, which is dependant on the dimension of the space.

If we change the base carefully, the dimension could be changed as well. Hence we can consider the Laplacian matrix of a k-NNG and find a low-rank approximation matrix which can be contained in a lower dimension space so that the dimension of the new space is smaller. The changing of basis could make the problem easier, and we call the new space *nearly-Euclidean space*.

As we mentioned in Section 2, SVD could help us get a low-rank approximation matrix $\mathbf{Q}$ whose rank is d with $d < m$. Suppose the column vectors of $\mathbf{Q}$ is $(q_1, \ldots, q_n)$ and these n points form a new point set Q. Suppose that G' is the $(1 + \frac{7\delta}{s})$-overlap graph of the k-NNG of Q, the maximum degree of G' is Δ, s is the length of the shortest edge in G', δ is the maximum distance between each p_i and q_i for any $i \in \{1, \ldots, n\}$, we can prove the following theorem and get a more accurate bound for $L(\mathbf{P})$.

Theorem 7. *If G is the k-NNG of the point set P in $\mathbb{R}^m$ space, then using SVD, we can find an approximate point set Q with rank$(\boldsymbol{Q}) = d < m$, and the Fiedler value of $L(\boldsymbol{P})$ can be bounded by $(1 + \frac{7\delta}{s})^2 \gamma_d \Delta \tau_d k/n)^{\frac{2}{d}}$ where $\gamma_d = 2(\pi + 1 + \frac{\pi}{\alpha})^2 (\frac{A_{d+1}}{V_d})^{2/d}$.*

Here A_d is the surface volume of a unit d-dimensional ball, and V_d is the volume of a unit d-dimensional ball. To make the idea look clearer, let's consider a simple example in $\mathbb{R}^2$ space. $Q = \{q_1, \ldots, q_n\}$ is a set of n points in $\mathbb{R}^2$ space. We perturb these n points in the direction perpendicular to the original plane and get a new set of n points, denoted by $P = \{p_1, \ldots, p_n\}$, in $\mathbb{R}^3$ space. Assuming that the smallest distance between any two points of Q is s, and the perturbation distance is at most δ. If $s \geq \delta$, we can get the following inequalities.

$$\|p_i - p_j\| \leq \sqrt{(2\delta)^2 + \|q_i - q_j\|^2} \leq \sqrt{5}\|q_i - q_j\|$$

If r_i is the k-NNG radius for q_i, and R_i is the k-NNG radius for p_i, then we can see that $R_i \leq \sqrt{5} r_i$ for all $i \in \{1, \ldots, n\}$. Therefore, we can use $\sqrt{5}$-overlap graph G' of Q to approximate the k-NNG G of P. And the Fiedler value of $L(\mathbf{G}')$ can also be bounded by the Fiedler value of $L(\mathbf{G})$. In fact, we can think

that all those n points of P in $\mathbb{R}^3$ are perturbed perpendicularly to the same plane and the new point set on the plane is Q.

To prove Theorem 7, let's get some preparations.

Lemma 4. *$\forall p_i \in P$, its k-NNG radius R_i is no more than $r_i + 2\delta$, where r_i is the k-NNG radius of the corresponding point q_i in Q.*

Lemma 5. *The 1-overlap graph of k-NNG of P in $\mathbb{R}^m$ is isomorphic to a subgraph of the $(1 + 7\delta/s)$-overlap graph of k-NNG of Q in $\mathbb{R}^d$.*

Lemma 6. *The k-NNG is a subgraph of its 1-overlap graph.*

Proof. Suppose G is a k-NNG, G' is the 1-overlap graph of G and (p_i, p_j) is an arbitrary edge of G. Then we can see that $\|p_i - p_j\| \leq r_i$ or $\|p_i - p_j\| \leq r_j$. In addition, $\|p_i - p_j\| \leq r_i + r_j$. Hence (p_i, p_j) must exists in the graph G'. From the generality of (p_i, p_j), we can see that the k-NNG graph is a subgraph of its 1-overlap graph. □

Combining Lemma 4, 5 and 6, we can derive the following corollary.

Corollary 4. *The k-NNG of P in $\mathbb{R}^m$ is isomorphic to a subgraph of the $(1 + 7\delta/s)$-overlap graph of a k-NNG of Q in $\mathbb{R}^d$, where $\delta = \max \|p_i - q_i\|$ and $s = \min \|q_i - q_j\|$.*

In [11] it is shown that any k-NNG is a subgraph of a $k\tau_d$-ply neighborhood system where τ_d is the kissing number in dimension d. If G is an α-overlap graph of a k-NNG in $\mathbb{R}^d$ then G is a subgraph of an α-overlap graph of a $k\tau_d$-neighborhood system in $\mathbb{R}^d$. Suppose that the maximum degree of G is Δ, we can apply Theorem 6 and get the following corollary directly.

Corollary 5. *If G is a subgraph of the α-overlap graph of k-NNG in $\mathbb{R}^d$ with maximum degree Δ, then the Fiedler value of $L(G)$ is bounded by $\gamma_d \Delta \alpha^2 (\tau_d k/n)^{2/d}$, where $\gamma_d = 2(\pi + 1 + \pi/\alpha)^2 (A_{d+1}/V_d)^{2/d}$.*

Finally, we give the proof of Theorem 7.

Proof of Theorem 7. The k-NNG of P in $\mathbb{R}^m$ is isomorphic to a subgraph of the $(1+7\delta/s)$-overlap graph of k-NNG of Q in $\mathbb{R}^d$, according to Corollary 6. The isomorphic graph has the same Fiedler value as the original graph because they have the same Laplacian matrices. Hence the k-NNG of P in $\mathbb{R}^m$ has no larger Fiedler value than the $(1 + 7\delta/s)$-overlap graph of k-NNG of Q in $\mathbb{R}^d$ according to Corollary 1.Due to Corollary 5, we know that the Fiedler value of k-NNG in $\mathbb{R}^m$ is bounded by $(1 + \frac{7\delta}{s})^2 \gamma_d \Delta \tau_d k/n)^{\frac{2}{d}}$ where $\gamma_d = 2(\pi + 1 + \frac{\pi}{\alpha})^2 (\frac{A_{d+1}}{V_d})^{2/d}$. □

5 Conclusion

In this paper, we concentrate on the combinatorial and spectral aspects of nearest neighbor graphs in doubling dimensional metric spaces and nearly-Euclidean spaces. For those k-nearest neighbor graphs in metric spaces with doubling

dimension γ, we give the ply bound and degree bound, where there are no definitions like volume or parallel. We analyze the shallow minor excluded property and bound the separator size. For those graphs in Euclidean spaces with high dimension, we prove that the k-nearest neighbor graphs could have better spectral properties using SVD. If the number k is a constant, then we can show that its Fiedler value can be bounded by $O(\Delta(1 + 7\delta/s)^2 n^{-2/d})$ where Δ is the maximum degree of the approximation graph.

References

1. E. R. Barnes and A. J. Hoffman, *Partitioning, spectra and linear programming*, Progress in Combinatorial Optimization, pages 13–25, (1984).
2. P. Dewilde and E. F. Deprettere, *Singular value decomposition: An introduction*, In Ed. F. Deprettere, editor. SVD and Signal Processing: Algorithms, Applications, and Architectures pages Elsevier Science Publishers North Holland.
3. C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1:211-218, (1936)
4. C. Eckart and G. Young, *A principal axis transformation for non-Hermitian matrices*, Bulletin of the American Mathematical Society, 45:118–121 (1939)
5. T. Feder, P. Hell, S. Klein, R. Motwani, *Complexity of Graph Partition Problems*, STOC 1999: 464-472.
6. M. Fiedler, *Algebraic connectivity of graphs*. Czechoslovak Mathematical Journal, 23(98):298-305, (1973)
7. Gene H. Golub and Gharles F. Van Loan,*Matrix Computations*, pages 16-21,293. Johns Hopkins University Press, Baltimore, Maryland (1992).
8. David R. Karger, Matthias Ruhl, *Finding nearest neighbors in growth-restricted metrics*. STOC 2002: 741-750.
9. Richard J. Lipton, Robert Endre Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math. 36:177-189, (1979)
10. Richard J. Lipton, Robert Endre Tarjan, *Applications of a planar separator theorem*, SIAM Journal on Computing 9:615-627, (1980)
11. Gary L. Miller, Shang-Hua Teng, William P. Thurston, Stephen A. Vavasis, *Separators for sphere-packings and nearest neighbor graphs*, J. ACM 44(1): 1-29 (1997)
12. A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl, 11:430–452, (1990)
13. S. Plotkin, S. Rao, and W.D. Smith, *Shallow excluded minors and improved graph decomposition*, In proceedings at the 5th Symposium Discrete Algorithms. SIAM, New York, pp.462-470 (1994)
14. Michael Ruzhansky, *On uniform properties of doubling measures*, Proc. of Amer. Math. Soc., 129: 3413-3416 (2001)
15. E. Schmidt, *Zur Theorie der linearen und nichtlinearen Integralgleichungen. I Teil. Entwicklung willkürlichen Funktionen nach System vorgeschriebener*, Mathematische Annalen, 63:433-476, (1907)
16. Daniel A. Spielman, Shang-Hua Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, STOC 2004: 81-90.
17. Daniel A. Speilman and Shang-Hua Teng, *Spectral Partitioning Works: Planar Graphs and Finite Element Meshes*, FOCS 1996: 96-105.
18. Shang-Hua Teng, *Combinatorial aspects of geometric graphs*, Computational Geometry no. 9: 277-287 (1998)

Maximum Edge-Disjoint Paths Problem in Planar Graphs

Mingji Xia*

State Key Lab. of Computer Science, Institute of Software,
Chinese Academy of Sciences,
P.O. Box 8717, Beijing 100080, China
Graduate University of Chinese Academy of Sciences, Beijing, China
xmj@gcl.iscas.ac.cn

Abstract. We give a randomized algorithm for maximum edge-disjoint paths problem (MEDP) and the minimal total length of MEDP, if the graphs are planar and all terminals lie on the outer face in the order $s_1, s_2, \ldots s_k, t_k, t_{k-1}, \ldots t_1$. Moreover, if the degree of the graph is bounded by 3, the algorithm becomes deterministic and can also output the number of optimal solutions. On the other hand, we prove that the counting version of these problems are #P-hard even if restricted to planar graphs with maximum degree 3.

Keywords: maximum edge-disjoint paths, determinant, #P-hard.

1 Introduction

In this paper, we investigate the maximum edge-disjoint paths problem in planar graphs. Given a graph $G = (V, E)$ and a set $T = \{\{s_1, t_1\}, \{s_2, t_2\}, \ldots \{s_k, t_k\}\}$ of pairs of vertices, the objective of edge-disjoint paths problem (denoted by EDP) is to connect all the pairs in T via edge-disjoint paths. This problem has a long history, and it has some application in areas such as VLSI. In the maximum edge-disjoint paths problem (MEDP), the objective is to find the maximum number of pairs in T that can be connected via edge-disjoint paths. #EDP (resp. #MEDP) denotes the number of methods of connecting all (resp. maximum) pairs via edge-disjoint paths. For convenience, we define function problems MLMEDP and #MLMEDP. MLMEDP asks for the minimum total length of disjoint paths connecting maximum pairs in T, and #MLMEDP asks for the number of methods to connecting maximum number of pairs in T with minimum total length. The vertices in T are called *terminals*. Let $R = \{s_1t_1, s_2t_2, \ldots s_kt_k\}$, we call $H = (V, R)$ a *demand graph*, and let $G + H = (V, E \cup R)$, where the disjoint union of E and R is taken, respecting multiplicities.

* The author is grateful to his supervisor Prof. Angsheng Li for advice and encouragement. Supported by NSFC Grant no. 60325206 and no. 60310213.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 566–572, 2007.
© Springer-Verlag Berlin Heidelberg 2007

EDP was shown to be NP-hard even on very restricted instances. Middendorf and Pfeiffer showed that even if $G + H$ is planar, EDP is NP-complete in [5]. So, MEDP is also NP-hard under this situation. It is shown in [1] that minimum total length of paths connecting all pairs in T is NP-hard, even if G is planar, all terminals lie on the boundary of outer face, and $G + H$ is Eulerian with maximum degree 4. MEDP and MLMEDP are also difficult to approximate. For example, it is shown in [17] that there is no constant approximation algorithm for MLMEDP even in planar graph, unless **P**=**NP**.

It has been shown that EDP is polynomial solvable, if $G + H$ is planar and k is bounded [8], or if G is planar, and all terminals lie on the outer face of G and $G+H$ is Eulerian [10]. The later follows from Okamura-Seymour theorem, which shows the cut condition is sufficient. A good survey is presented by Schrijver in part VII of [7].

If the supply graph G is planar and all terminals lie on the outer face in the order $s_1, s_2, \ldots s_k, t_k, t_{k-1}, \ldots t_1$, we give a randomized algorithm for MEDP and MLMEDP. Moreover, if the degree of $G + H$ is bounded by 3, the algorithm can compute them and #MLMEDP determinately. All these results also hold for directed graphs, and come out by reducing original problems to determinant. It is interesting to see that intuitively the gap between RP and NP-hard on problem MLMEDP is just the order of terminals. In contrast to our positive result for #MLMEDP, we also proved that #EDP, #MEDP and #MLMEDP are #P-hard (under Turing reduction) when $G + H$ is 3-regular planar graph, using polynomial interpolation technique [12], [11].

In section 2, we give the algorithm. In section 3, we prove the #P-hardness results.

2 Maximum Edge-Disjoint Paths Problem

We only consider the problems in unweighted graphs, by reducing it to the determinant of the adjacent matrix of a weighted graph.

Given a weighted directed graph $K = (V, E)$, there is a corresponding matrix $A_K = (w_{ij})$ of size $|V| \times |V|$, where w_{ij} is the weight of edge (i, j). A cycle cover of K is a subset E' of E, such that $K' = (V, E')$ is a graph whose vertices have out degree 1 and in degree 1, and the weight of a cycle cover E' is the product of the weights of its edges. The signed weight of a cycle cover is equal to the weight multiplied by $(-1)^m$, where m is the number of cycles of even length in this cycle cover. The determinant of A_K is

$$|A_K| = \sum_{\pi} \prod_{i=1}^{n} \sigma(\pi) w_{i,\pi(i)},$$

where the summation is over all permutation π on V, and $\sigma(\pi)$ is the sign of π. Obviously, $|A_K|$ is the summation of the signed weights of all cycle covers of K.

It is a natural idea to reduce MEDP to cycle cover, because if we consider the demand edges in the MEDP problem, a solution to MEDP with satisfied

demand edges compose a set of cycles. But there are some difference between the two problems. Each node is covered by exactly one cycle of a cycle cover, while there may be vertex not covered by any path or covered by many paths of a set of edge-disjoint paths. So we use a clique of size d with self-loops as gadget to simulate a vertex of degree d.

An example is shown in Fig. 1. The first graph is an instance $G + H$ of the MEDP problem, where the dotted edges mean demand edges. G is planar and the four terminals lie on the outer face in the order s_1, s_2, t_2, t_1. Replace each vertex in $G + H$ by the gadget, we get the second graph $K = (V_K, E_K)$. There are four kinds of edges in K, demand edges (edges in H), self-loops, clique edges (edges in gadgets except for self-loops), and path edges (the remaining edges which correspond to the edges in G).

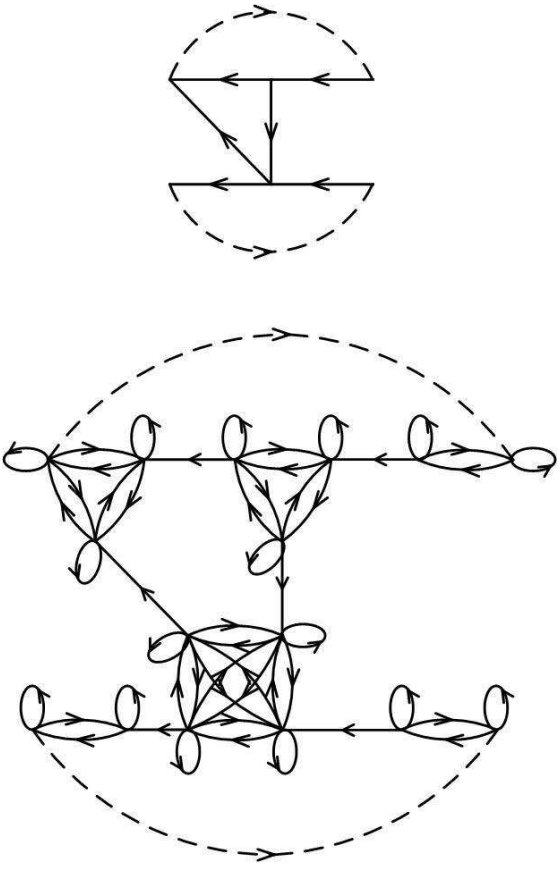


Fig. 1. An example of the reduction

Now we consider the relation of the edge-disjoint paths of G and the cycle cover of K. We have the following lemma.

Lemma 1. *If there is a set of edge-disjoint paths of G such that m pais are connected and the total length is l, then there is a cycle cover of K, which uses m demand edges, l path edges, and $l + m$ clique edges. If there is a cycle cover of K, which uses m demand edges and l path edges, then there is a set of edge-disjoint paths of G such that m pairs are connected and the total length is no more than l.*

Proof. The first conclusion is straightforward. The m satisfied demand edges, path edges taken in the edge-disjoint paths, some proper clique edges which connect the taken path edges, and the self-loops of the remaining uncovered nodes, compose a required cycle cover of K.

Given a cycle cover of K. We only consider the path edges in the cycle cover. Because G is planar and all terminals lie on the outer face in the order $s_1, s_2, \ldots s_k,\ t_k, t_{k-1}, \ldots t_1$, we can rearrangement them into edge-disjoint paths of G.

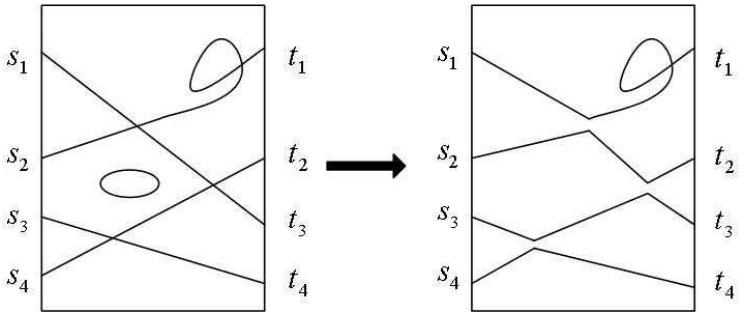


Fig. 2.

An example was shown in Fig. 2. There are only two circles of the circle cover are shown in the first graph of Fig. 2, since the other circles are self-loops. One of the two circles uses no demand edges, and the other is a circle passing $s_1, t_3, s_3, t_4, s_4, t_2, s_2, t_1, s_1$. The second circle can be decomposed into four paths as shown in the second graph. □

By this lemma, the MEDP of $G + H$ is just the maximum number of demand edges taken by cycle covers of K, and the MLMEDP is just the minimum number of path edges taken by the cycle covers of K which contain maximum demand edges.

Now we set the weights of edges in K. If $e \in E_K$ is a demand edge, it is given a weight $w_e x$, and if e is a path edge, it is given a weight $w_e y$, otherwise e is given weight w_e. Weighted directed graph K corresponds to a matrix A_K, whose determinant is a polynomial in x, y, and $w_e, e \in E_K$. The maximum x degree of this polynomial is just the answer to the MEDP problem on input $G + H$.

Although the determinant of a numerical matrix is polynomial computable, we don't know how to compute the determinant of a matrix with variable entries. But if there are only finite variables, we can compute the determinant in polynomial time by solving a system of linear equations in its coefficients, and each equation is obtained by computing the determinant after setting some specific values to the finite variables.

Given an instance $G+H$ of the MEDP problem, we get K and its matrix A_K, choose a random value from $\{0, 1, 2, \ldots, 2|E_K|\}$ for each variable $w_e, e \in E_K$, compute the determinant of A_K, which is a polynomial in x and y, output the maximum degree of x as the answer to the MEDP problem, and output the minimum degree of y of all monomials with maximum x degree as answer to MLMEDP problem.

The following lemma is used in the proof of the correctness of the algorithm.

Lemma 2 (Schwarz-Zippel Lemma). *Let $p(x_1, \ldots, x_m)$ be a polynomial, not identically zero, in m variables each of degree at most d in it , and let $M > 0$ be an integer. Then the number of m-tuples $(x_1, \ldots, x_m) \in \{0, 1, \ldots, M-1\}^m$ such that $p(x_1, \ldots, x_m) = 0$ is at most mdM^{m-1}.* □

Now we have,

Theorem 1. *Suppose G (directed or undirected) is planar and all terminals lie on the outer face in the order $s_1, s_2, \ldots s_k, t_k, t_{k-1}, \ldots t_1$, there is a randomized polynomial time algorithm for MEDP and MLMEDP, and if the maximum degree of $G + H$ is 3, there is a deterministic polynomial time algorithm for MEDP, MLMEDP and #MLMEDP.*

Proof. Suppose m and l are the right answers for MEDP and MLMEDP respectively. By lemma 1, the coefficient of $x^m y^l$ in $|A_K|$ is a nonzero polynomial f in $w_e, e \in E_K$, and m is the maximum degree of x, l is the minimum degree of y of all monomials with maximum x degree in $|A_K|$.

Polynomial f is in $|E_K|$ variables each of degree at most 1. By lemma 2, with probability at least $1/2$, the algorithm outputs m and l.

Suppose the maximum degree of $G + H$ is 3. The weight of clique edge is changed to z, and set all w_e to 1. Find the monomial with minimum z degree among the monomials of $|A_K|$ with maximum x degree, output the the x degree, the y degree and the absolute value of the coefficient of this monomial as the answers to MEDP, MLMEDP and #MLMEDP respectively.

Since the maximum degree of $G + H$ is 3, no paths share a vertex. Since the degree of z is minimized, there are only one way to pass a vertex gadget (the way composed of two clique edges is excluded), if the entrance and exit vertices are fixed. So the EDPs of total length l satisfying m demands, are one to one mapped to cycle covers of K which contain m demand edges, l path edges and $m + l$ clique edges. Moreover, it is obviously that all these cycle covers have the same sign, that is they all have signed weight $(-1)^m x^m y^l z^{m+l}$, because each of them contains m even cycles and some other cycles of length 1.

For undirected graphs, the analysis is the same and so the same conclusions follow. □

3 #P-Hardness

In this section, we prove,

Theorem 2. *#MEDP, #MLMEDP and #EDP is #P-hard(under Turing reduction), even if $G + H$ is 3-regular and planar.*

Proof. #3-regular bipartite planar vertex cover is #P-complete [16]. Given an instance $G = (V_G, E_G)$ of this problem, we construct some EDP instances G_i satisfying the requirements, and all of terminals of G_i can be connected by edge-disjoint paths, and all ways to connect them have the same total length, that is, $\#\mathrm{MEDP}(G_i) = \#\mathrm{MLMEDP}(G_i) = \#\mathrm{EDP}(G_i)$.

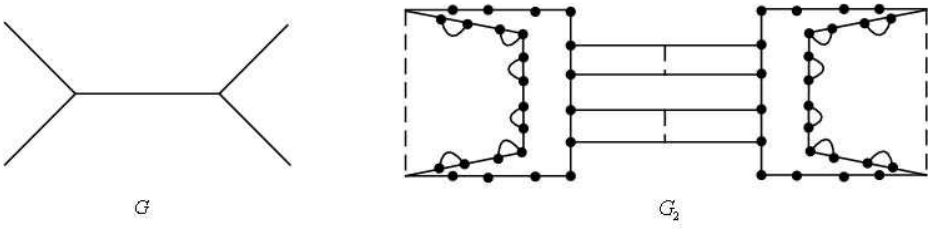


Fig. 3. Part of the graphs and the reduction

The reduction of local part is shown in Fig. 3. For each vertex in G, there is a demand edge (broken line) in G_i connected by two paths of length $6i + 1$. Some edges are added to the inner path to make the whole graph 3-regular. This (the square in the figure) is called a vertex gadget. The $6i$ vertices (the endpoints are excluded) of the outer path is divided into 3 groups each containing i pairs of vertices. Each group will be connected to the corresponding group of adjacent vertex gadget using i bridges as shown in the figure above.

Suppose G contains n vertices and m edges, and

$$\#\mathrm{VC}(G) = \sum_{j=1}^{n} a_j, \tag{1}$$

where a_j denotes the number of vertex covers of G containing j vertices.

We establish a map from EDPs of G_i to the vertex covers of G. A vertex is taken in the subset of V_G which is the image of an EDP, if and only if its gadget takes the inner path in the EDP. Obviously the subset is a VC of G and this is an onto map.

Now we count the number of EDPs that are mapped to one VC. Because G is 3-regular, a vertex cover of size j covers $3j - m$ edges from both sides and cover the other edges from one side. For a vertex gadget of G_i, there are 2^{6i} methods to take the inner path. For an edge covered from both sides, there are 2^i methods to satisfy the demand edge in the i bridges. So $2^{6ij}2^{i(3j-m)}$ EDPs of G_i are mapped to one vertex cover of size j of G. We have

$$\#\mathrm{EDP}(G_i) = \sum_{j=1}^{n} a_j 2^{6ij} 2^{i(3j-m)} = \sum_{j=1}^{n} a_j (2^{9i})^j 2^{-im}.$$

Taking i from 1 to n, we can get n equations about a_j, and get $\#\mathrm{VC}(G)$ from the values of a_j by equation 1. This is a polynomial time Turing reduction, the conclusion follows. □

References

1. U. Brandes, G. Neyer, and D. Wagner, Edge-disjoint paths in planar graphs with short total length. Technical Reports, 1996.
2. M. E. Fishier, Statistical mechanics of dimers on a plane lattice. Phys. Rev., 124 (1961), pp. 1664–1672.
3. P. W. Kasteleyn, The statistics of dimers on a lattice. Physica, 27 (1961), pp. 1209–1225.
4. P. W. Kasteleyn, Graph theory and crystal physics. In Graph Theory and Theoretical Physics (F. Harary, ed.), Academic Press, London, 1967, pp. 43–110.
5. M. Middendorf and F. Pfeiffer, On the complexity of the disjoint paths problem. Combinatorica, 13 (1993), pp. 97–107.
6. C. Papadimitriou: Computational Complexity. Addison–Wesley 1994.
7. A. Schrijver, Combinatorial Optimization: Polyhedra and Efficieny. Springer-Verlag, Berlin Hiedelberg, 2003.
8. A. Sebő, Integer plane multiflows with a fixed number of demands. J. Comb. Theory Ser. B, 59(1993), pp. 163–171.
9. H. N. V. Temperley and M. E. Fishier, Dimer problems in statistical mechanicsAn exact result. Philosophical Magazine, 6 (1961) pp. 1061–1063.
10. D. Wagner and K. Weihe, A linear-time algorithm for edge-disjoint paths in planar graphs. Conbinatorica, 15 (1995), pp. 135–150.
11. S. P. Vadhan, The complexity of counting in sparse, regular, and planar graphs. SIAM Journal on Computing, 31 (2001), pp. 398–427.
12. L. G. Valiant, The complexity of enumeration and reliability problems. SIAM Journal on Computing, 8 (1979), pp. 410–421.
13. L. G. Valiant, Quantum circuits that can be simulated classically in polynomial time, SIAM Journal on Computing, 31 (2002), pp. 1229-1254.
14. L. G. Valiant, Holographic algorithms (extended abstract). FOCS, 2004 pp. 306–315.
15. L. G. Valiant: Accidental Algorithms. FOCS, 2006 pp. 509–517.
16. M. Xia and W. Zhao, #3-regualar bipartite planar vertex cover is #P-complete. Theory and Applications of Models of Computation: Third International Conference, TAMC 2006, Beijing, China, May 15-20, (2006), pp. 356–364.
17. P. Zhang and W. Zhao, On the complexity and approximation fo the min-sum and min-max disjoint paths problems, manuscripts.

An Efficient Algorithm for Generating Colored Outerplanar Graphs

Jiexun Wang[1], Liang Zhao[1], Hiroshi Nagamochi[1], and Tatsuya Akutsu[2]

[1] Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{wangjx,liang,nag}@amp.i.kyoto-u.ac.jp
[2] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan
takutsu@kuicr.kyoto-u.ac.jp

Abstract. Given two integers n and m with $1 \leq m \leq n$, we consider the problem of generating nonisomorphic colored outerplanar graphs with at most n vertices, where each outerplanar graph is colored with at most m colors. In this paper, we treat outerplanar graphs as rooted outerplane graphs, i.e., plane embeddings with a designated vertex as the root, and propose an efficient algorithm for generating all such colored graphs based on a unique representation of those embeddings. Our algorithm runs in $O(n)$ space and outputs all colored and rooted outerplane graphs without repetition in $O(1)$ time per graph.

1 Introduction

The problem of enumerating all graphs in a particular class is one of the most fundamental issues in graph theory, as well as being very important in its wide applications in computer science and industry. The series of enumerated graphs can be used to find an optimal graph that satisfies more restricted conditions or to verify whether the graph class possesses some new features [1,2,3,6].

A *colored* graph is a graph in which every vertex is assigned a color in a finite set Σ. A graph is called *outerplanar* if it admits an embedding in the plane such that all vertices appear on the boundary of the outer face and no two edges cross each other except at their endvertices. It is well known that trees are a special type of outerplanar graphs. Some algorithms for enumerating trees for particular classes such as rooted trees and colored trees [7,8] have been developed.

Algorithms for enumerating colored graphs have extensive applications such as drug design via a theoretical approach. In recent years, many approaches in drug design lie on advanced computer-assisted techniques for cost and time saving. For example, Fujiwara et al. [3] designed an algorithm to generate tree-like chemical graphs based on the enumeration algorithm for colored trees due to Nakano and Uno [8]. Moreover, Horváth et al. [4] revealed that 94.3% of the chemical compounds in the NCI chemical database are outerplanar graphs. Thus, to establish a drug-design system based on the graph theoretical approach, it is

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 573–583, 2007.
© Springer-Verlag Berlin Heidelberg 2007

required to design an efficient algorithm for enumerating outerplanar graphs. To our best knowledge, however, few papers have focused on enumeration of colored or uncolored outerplanar graphs so far. These motivate us to study the problem of enumerating nonisomorphic colored outerplanar graphs efficiently.

In this paper, we present an algorithm that enumerates all colored outerplanar graphs with at most n vertices and at most m colors, where n and m are given integers such that $1 \leq m \leq n$. We show that the algorithm runs in $O(n)$ space and outputs all colored outerplanar graphs without repetition in $O(1)$ time per graph (this means that it outputs an $O(1)$ difference between two consecutive graphs in the series of outputs).

In this study, we first treat outerplanar graphs as rooted outerplane graphs, i.e., plane embeddings with a designated vertex as the root. Note that, unlike the tree case, outerplane graphs can contain cycles. It is of particular importance to find a reasonable and good unique representation of colored and rooted outerplane graphs, which avoids a predefined embedding drastically being changed in the process of generating cycles. Thus we introduce "left-heavy" embeddings based on a new vertex ordering "rightmost block-first depth-first search" (RBF DFS), which is originated from the "left-heavy" ordered trees based on depth-first search in [8]. We then define a parent-child relationship among all left-heavy embeddings to form a family tree rooted at an embedding of the designated vertex, and design a procedure to visit all nodes in the family tree in $O(1)$ time per node.

The rest of the paper is organized as follows. Section 2 reviews some basic notions. Section 3 constructs a family tree of left-heavy embeddings of colored outerplanar graphs based on a defined unique representation. Section 4 presents our algorithm to enumerate all outerplane graphs with at most n vertices and at most m colors. Section 5 makes some concluding remarks.

2 Preliminaries

A connected graph with at least three vertices is called *biconnected* if it is simple and has no cut vertex, i.e., a vertex whose removal results in a disconnected graph. A maximal induced subgraph of a graph is called a *block* if it has no cut vertex (i.e., it is biconnected or consists of a single edge). Two blocks in a graph are called *adjacent* if they share a common cut vertex. A block in a graph is called a *leaf-block* if it has at most one adjacent block.

A graph is called *planar* if it has a plane embedding. An *outerplanar* graph is a planar graph that admits a plane embedding such that every vertex lies on the boundary of the outer face. A *plane* (resp., *outerplane*) graph is a planar (resp., outerplanar) graph with (such) a fixed embedding. In a plane embedding, an edge is called *outer* if it is contained in the outer facial cycle, and *inner* otherwise. In this paper, we consider simple connected outerplanar graphs. Generally, there are three types of blocks in outerplane graphs:

Type 1: a single outer edge;
Type 2: a cycle consisting of at least three outer edges with no inner edge;

Type 3: (otherwise) a cycle consisting of at least three outer edges and at least one inner edge.

In an outerplane graph, a block of $p \geq 3$ vertices can have at most $p-3$ inner edges. We call a block *unsaturated* if it has less than $p-3$ inner edges, and *saturated* otherwise. Figure 1 illustrates an outerplane graph G consisting of five blocks, where blocks 2 and 4 are of Type 1, blocks 1 and 3 are of Type 2, block 5 is of Type 3, and blocks 3 and 5 are saturated leaf-blocks.

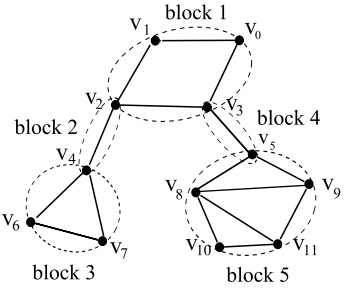


Fig. 1. An illustration of an outerplane graph G

Let a vertex r be designated as the root of an outerplane graph G. Define the root $r(B)$ for each block B be the vertex in B that is the closest to r. Let $\mathcal{B}(v)$ denote the set of all blocks B with $r(B) = v$. For two adjacent blocks B and B' with $r(B) \in V(B')$, we say that B is a *child-block* of B' if $r(B) \neq r(B')$ holds. We define a new search for rooted outerplane graphs, denoted *rightmost block-first depth-first search* (*RBF DFS*), which visits the graph by the following rules.

Rule 1: when visiting a block B, traverse all its outer edges from $r(B)$ by following the rightmost edges, and then backtrack to visit all inner edges;
Rule 2: then visit all child-blocks of B from right to left.

For instance, an outerplane graph G in Fig. 1 has the vertices $\{v_0, \ldots, v_{11}\}$. Starting from the root v_0, the RBF DFS labels other vertices in the order: v_1, v_2, v_3, v_4, v_6, v_7, v_5, v_8, v_{10}, v_{11}, v_9.

Correspondingly, the *leftmost block-first depth-first search* (*LBF DFS*) is defined by exchanging "right" and "left" in the above rules. Throughout the paper, we adopt the RBF DFS as the labeling method unless stated otherwise. From the rule of labeling, an edge (v_i, v_j) in G is called a *forward edge* if v_i is labeled earlier than v_j; otherwise it is called a *backward edge* .

Given a *colored* and rooted outerplane graph G with n vertices whose colors belong to a finite set Σ, we label the root as 0 and other vertices as $1, 2, \ldots, n-1$ by the RBF DFS. Denote the label sequence of G as $LS^R(G) = [c_0, p_0c_1, p_1c_2, \ldots, p_{n-2}c_{n-1}; x_1y_1, x_2y_2, \ldots, x_hy_h]$, where c_i is the color of the i-th vertex, p_{i-1} is the labeled vertex closest to the i-th vertex in the labeling, $i = 1, 2, \ldots, n-1$,

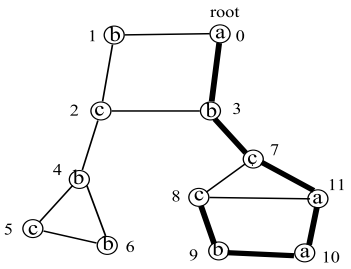


Fig. 2. An illustration of a colored and rooted outerplane graph G

and x_jy_j is the pair of labels of the end vertices of the j-th backward edge (may not exist), $j = 1, 2, \ldots, h$. Similarly, we denote as $LS^L(G)$ the label sequence of G obtained by the LBF DFS. For simplicity, $LS^R(G)$ may be written by $LS(G)$. Figure 2 illustrates a colored and rooted outerplane graph G, where $LS(G) = [a, 0b, 1c, 2b, 2b, 4c, 5b, 3c, 7c, 8b, 9a, 10a; 3\ 0, 6\ 4, 11\ 7, 11\ 8]$ ($\Sigma = \{a, b, c\}$). Letting r_0 be the root of graph, the *rightmost path* of G is denoted as $RP(G) = (r_0, r_1, \ldots, r_i, \ldots, r_z)$, which is defined in the following way: each r_j is the rightmost child of r_{j-1}, and either r_z has no child or it has some r_i as its rightmost child. In the latter case, the rightmost path contains a Type 2 or Type 3 leaf-block rooted at r_i. In Fig. 2, the rightmost path is depicted by the bold line.

3 The Family Tree of Left-Heavy Outerplane Graphs

For enumerating all outerplane graphs without repetition, we introduce "unique representations" of them, and define a "family relationship" among these representations.

3.1 Left-Heavy Outerplane Graphs

For two outerplane graphs G_1 and G_2, let their label sequences be

$$LS(G_1) = [c_0, p_0c_1, p_1c_1, \ldots, p_{n_1-2}c_{n_1-1}; x_1y_1, x_2y_2, \ldots, x_{h_1}y_{h_1}],$$
$$LS(G_2) = [c'_0, p'_0c'_1, p'_1c'_1, \ldots, p'_{n_2-2}c'_{n_2-1}; x'_1y'_1, x'_2y'_2, \ldots, x'_{h_2}y'_{h_2}],$$

where n_i is the number of vertices of G_i and h_i (can be 0) is the number of backward edges of G_i, $i = 1, 2$. We define relations $=_1$, $\gg_1$, $\supset_1$ and $>_1$ as follows:

(1) $LS(G_1) =_1 LS(G_2)$ if the label sequence of *forward* edges of G_1 (i.e., $[c_0, p_0c_1, \ldots, p_{n_1-2}c_{n_1-1}]$) is the same as that of G_2 ($[c'_0, p'_0c'_1, \ldots, p'_{n_2-2}c'_{n_2-1}]$;
(2) $LS(G_1) \gg_1 LS(G_2)$ if the label sequence of *forward* edges of G_1 is strictly lexicographically larger than that of G_2 but does not contain it as a prefix. For example, $[b, 0c, 1b, 2a; 3\ 0] \gg_1 [b, 0c, 1a, 2c; 3\ 0]$;

(3) $LS(G_1) \supset_1 LS(G_2)$ if the label sequence of *forward* edges of G_2 is a prefix of that of G_1 but not equal. For example, $[b, 0c, 1b, 2a, 3c; 4\ 0, 4\ 1] \supset_1 [b, 0c, 1b; 2\ 0]$;
(4) $LS(G_1) >_1 LS(G_2)$ if the label sequence of forward edges of G_1 is lexicographically larger than that of G_2, i.e., $LS(G_1) \gg_1 LS(G_2)$ or $LS(G_1) \supset_1 LS(G_2)$.

We let $LS(G_1) \supseteq_1 LS(G_2)$ mean $LS(G_1) \supset_1 LS(G_2)$ or $LS(G_1) =_1 LS(G_2)$, and let $LS(G_1) \geq_1 LS(G_2)$ mean $LS(G_1) >_1 LS(G_2)$ or $LS(G_1) =_1 LS(G_2)$. Accordingly, the definitions of $LS(G_1) =_2 LS(G_2)$, $LS(G_1) \gg_2 LS(G_2)$, $LS(G_1) \supset_2 LS(G_2)$, $LS(G_1) >_2 LS(G_2)$, $LS(G_1) \supseteq_2 LS(G_2)$ and $LS(G_1) \geq_2 LS(G_2)$ are respective with *backward* edges.

Besides, we denote as $LS(G_1) = LS(G_2)$ if $LS(G_1)$ equals to $LS(G_2)$, i.e., $LS(G_1) =_1 LS(G_2)$ and $LS(G_1) =_2 LS(G_2)$; and denote as $LS(G_1) > LS(G_2)$ if $LS(G_1)$ is lexicographically larger than $LS(G_2)$, i.e., $LS(G_1) >_1 LS(G_2)$, or, $LS(G_1) =_1 LS(G_2)$ and $LS(G_1) >_2 LS(G_2)$. Similarly, we let $LS(G_1) \geq LS(G_2)$ mean $LS(G_1) > LS(G_2)$ or $LS(G_1) = LS(G_2)$.

We say that a colored and rooted outerplane graph G is a *left-heavy graph* if the following conditions are satisfied:

(1) each pair of blocks B_1 and B_2 from left to right rooted at the same vertex satisfies $LS(T(B_1)) \geq LS(T(B_2))$, where $T(B_i)$ represents the subgraph consisting of B_i and all its descendants, $i = 1, 2$.
(2) for any Type 2 or Type 3 block B, it satisfies $LS^R(T(B)) \geq LS^L(T(B))$.

Clearly, any colored outerplanar graph has a unique left-heavy outerplane embedding, since its label sequence is larger than that of any other embeddings. Moreover, any two colored outerplanar graphs with the same left-heavy embedding are isomorphic. Thus, the task of enumerating all nonisomorphic colored outerplanar graphs can be completed by enumerating all left-heavy embeddings.

3.2 The Family Tree

Now we define a parent-child relationship between left-heavy graphs. Let G be a left-heavy outerplane graph. We define its *parent* $P(G)$ as the graph obtained by removing the rightmost leaf-block (leaving its root unremoved) if it is of Type 1 or Type 2, or, by removing the inner edge (s, t) with the smallest s and the largest t for s if it is of Type 3. It can be easily inferred that $P(G)$ is also left-heavy. Accordingly, G is called a *child* of $P(G)$. For a given G, we can repeatedly find a series of graphs $P(G), P^2(G) = P(P(G)), \ldots$ and obtain all its ancestors. See Fig. 3 for an example of the ancestors of G.

Clearly, the set of all left-heavy outerplane graphs with *exact* n vertices and all their ancestors constructs a *family tree* of all left-heavy outerplane graphs with *at most* n vertices and *at most* m colors. We show that, by reversing the parent-generating procedure, a child left-heavy outerplane graph can be obtained by adding some Type 1 or Type 2 block to a vertex on the rightmost path or by adding some inner edge to the unsaturated rightmost leaf-block carefully.

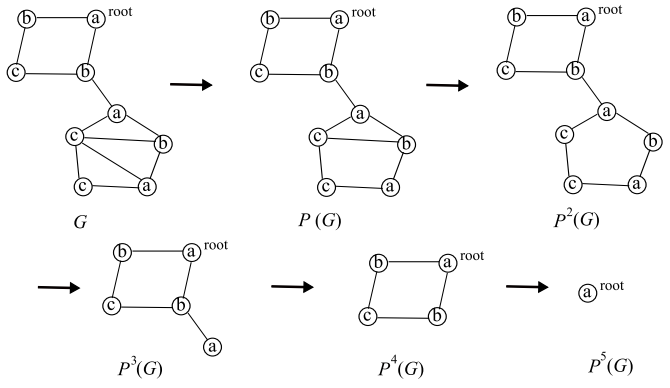


Fig. 3. The ancestors of a left-heavy outerplane graph G

4 Algorithm

Starting from a graph consisting of a single vertex (the root), our algorithm generates all left-heavy outerplane graphs by DFS (with some special handling, see Scenario A in the following part) in the family tree described in the previous section.

Before describing our algorithm, we need introduce some more notations. Suppose that a colored and rooted outerplane graph G have n' vertices, where every vertex has a color belonging to a finite set Σ. Let $l(v)$ denote the label of a vertex v. For example, the labels of vertices on the rightmost path in Fig. 2 are $(l(r_0), l(r_1), \ldots, l(r_6)) = (0, 3, \ldots, 8)$. For any r_i on the rightmost path with $|\mathcal{B}(r_i)| \geq 2$, we always denote the two rightmost blocks in $\mathcal{B}(r_i)$ from left to right as $\mathbf{B}_1(i)$ and $\mathbf{B}_2(i)$ (note $LS(T(\mathbf{B}_1(i))) \geq LS(T(\mathbf{B}_2(i)))$ if the left-heavy property is satisfied).

We say that $LS(T(\mathbf{B}_2(i)))$ is a *Class I prefix* of $LS(T(\mathbf{B}_1(i)))$ if $LS(T(\mathbf{B}_1(i))) \supseteq_1 LS(T(\mathbf{B}_2(i)))$ holds, and $LS(T(\mathbf{B}_2(i)))$ is a *Class II prefix* of $LS(T(\mathbf{B}_1(i)))$ if $LS(T(\mathbf{B}_1(i))) =_1 LS(T(\mathbf{B}_2(i)))$ and $LS(T(\mathbf{B}_1(i))) \supset_2 LS(T(\mathbf{B}_2(i)))$ hold. Also, we say that G is *active* at a vertex r_a in Class x if $|\mathcal{B}(r_a)| \geq 2$ and $LS(T(\mathbf{B}_2(a)))$ is a Class x prefix of $LS(T(\mathbf{B}_1(a)))$, where $x =$ I or II. Besides, we define the Class x *copy depth* d_x of G such that G is inactive at any vertex $r_0, \ldots, r_{d_x-1}$ but is active at vertex r_{d_x} in Class x; if no such a d_x exists, i.e., G is inactive at all vertices on the rightmost path in Class x, we let d_x be -1, where $x =$ I or II. As an exception, we let the Class II copy depth d_{II} of G be -2 if the rightmost leaf-block of G is of Type 1 or is saturated or there exists a vertex r_i on the rightmost path with $LS(T(\mathbf{B}_1(i))) = LS(T(\mathbf{B}_2(i)))$, where $0 \leq i < z$.

4.1 Generating Graphs by Adding a Type 1 or Type 2 Block

Scenario A: G is a single vertex v_0

If G consists of root v_0 only, a single outer edge or any left-heavy cycle of length at most n rooted at v_0 is a child graph of G. To enumerate all of them in $O(1)$

time each, we define a family tree $\mathcal{T}(v_0)$ on the set of the children of G (note that this family tree is different from that of left-heavy outerplane graphs mentioned in Sect. 3.2). The parent-child relationship in $\mathcal{T}(v_0)$ is defined in the following way. The parent of a left-heavy cycle (or an edge) with vertices $\{v_0, \ldots, v_k\}$ is defined to be the cycle obtained by replacing two edges $(v_{\lceil k/2 \rceil - 1}, v_{\lceil k/2 \rceil})$ and $(v_{\lceil k/2 \rceil}, v_{\lceil k/2 \rceil + 1})$ with a single edge $(v_{\lceil k/2 \rceil - 1}, v_{\lceil k/2 \rceil + 1})$ and removing the vertex $v_{\lceil k/2 \rceil}$. Observe that the parent remains leaf-heavy. Based on this, for a left-heavy cycle, all its children can be generated by inserting a new vertex with a color $c \in \Sigma$ between $v_{\lceil k/2 \rceil}$ and $v_{\lceil k/2 \rceil + 1}$, where if k is odd and there is no vertex v_ℓ ($\ell \in [1, \frac{k-1}{2}]$) such that $v_l > v_{k-l+1}$, then c should be not larger than the color of $v_{\frac{k+1}{2}}$.

Now we consider the case that G is left-heavy and has at least one block. We distinguish the condition $LS(T(\mathbf{B}_1(i))) \geq LS(T(\mathbf{B}_2(i)))$ as the following three cases: 1) $LS(T(\mathbf{B}_1(i))) \gg_1 LS(T(\mathbf{B}_2(i)))$; 2) $LS(T(\mathbf{B}_1(i))) =_1 LS(T(\mathbf{B}_2(i)))$ and $LS(T(\mathbf{B}_1(i))) \geq_2 LS(T(\mathbf{B}_2(i)))$; and 3) $LS(T(\mathbf{B}_1(i))) \supset_1 LS(T(\mathbf{B}_2(i)))$. We check the vertices on the rightmost path of a given graph starting from the root r_0. If all vertices on the rightmost path belong to 1) or there exists a vertex r_a belonging to 2), then we only need to consider how to make the subgraph $T^*(\mathbf{B}_2(i))$ obtained by adding a new block B to a vertex of $\mathbf{B}_2(i)$ be left-heavy, which guarantees the left-heavy property for the whole derived graph. Otherwise, if there exists a vertex r_a belonging to 3), then we have to check an additional condition ($LS(T(\mathbf{B}_1(a))) =$) $LS(T^*(\mathbf{B}_1(a))) \geq LS(T^*(\mathbf{B}_2(a)))$.

More precisely, when searching the vertices on the rightmost path starting from r_0, there are three scenarios depending on d_{I} (see Fig. 4 for an illustration):

*Scenario B*1: $d_{\mathrm{I}} = -1$;
*Scenario B*2: $d_{\mathrm{I}} > -1$, $LS(T(\mathbf{B}_1(d_{\mathrm{I}}))) =_1 LS(T(\mathbf{B}_2(d_{\mathrm{I}})))$ and $LS(T(\mathbf{B}_1(d_{\mathrm{I}}))) \geq_2 LS(T(\mathbf{B}_2(d_{\mathrm{I}})))$;
*Scenario B*3: (otherwise) $d_{\mathrm{I}} > -1$ and $LS(T(\mathbf{B}_1(d_{\mathrm{I}}))) \supset_1 LS(T(\mathbf{B}_2(d_{\mathrm{I}})))$.

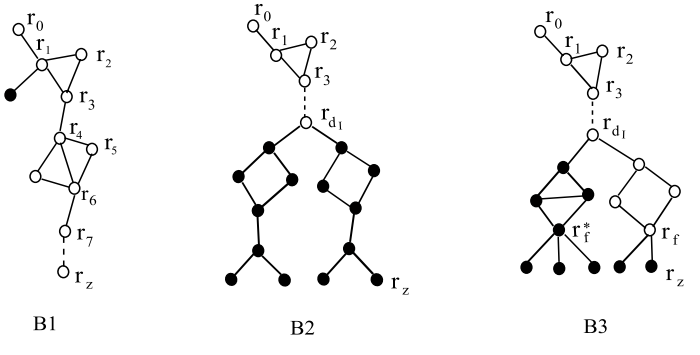


Fig. 4. An illustration of Scenarios B1, B2 and B3

Notes: The hollow vertices indicate that they can be added a new block

Let n_2 be the number of vertices of $T(\mathbf{B}_2(d_{\mathrm{I}}))$, p'_{n_2-1} be the label of the vertex closest to the n_2-th in the labeling in $T(\mathbf{B}_1(d_{\mathrm{I}}))$, and $r_f \in RP(G)$ be the unique vertex in $T(\mathbf{B}_2(d_{\mathrm{I}}))$ such that $l(r_f) = p'_{n_2-1} + l(r_{i+1}) - 1$ (in Scenario B3). Suppose that $\mathbf{B}_2(i)$ contains vertices $\{r_i, r_{i+1}, \ldots, r_g\}$ on the rightmost path. Then a Type 1 or Type 2 left-heavy block B with at most $n - n'$ vertices can be added to a vertex $r_j \in RP(G)$ in $\mathbf{B}_2(i)$ if the index i satisfies $0 \leq i \leq z$ in Scenario B1 (resp., $0 \leq i \leq d_{\mathrm{I}}$ in Scenario B2, and $0 \leq i < f$ in Scenario B3) and the index j satisfies one of the following conditions: "$j = i$ and $LS(T(\mathbf{B}_2(i))) \geq LS(B)$," "$i < j < g < z$," and "$i < j \leq g = z$."

Besides, in Scenario B3, when B is added to r_f, then it should satisfy the additional condition $LS(T(r_f^*)) \geq LS(B)$ if $LS(T(\mathbf{B}_1(d_{\mathrm{I}}))\backslash T(r_f^*)) \supset_2 LS(T(\mathbf{B}_2(d_{\mathrm{I}})))$ holds, otherwise the condition $LS(T(r_f^*)) >_1 LS(B)$, where r_f^* is the (n_2+1)-th labeled vertex of $T(\mathbf{B}_1(d_{\mathrm{I}}))$, $T(\mathbf{B}_1(d_{\mathrm{I}}))\backslash T(r_f^*)$ is the subgraph by removing $T(r_f^*)$ from $T(\mathbf{B}_1(d_{\mathrm{I}}))$ (leaving vertex r_f^* unremoved). It can be shown that these conditions guarantee the left-heavy property of the graph obtained by adding B to the vertex r_f. Further details are omitted due to the page limitation.

To achieve an $O(1)$ time enumeration, the copy depths need to be updated in $O(1)$ time. We show that the two copy depths of a child graph d'_{I} and d'_{II} can be calculated as Table 1, where Scenarios C1 and C2 will be discussed in Sect. 4.2 and the formulas (1)-(4) in the table are given as below.

Table 1. Updating copy depths of a child graph

	Scenario					
Copy depth	A	B1	B2	B3	C1	C2
d'_{I}	-1	(1)		(2)	d_{I}	
d'_{II}	(3)				-1	(4)

$$d'_{\mathrm{I}} = \begin{cases} i, & \text{if } B \text{ is added to the root of } \mathbf{B}_2(i) \text{ with } LS(\mathbf{B}_2(i)) \supseteq_1 LS(B), \\ & \quad \text{where } 0 \leq i < z; \\ -1, & \text{otherwise.} \end{cases} \tag{1}$$

$$d'_{\mathrm{I}} = \begin{cases} d_{\mathrm{I}}, & \text{if } B \text{ is added to the vertex } r_f \text{ satisfying } LS(T(r_f^*)) \supseteq_1 LS(B); \\ i, & \text{if } B \text{ is added to the root of } \mathbf{B}_2(i) \text{ with } LS(\mathbf{B}_2(i)) \supseteq_1 LS(B), \\ & \quad \text{where } 0 \leq i < f; \\ -1, & \text{otherwise.} \end{cases} \tag{2}$$

$$d'_{\mathrm{II}} = \begin{cases} -2, & \text{if } B \text{ is a Type 1 block or there exists } r_i \text{ satisfies } LS(T^*(\mathbf{B}_1(i))) \\ & \quad = LS(T^*(\mathbf{B}_2(i))), \text{ where } 0 \leq i < z; \\ i, & \text{if } B \text{ is added to the root of } \mathbf{B}_2(i) \text{ with } LS(T(\mathbf{B}_2(i))) =_1 LS(B) \\ & \quad \text{and } LS(T(\mathbf{B}_2(i))) \supset_2 LS(B), \text{ where } 0 \leq i < z; \\ -1, & \text{otherwise.} \end{cases} \tag{3}$$

$$d'_{\mathrm{II}} = \begin{cases} d_{\mathrm{II}}, & \text{if } s = x'_{h_2+1},\ t = y'_{h_2} \text{ and } h_1 > h_2 + 1; \\ -2, & \text{if } s = x'_{h_2+1},\ t = y'_{h_2} \text{ and } h_1 = h_2 + 1; \\ -1, & \text{if } y'_{h_2} \leq t \leq y'_{h_2+1} \text{ and } t + 1 < s < x'_{h_2+1}. \end{cases} \tag{4}$$

4.2 Generating Children Graphs by Adding an Inner Edge

Suppose that the rightmost leaf-block of the current graph G is an unsaturated block and there does not exist a vertex r_i on the rightmost path whose two rightmost children blocks $\mathbf{B}_1(i)$ and $\mathbf{B}_2(i)$ satisfy $LS(T(\mathbf{B}_1(i))) = LS(T(\mathbf{B}_2(i)))$, where $0 \leq i < z$. We can generate one child of G by adding a new inner edge (s,t) to the rightmost unsaturated leaf-block in the predefined order, where s is decreasing and t is increasing for the same s. Similar with the analysis in Sect. 4.1, we can search the vertices on the rightmost path starting from r_0. There are following two scenarios depending on the Class II copy depth d_{II} of G.

*Scenario C*1: $d_{\mathrm{II}} = -1$

Suppose that the rightmost leaf-block $\mathbf{B}_2(i)$ contains h (≥ 1) backward edges and other q vertices, where the vertices $\{r_i, \ldots, r_z\}$ are on the rightmost path, $z \geq i+2$. Its label sequence can be denoted by $LS(\mathbf{B}_2(i)) = [c_{l(r_i)}, 0c_{l(r_i)+1}, \ldots, (q-1)c_{l(r_i)+q}; x_1y_1, \ldots, x_hy_h]$. Since $\mathbf{B}_2(i)$ is unsaturated, there exists a p satisfying one of the following cases (see Fig. 5):

(1) if $x_h - y_h > 2$ holds, i.e. $p = h$, then (s,t) should satisfy $y_h < t < t+1 < s = x_h$ or $y_h \leq t < t+1 < s < x_h$;
(2) if $x_h - y_h = 2$, $y_w - y_{w-1} = 2$ and $y_p - y_{p-1} > 2$ hold, where $w = p+1,\ p+2,\ \ldots,\ h$, $2 \leq p \leq h$, then (s,t) should satisfy $y_{p-1} < t < t+1 < s = y_p$ or $y_{p-1} \leq t < t+1 < s < y_p$.

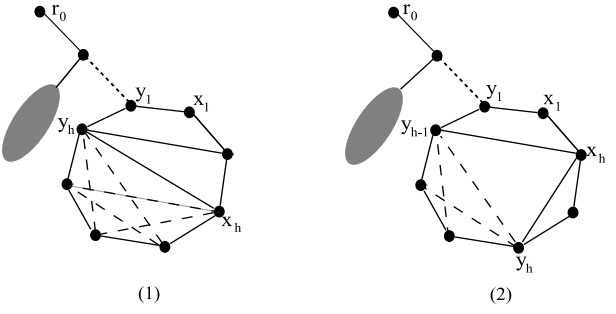


Fig. 5. An illustration of two cases that inner edge (s,t) can be added to the rightmost leaf-block in Scenario C1 (all candidate inner edges are shown by the dash line)

*Scenario C*2: $d_{\mathrm{II}} > -1$

Here the conditions $LS(T(\mathbf{B}_1(d_{\mathrm{II}}))) =_1 LS(T(\mathbf{B}_2(d_{\mathrm{II}})))$ and $LS(T(\mathbf{B}_1(d_{\mathrm{II}}))) \supset_2 LS(T(\mathbf{B}_2(d_{\mathrm{II}})))$ hold. Suppose that $LS(T(\mathbf{B}_1(d_{\mathrm{II}})))$ and $LS(T(\mathbf{B}_2(d_{\mathrm{II}})))$ are

$$LS(T(\mathbf{B}_1(d_{\mathrm{II}}))) = [c'_0, p'_0c'_1, \ldots, p'_{n_1-2}c'_{n_1-1}; x'_1y'_1, \ldots, x'_{h_2}y'_{h_2}, \ldots, x'_{h_1}y'_{h_1}],$$
$$LS(T(\mathbf{B}_2(d_{\mathrm{II}}))) = [c'_0, p'_0c'_1, \ldots, p'_{n_1-2}c'_{n_1-1}; x'_1y'_1, \ldots, x'_{h_2}y'_{h_2}],$$

where n_i is the number of vertices of subgraph $T(\mathbf{B}_i(d_{\mathrm{II}}))$ and h_i is the number of backward edges of $T(\mathbf{B}_i(d_{\mathrm{II}}))$, $i = 1, 2$, $1 \leq h_2 < h_1$. A new inner edge

(s,t) can be added to the rightmost leaf-block in the predefined order satisfying $y'_{h_2} \leq t < t+1 < s < x'_{h_2+1}$, or, $t+1 < s = x'_{h_2+1}$ and $y'_{h_2} \leq t \leq y'_{h_2+1}$.

Based on the above analysis, we present the outline of the algorithm as follows.

Find-all-graph
Input: An integer $n \geq 1$ and a color set $\Sigma = \{c_1, \ldots, c_m\}$
Output: All left-heavy outerplane graphs with at most n vertices and at most m colors
begin
  for each color $c \in \Sigma$ **do**
    $G := (\{v_0\}, \emptyset)$; Color G with c;
    for each child G' of G **do** /* G' is an edge or a cycle in *Scenario A* */
      Call **Find-all-children-I**(G');
      Call **Find-all-children-II**(G');
    end for
  end for
end

Find-all-children-I (G)
/* Generate the children of G by adding a Type 1 or Type 2 block */
begin
  for each child G' of G **do**
   /* G' is generated by the method in Scenarios B1, B2 and B3
    if "$d_{\mathrm{I}} = -1$," "$d_{\mathrm{I}} \geq 0$ and $LS(T(\mathbf{B}_1(d_I))) =_1 LS(T(\mathbf{B}_2(d_I)))$," and
   "$d_{\mathrm{I}} \geq 0$ and $LS(T(\mathbf{B}_1(d_I))) \supset_1 LS(T(\mathbf{B}_2(d_I)))$," respectively */
    Call **Find-all-children-I**(G');
    Call **Find-all-children-II**(G');
  end for
end

Find-all-children-II (G)
/* Generate the children of G by adding an inner edge */
begin
  for each child G' of G **do**
   /* G' is generated by the method in Scenarios C1 and C2
    if "$d_{\mathrm{II}} = -1$" and"$d_{\mathrm{II}} \geq 0$," respectively */
    Call **Find-all-children-I**(G');
  end for
end

We can show the time and space complexity of our algorithm in a similar way with the tree-enumeration algorithm due to Nakano and Uno [8], and obtain the following theorem whose proof is omitted due to the space limitation.

Theorem 1. *All colored and rooted outerplane graphs with at most n vertices and at most m colors can be enumerated without repetition in $O(1)$ time per graph and in $O(n)$ space.* □

5 Conclusion

In this paper, we have presented an efficient algorithm to enumerate all colored and rooted left-heavy outerplane graphs with at most n vertices and at most m ($\leq n$) colors. It runs in $O(n)$ space and outputs all colored and rooted outerplane graphs without repetition in $O(1)$ time per graph.

References

1. T. Akutsu and D. Fukagawa, Inferring a graph from path frequency. In Proc. 16th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, 3537 (2005), 371-382.
2. T. Akutsu and D. Fukagawa, On inference of a chemical structure from path frequency. In Proc. 2005 International Joint Conference of InCoB, AASBi and KSBI, (2005) 96-100.
3. H. Fujiwara, L. Zhao, H. Nagamochi and T. Akutsu, Enumerating tree-like chemical structures from feature vector (in preparation).
4. T. Horváth, T. Akutsu and S. Wrobel, A refinement operator for outerplanar graphs. In Proc. 16th International Conference of Inductive Logic Programming, (2006) 95-97.
5. G. Li and F. Ruskey, The advantage of forward thinking in generating rooted and free trees. In Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, (1999) 939-940.
6. H. Nagamochi, A detachment algorithm for inferring a graph from path frequency. In Proc. 12th Annual International Computing and Combinatorics Conference (COCOON'06), Lecture Notes in Computer Science, 4112 (2006) 274-283.
7. S. Nakano and T. Uno, Efficient generation of rooted trees, NII Technical Report (NII-2003-005) (2003) (http://research.nii.ac.jp/TechReports/03-005E.html).
8. S. Nakano and T. Uno, Generating colored trees. Lecture Notes in Computer Science, 3787 (2005) 249-260.

Orthogonal Drawings for Plane Graphs with Specified Face Areas

Akifumi Kawaguchi and Hiroshi Nagamochi

Department of Applied Mathematics and Physics,
Graduate School of Informatics, Kyoto University, Japan
kawaguti@amp.i.kyoto-u.ac.jp, nag@amp.i.kyoto-u.ac.jp

Abstract. We consider orthogonal drawings of a plane graph G with specified face areas. For a natural number k, a k-gonal drawing of G is an orthogonal drawing such that the outer cycle is drawn as a rectangle and each inner face is drawn as a polygon with at most k corners whose area is equal to the specified value. In this paper, we show that several classes of plane graphs have a k-gonal drawing with bounded k; A slicing graph has a 10-gonal drawing, a rectangular graph has an 18-gonal drawing and a 3-connected plane graph whose maximum degree is 3 has a 34-gonal drawing. Furthermore, we showed that a 3-connected plane graph G whose maximum degree is 4 has an orthogonal drawing such that each inner facial cycle c is drawn as a polygon with at most $10p_c + 34$ corners, where p_c is the number of vertices of degree 4 in the cycle c.

1 Introduction

Graph drawing has important applications in many areas in computer science such as VLSI design, information visualization and so on. Various graphic standards are used and studied for drawing graphs [5].

Orthogonal drawings, in which every edge is drawn as a sequence of alternate vertical and horizontal segments, have applications in circuit design, geometry and construction. Many aspects have been studied on orthogonal drawings. Studies of an orthogonal drawing with specified face areas have begun recently. For a natural number k, a k-gonal drawing of a graph is an orthogonal drawing such that the outer cycle of the graph is drawn as a rectangle and that each inner face is drawn as a polygon with k corners. Rahman, Miura and Nishizeki [6] proposed an 8-gonal drawing for a special class of plane graphs called a good slicing graph. Recently, de Berg, Mumford and Speckmann [2] proved that a general slicing graph admits a 12-gonal drawing. They also showed that a rectangular graph admits a 20-gonal drawing and a 3-connected plane graph whose maximum degree is 3 admits a 60-gonal drawing.

In this paper, we show that a general slicing graph has a 10-gonal drawing, a rectangular graph has an 18-gonal drawing and a 3-connected plane graph whose maximum degree is 3 has a 34-gonal drawing. Our approach for a general slicing graph is different from that by de Berg et al. [2]. We also show that every 3-connected plane graph G whose maximum degree is 4 has an orthogonal drawing

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 584–594, 2007.
© Springer-Verlag Berlin Heidelberg 2007

such that each inner facial cycle c is drawn as a polygon with at most $10p_c + 34$ corners if no vertex whose degree is 4 is on the outer cycle of G, where p_c is the number of vertices of degree 4 in the cycle c.

The paper is organized as follows. Section 2 gives some definitions of graphs. Section 3 introduces outlines of the algorithm for a 10-gonal drawing of a slicing graph. Sections 4 and 5 discuss an 18-gonal drawing of a rectangular graph, and an orthogonal drawing of a 3-connected plane graph, respectively. Finally Section 6 concludes.

2 Preliminary

A plane graph is denoted by $G = (V, E, F, c_0)$, where V, E, F and c_0 denote a set of vertices, a set of edges, a set of inner faces and the outer face, respectively. Let $n = |V|, m = |E|$ and $f = |F|$. Since G is a plane graph, $m = O(n)$ and $f = O(n)$ hold. A vertex of degree k is called a *k-degree vertex*. We denote the maximum degree of a graph G by $\Delta(G)$. An *orthogonal drawing* of a plane graph G is a drawing such that each edge $e \in E$ is drawn as an alternate sequence of vertical and horizontal line segments, and any two edges do not intersect except at their common end. It is known [3] that a plane graph G admits an orthogonal drawing if and only if $\Delta(G) \leq 4$. For a natural number k, an orthogonal drawing is called a *k-gonal drawing* if the outer cycle of G is drawn as a rectangle, and each inner facial cycle c_i is drawn as a polygon with at most k corners.

We consider a plane graph G such that the area of each inner face $c_i \in F$ is specified by a real $a_i > 0$. Let A be a set of areas a_i, and we denote a plane graph with the specified face areas by (G, A). For a plane graph (G, A), we consider an orthogonal drawing such that the area of each face c_i is equal to a_i. Figure 1 illustrates an example of a plane graph with specified face areas, and its 10-gonal drawing.

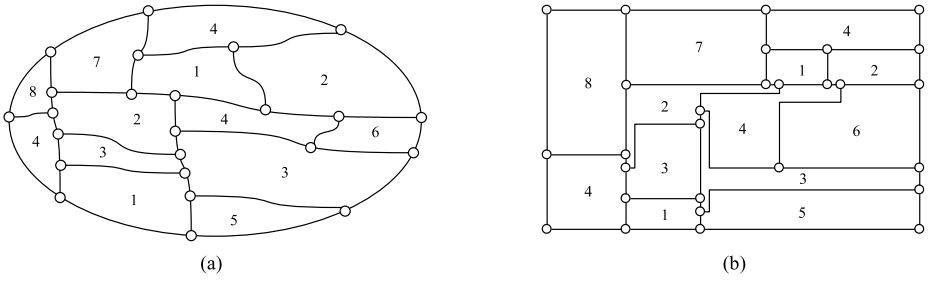


Fig. 1. (a) An example of a plane graph (G, A) with specified areas, where the number in each face represents the area specified for the face; (b) A 10-gonal drawing of (G, A)

Let G be a plane graph that has exactly four 2-degree vertices a, b, c and d in its outer cycle. We call these four vertices a, b, c and d *corner vertices*. The

four corners a, b, c and d divide the outer cycle of G into four paths sharing end vertices; the *top path*, the *bottom path*, the *left path* and the *right path*. We call each of these four paths an *unit path*. A path π in G which does not pass through any other outer vertex is called a *vertical* (*horizontal*) path of G if one end of π is on the top (left) path and the other is on the bottom (right) path. Such a path π divides the interior of G into two areas, each of which is enclosed by a cycle and induces a subgraph of G (the subgraph consisting of edges and vertices in the area and the cycle). We say that π *slices* G into these two subgraphs of G.

A *slicing graph* G is a plane graph that is defined recursively as follows; a cycle G of length 4 with a single inner face is a slicing graph, and G has a vertical or horizontal path π such that each of the two subgraphs generated from G by slicing G with π is a slicing graph. Note that $\Delta(G) \leq 4$ for every slicing graph G. A vertical or horizontal path in slicing graph G is called a *slicing path* if two subgraphs generated by slicing G with π are slicing graphs.

A *slicing tree* T is a binary tree which represents a recursive definition of a slicing graph G. We call a non-leaf node of T an *internal node*. Each node u in T corresponds to a subgraph G_u of G. Let u be an internal node in T, and v and w be the left and right child of u, respectively. Then we denote by π_u the slicing path that slices G_u into G_v and G_w; If π_u is vertical (horizontal), then G_v is the upper (left) subgraph of G_u, and G_w is the lower (right) subgraph of G_u. The node u is called a *V-node* if π_u is vertical, and u is called an *H-node* if π_u is horizontal. For a leaf u' of T, the corresponded subgraph $G_{u'}$ has one inner face c_i. Figure 2 illustrates an example of a slicing tree and a slicing graph corresponded to each node of T.

A *rectangular graph* is a plane graph whose outer face and each inner face can be drawn as a rectangle. Note that $\Delta(G) \leq 4$ for every rectangular graph G. A *3-connected plane graph* is a plane graph that remains connected even after removal of any two vertices together with edges incident to them.

In this paper, we show the following results, where a "combined decagon" is defined in the next section.

Theorem 1. *Every slicing graph with specified face areas has a 10-gonal drawing such that each inner face is drawn as a combined decagon. Such a drawing can be found in $O(n)$ time if its slicing tree and four corner vertices on the outer rectangle are given.* □

Theorem 2. *Every rectangular graph with specified face areas has an 18-gonal drawing. Such a drawing can be found in $O(n \log n)$ time if its outer rectangle and its four corner vertices are given.* □

Theorem 3. *Every 3-connected plane graph (G, A) with $\Delta(G) = 3$ has a 34-gonal drawing. Such a drawing can be found in $O(n \log n)$ time.* □

Corollary 1. *For every 3-connected plane graph (G, A) with $\Delta(G) = 4$ such that there are no 4-degree vertices on the outer cycle of G, there is an orthogonal drawing such that* (i) *each face has at most $10p_c + 34$ corners, where p_c is the number of 4-degree vertices in its facial cycle of $c \in F$, and* (ii) *the number of straight-lines in the entire drawing is at most $28n$.* □

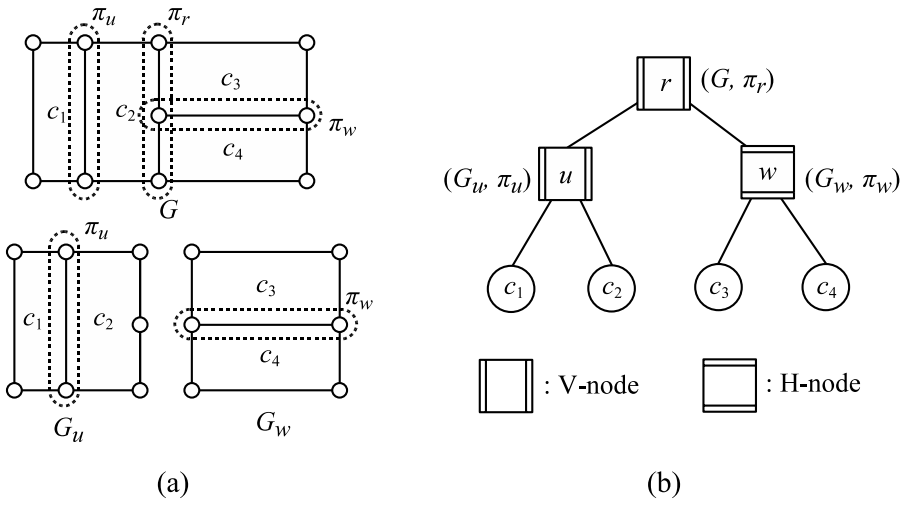


Fig. 2. (a) A slicing graph G and subgraphs G_u and G_w of G; (b) A slicing tree with nodes r, u and w

3 Drawings of Slicing Graphs

By definition, every inner face of a slicing graph can be drawn as a rectangle if we ignore the area constraint. To equalize the area of inner face to the specified value, we need to draw some edges with sequences of several straight-line segments.

We define a *step-line* as an alternate sequence of three vertical and horizontal straight-line segments. A step-line has two corners, which we call *bends*. A vertical step-line (VSL) is a sequence of vertical, horizontal and vertical straight-line segments. A horizontal step-line (HSL) is a sequence of horizontal, vertical and horizontal straight-line segments.

Based on step-lines, we introduce a polygon called a "combined decagon," which plays a key role to find a 10-gonal drawing of a slicing graph.

3.1 Combined Decagon

We introduce how to draw a cycle with four corner vertices as a k-gon with $4 \leq k \leq 10$. We consider a plane graph G of cycle $G = (\{a, b, c, d\}, \{(a, b), (b, c), (c, d), (d, a)\})$. Note that path ab is the top path, dc is the bottom path, ad is the left path and bc is the right path of G. We call path dab the *top-left path* of G.

We consider a k-gon ($4 \leq k \leq 10$) in which each path is drawn as a line segment, a VSL, an HSL or a pair of these. We use several types of combinations

of lines for each of the top-left path, the right path and the bottom path; Five types for the top-left path (Fig. 3), three types for the right path (Fig. 4), and three types for the bottom path (Fig. 5).

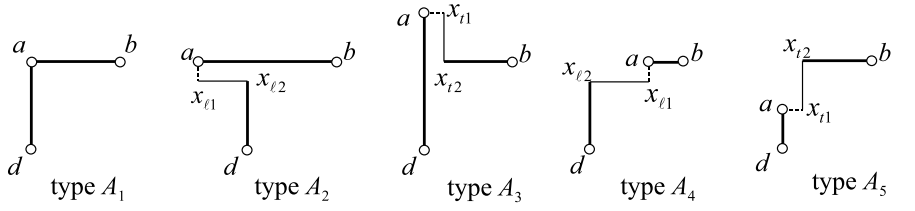


Fig. 3. Five types of drawing pattern for the top-left path *dab*

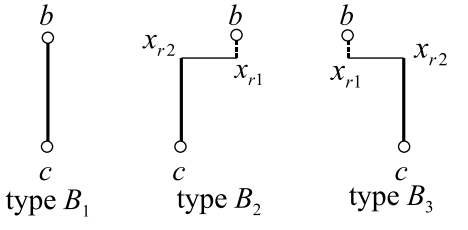


Fig. 4. Three types of drawing pattern for the right path *bc*

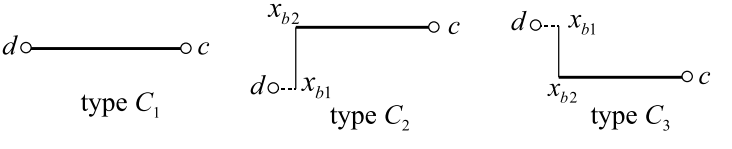


Fig. 5. Three types of drawing pattern for path *dc*

We draw cycle (a, b, c, d) by choosing a drawing pattern A_i $(i = 1, 2, 3, 4, 5)$ for the top-left path, B_j $(j = 1, 2, 3)$ for the right path and C_k $(k = 1, 2, 3)$ for the bottom path. Note that the resulting polygon has at most 10 corners. A *combined decagon* P is defined as a polygon such that each unit path of P is drawn as a straight-line or a step-line and at least one of its top and left paths is drawn as a straight-line. Figure 6 illustrates examples of a combined decagon. We may let A_i denote the set of combined decagons such that the top-left path is drawn as a pattern in A_i. Similarly for B_j and C_k.

Let P be a combined decagon. A line segment in the top-left path is called *connectable* if it is incident to corner b or d. Similarly a line segment in the right (bottom) path is called *connectable* if it is incident to corner c. Other line

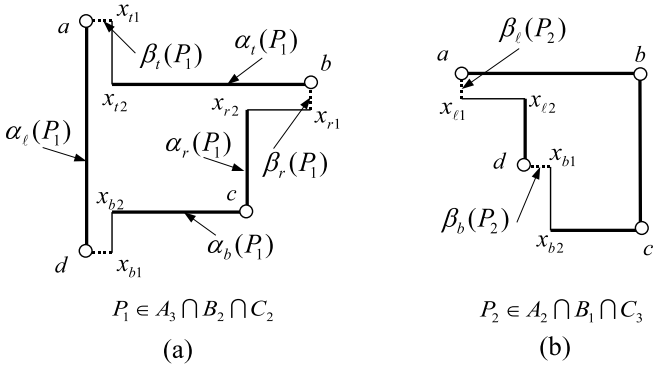


Fig. 6. Illustration of combined decagons P_1 and P_2

segments are called *unconnectable*. In Figs. 3, 4 and 5, connectable segments are depicted by thick lines.

We denote the connectable segment in the top path, the left path, the right path and the bottom path of P by $\alpha_t(P)$, $\alpha_\ell(P)$, $\alpha_r(P)$ and $\alpha_b(P)$, respectively. An unconnectable line segment in the top-left path is called a *control segment* if it is incident to corner a. Similarly an unconnectable line segment in the right (bottom) path is called a *control segment* if it is incident to corner b (d). In Figs. 3, 4 and 5, control segments are depicted by dashed lines. We denote the control segment in the top path, the left path, the right path and the bottom path of P by $\beta_t(P)$, $\beta_\ell(P)$, $\beta_r(P)$ and $\beta_b(P)$, respectively. Let $\beta_{\max}(P)$ be a control segment whose length is maximum in P. A control segment e is called *convex* if both of the two interior angles of P at the both ends of e are 90 degree.

The *width* $w(P)$ of P is the distance from the leftmost vertical segment to the rightmost one, and the *height* $h(P)$ of P is the distance from the top horizontal segment to the bottom one. We denote by xy the *line segment* with end points x and y. We denote the length of segment xy by $|xy|$, the area of a polygon P by $A(P)$, and the sum of the areas specified for all inner faces of a plane graph G by $A(G)$. For a node u of a slicing tree T, we call the following condition the *size condition* of combined decagon P_u; $A(P_u) = A(G_u)$.

3.2 Outline of Algorithm

This subsection outlines our algorithm for slicing graphs with specified areas. The algorithm is a divide-and-conquer based on slicing trees. We are given a slicing graph G with specified areas, its slicing tree T, and rectangle P_r with corner vertices for the outer cycle of G. At this point, the positions of all vertices have not been determined yet. A vertex whose position is determined during the algorithm is called *fixed*. We first draw the outer cycle of G as the specified rectangle P_r, fixing the corner vertices. We then visit all internal nodes in T in preorder and slice P_r recursively to obtain an entire drawing of G. For a node u

of T, suppose that the outer cycle of G_u is to be drawn as a combined decagon P_u which satisfies the size condition.

Let u be a V-node. Then G_u has the vertical slicing path π_u, and let z_t and z_b be end vertices of π_u on the top and bottom path of G_u, respectively. First, we try to slice P_u into two combined decagons which satisfy the size condition by choosing a (unique) vertical straight-line segment L as its slicing path π_u (see Fig. 7). If L can be drawn correctly, i.e., the end points z_t and z_b of L are on $\alpha_t(P_u)$ and $\alpha_b(P_u)$, respectively, then we slice P_u by L to obtain two combined decagons. Otherwise, we split P_u by choosing a step-line as its slicing path π_u (see Fig. 7). We can show that the existence of such a suitable step-line π_u is ensured if P_u satisfies the size condition and "boundary condition," which will be described later (the detail of the proof is omitted due to space limitation).

The slicing procedure for H-nodes u is analogous with that for V-nodes. An entire drawing of the given slicing graph G will be constructed by applying the above procedure recursively. We call the algorithm described above Algorithm *Decagonal-Draw*.

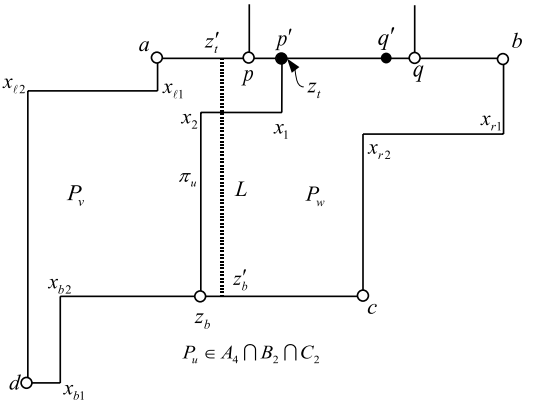


Fig. 7. Vertical slicing of P_u

To ensure that a combined decagon can be chosen as the polygon for the outer facial cycle of each subgraph G_u, the positions of end vertices z_t and z_b of π_u will be decided so that certain conditions are satisfied. We now describe these conditions.

For each node u of T, let f_u^t be the number of inner faces of G_u that are adjacent to the top path of G_u, and f_u^ℓ be the number of inner faces of G_u that are adjacent to the left path of G_u.

Let $a_{\min}$ be the minimum area of all areas for inner faces of G. Let W and H be the width and height of the rectangle specified for the outer facial cycle of a given slicing graph G. We define

$$\lambda = \frac{a_{\min}}{3f \cdot \max(W, H)}. \tag{1}$$

We define some conditions on combined decagon P_u.

A control segment e of P_u is called (λ, f)-*admissible* if one of the followings holds:

e is a convex and vertical segment, and $f_u^t\lambda \leq |e| < f\lambda$,
e is a convex and horizontal segment, and $f_u^\ell\lambda \leq |e| < f\lambda$,
e is a non-convex and vertical segment, and $|e| < (f - f_u^t)\lambda$,
e is a non-convex and horizontal segment, and $|e| < (f - f_u^\ell)\lambda$.

A combined decagon P_u is called (λ, f)-*admissible* if it satisfies the followings.

(a1) $|\alpha_t(P_u)| \geq f_u^t\lambda$,
(a2) $|\alpha_\ell(P_u)| \geq f_u^\ell\lambda$,
(a3) Every control segment of P_u is (λ, f)-admissible,
(a4) If $P_u \in A_1$, then $|\alpha_t(P_u)| \geq (f + f_u^t)\lambda$ or $|\alpha_\ell(P_u)| \geq (f + f_u^\ell)\lambda$,
(a5) If $P_u \in A_2 \cup A_4$, then $|\alpha_\ell(P_u)| + |\beta_\ell(P_u)| \geq (f + f_u^\ell)\lambda$,
(a6) If $P_u \in A_3 \cup A_5$, then $|\alpha_t(P_u)| + |\beta_t(P_u)| \geq (f + f_u^t)\lambda$,
(a7) If $P_u \in A_2 \cap B_3$, then $|\beta_\ell(P_u)| - |\beta_r(P_u)| \geq f_u^t\lambda$,
(a8) If $P_u \in A_3 \cap C_3$, then $|\beta_t(P_u)| - |\beta_b(P_u)| \geq f_u^\ell\lambda$,
(a9) If $P_u \in A_4 \cap B_2$, then $|\beta_r(P_u)| - |\beta_\ell(P_u)| \geq f_u^t\lambda$,
(a10) If $P_u \in A_5 \cap C_2$, then $|\beta_b(P_u)| - |\beta_t(P_u)| \geq f_u^\ell\lambda$.

By (λ, f)-admissibility of P_u, P_u is a simple polygon, and the distance of any pair of vertical line segments or any pair of horizontal line segments of P_u is at least λ.

For a combined decagon P_u, let a be the top-left corner vertex of P_u, b' be a fixed vertex which is the nearest to a on the top path of P_u, and d' be a fixed vertex which is the nearest to a on the left path of P_u. We call the following conditions the *boundary condition* of P_u.

(b1) If there exists fixed vertices on the top path of P_u, then these vertices are on $\alpha_t(P_u)$. The distance of any pair of fixed vertices on $\alpha_t(P_u)$ is at least $f_u^t\lambda$, and the distance from both ends of $\alpha_t(P_u)$ to any fixed vertex is at least $f_u^t\lambda$.
(b2) If there exists fixed vertices on the left path of P_u, then these vertices are on $\alpha_\ell(P_u)$. The distance of any pair of fixed vertices on $\alpha_\ell(P_u)$ is at least $f_u^\ell\lambda$, and the distance from both ends of $\alpha_\ell(P_u)$ to any fixed vertex is at least $f_u^\ell\lambda$.
(b3) If $P_u \in A_1$, then the distance from b' to the left path of P_u is greater than $(f + f_u^t)\lambda$ or the distance from d' to the top path of P_u is greater than $(f + f_u^\ell)\lambda$.
(b4) If $P_u \in A_2 \cup A_4$, then the distance from d' to the top path of P_u is greater than $(f + f_u^\ell)\lambda$.
(b5) If $P_u \in A_3 \cup A_5$, then the distance from b' to the left path of P_u is greater than $(f + f_u^t)\lambda$.

Let $\mathcal{D}$ be the set of all (λ, f)-admissible decagons that satisfy the boundary and size conditions.

The following lemma guarantees the correctness of the algorithm, whose proof can be found in the full version of the paper.

Lemma 1. *For a decagon $P_u \in \mathcal{D}$, let P_v and P_w be combined decagons generated by slicing P_u in Decagonal-Draw. Then P_v and P_w belong to $\mathcal{D}$.* □

By this lemma, we can prove the existence of 10-gonal drawings in Theorem 1.

Lemma 2. *Algorithm Decagonal-Draw finds a 10-gonal drawing of a slicing graph G with specified face areas correctly.*

Proof. Let P_r be a rectangle given as the boundary of G. Clearly P_r has no control segments and satisfies the size condition. Hence, P_r satisfies (λ, f)-admissibility. Since P_r satisfies the boundary condition, we have $P_r \in \mathcal{D}$. By Lemma 1, every face of G is drawn as a decagon in $\mathcal{D}$ recursively. Hence, algorithm Decagonal-Draw finds a 10-gonal drawing of a slicing graph G with specified face areas. □

It is not difficult to observe the time complexity of the algorithm.

Lemma 3. *Algorithm Decagonal-Draw can be implemented to run in $O(n)$ time and space.* □

Lemmas 2 and 3 prove Theorem 1.

4 Drawing of Rectangular Graphs

We assume that a given rectangular graph G has no 2-degree vertex except for its corner vertices. To utilize Theorem 1, we convert G into a slicing graph. For this, we slice some inner faces in G by adding new vertices and edges. By the result due to F. d'Amore and P. G. Franciosa [1], we can convert a rectangular graph G into a slicing graph G' by slicing each inner face c of G into at most 4 inner faces so that each unit path of c contains at most 2 unit paths of inner faces generated in the interior of c (see Fig. 8). A cycle c' in G' which corresponds to the facial cycle of c contains at most 8 unit paths of inner faces generated in the interior of c'. Figure 9(a) illustrates a inner face c of G sliced into 4 inner faces c_1, c_2, c_3 and c_4. Also, we obtain a slicing tree of the resulting slicing graph from the way of slicing the region by their slicing algorithm. When an inner face c of G is sliced into two inner faces c_1 and c_2, the area of each c_i is set to be a half of the area of c.

We apply Theorem 1 to the resulting slicing graph G' to obtain a 10-gonal drawing D', and remove the edges and vertices added to G from the drawing D'. In the resulting drawing of G, each inner face c is drawn as a polygon with at most 7 step-line segments and some straight-line segments since either the top path or the left path of a combined decagon is always drawn as a straight-line segment (see Fig. 9(b)). Hence c is drawn as a polygon with at most 14 corners of step-lines and 4 corners of the original rectangle. Then we obtain an 18-gonal drawing as described in Theorem 2.

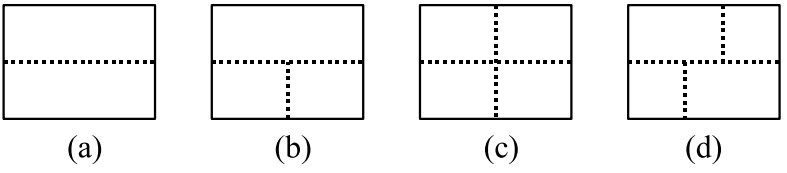


Fig. 8. Four slicing patterns of a rectangle

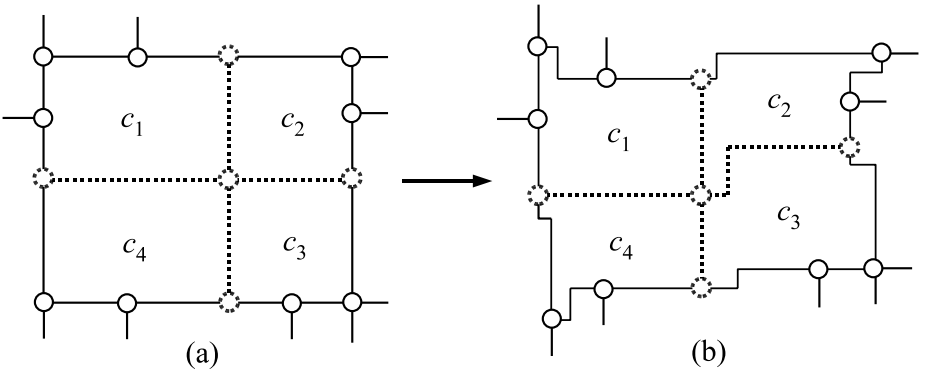


Fig. 9. (a) An inner face c sliced into four inner faces c_1, c_2, c_3 and c_4 in G'; (b) the drawing of c in D'

5 Drawing of 3-Connected Plane Graph

By the definition of a 3-connected plane graph, the degree of every vertex is at least three. First, we consider a 3-connected plane graph G with $\Delta(G) = 3$.

An *inner dual graph* G^* for G is a plane graph such that G^* has an embedded vertex in each inner face of G, and G^* has an edge (v, w) if and only if two inner faces c_v and c_w in G that correspond to the vertices v and w in G^* are adjacent. Then edge (v, w) in G^* intersects only one edge of G which separates faces c_v and c_w. Since the degree of each vertex of G is three, G^* is a triangulated plane graph. Note that G^* has no parallel edges since otherwise some two inner faces in G would share more than one edge as the boundary of each of them, contradicting the 3-connectivity of G.

C.-C. Liao et al. [4] proved that a plane graph G whose inner dual graph G^* is a triangulated plane graph without parallel edges has an orthogonal drawing where outer face of G is drawn as a rectangle and each inner face is drawn as a rectangle, an L-shape or a T-shape. For each inner face c which is drawn as an L-shape or a T-shape, we can slice c into 2 inner faces c_1 and c_2 drawn as rectangles. Hence by the slicing procedure described in the previous section, we can convert a 3-connected plane graph G into a slicing graph G' by slicing each inner face c of G into at most 8 inner faces. Let c', c'_1 and c'_2 be cycles in G' which correspond to the facial cycle of c, c_1 and c_2, respectively. Both c'_1 and c'_2

contains at most 8 unit paths of inner faces generated in their interior, and two of them are always in the interior of c'. Hence c' contains 14 unit paths of inner faces generated in the interior of c'. We apply Theorem 1 to slicing graph G' to obtain a 10-gonal drawing D', and remove the edges and vertices added to G from the drawing D'. In the resulting drawing of G, each inner face c is drawn as a polygon with at most 13 step-line segments and some straight-line segments. Hence c is drawn as a polygon with at most 26 corners of step-lines and at most 8 corners of the original shape (a rectangle, an L-shape or a T-shape). Then we can obtain a 34-gonal drawing of G, as claimed in Theorem 3.

Now, we consider a 3-connected plane graph G with $\Delta(G) = 4$. We can assume that there are no 4-degree vertices on the outer cycle of G. We replace each 4-degree vertex v of G with a cycle of length 4 introducing a new inner face c_v, and then convert the resulting graph into a slicing graph G'. Each inner face c of G is sliced into $8 + p_c$ inner faces in G', where p_c is the number of 4-degree vertices in the facial cycle of a inner face c (the detail of the proof is omitted due to space limitation).

Then we apply Theorems 1 and 3 to the slicing graph G'. A drawing of G in Corollary 1 can be obtained by modifying the drawing of G'.

6 Conclusion

In this paper, we showed that every slicing graph has a 10-gonal drawing, every rectangular graph has an 18-gonal drawing, and every 3-connected plane graph whose maximum degree is three has a 34-gonal drawing. We also gave a linear time algorithm to find a 10-gonal drawing for a slicing graph.

It is left as a future work to derive lower bounds on the number k such that every slicing graph admits a k-gonal drawing.

References

1. F. d'Amore and P. G. Franciosa: On the optimal binary plane partition for sets of isothetic rectangles, *Information Proc. Letters*, Vol. 44, pp. 255-259, 1992.
2. M. de Berg, E. Mumford and B. Speckmann: On rectilinear duals for vertex-weighted plane graphs. In *Proc 13th International Symposium on Graph Drawing*, pp. 61-72, 2005.
3. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis: Graph drawing: Algorithms for the visualization of graphs, Prentice hall, 1999.
4. C.-C. Liao, H.-I. Lu and H.-C. Yen: Floor-planning via orderly spanning trees, In *Proc. 9th International Symposium on Graph Drawing*, pp. 367-377, 2001.
5. T. Nishizeki, K. Miura and Md. S. Rahman: Algorithms for drawing plane graphs, *IEICE Trans. Electron.*, Vol.E87-D, No.2, pp.281-289, 2004.
6. M. S. Rahman, K. Miura and T. Nishizeki: Octagonal drawings of plane graphs with prescribed face areas, In *Graph Theoretic Concepts in Computer Science: 30th International Workshop*, Vol. 3353 of Lecture Notes in Computer Science, pp. 320-331, 2004.

Absolutely Non-effective Predicates and Functions in Computable Analysis*

Decheng Ding, Klaus Weihrauch, and Yongcheng Wu**

Department of Mathematics, Nanjing University, 210093 Nanjing, China
Department of Mathematics and Computer Science, University of Hagen, Germany
Nanjing University of Information Science and Technology, 210044 Nanjing, China
dcding@nju.edu.cn, Klaus.Weihrauch@FernUni-Hagen.de, ycwu@nuist.edu.cn

Abstract. In the representation approach (TTE) to Computable Analysis those representations of an algebraic or topological structure are of interest, for which the basic predicates and functions become computable. There are, however, natural examples of predicates and functions, which are not computable, even not continuous, for any representations. All these results follow from a simple lemma. In this article we prove this lemma and apply it to a number of examples. In particular we prove that various predicates and functions on computable measure spaces are not continuous for any representations, that means "absolutely non-effective".

Keywords: computability, undecidability, representation, measure, set.

1 Introduction

In the representation approach to Computable Analysis (TTE) [7] computability on finite words $w \in \Sigma^*$ and infinite sequences $p \in \Sigma^\omega$ of symbols from a finite alphabet Σ is defined explicitly, for example, by Turing machines. Computability on other sets M is introduced via naming systems, that is, notations $\nu :\subseteq \Sigma^* \to M$ or representations $\delta :\subseteq \Sigma^\omega \to M$ where finite or infinite sequences of symbols serve as "concrete names" of the "abstract" objects $x \in M$. The concept of computability induced on M depends crucially on the naming system. For a mere set M there is no criterion to select from the set of all naming systems the relevant or interesting ones.

If, however, a structure on M is given, for example, a topology or an algebra, those naming systems are of interest, for which the functions and predicates from the structure become computable. There are examples of structures, for which there is exactly one (up to equivalence) naming system with this property [5].

On the other hand, there are functions and predicates, which become computable for no representation. We may call them "absolutely non-computable".

* The work has been supported by NSFC (Natural Science Foundation of China) and DFG (Deutsche Forschungsgemeinschaft).

** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 595–604, 2007.
© Springer-Verlag Berlin Heidelberg 2007

For example, for no representation δ of the real numbers equality is (δ,δ)-decidable, for no representation of the closed subsets of Euclidean space intersection is $(\delta,\delta,\psi^<)$-computable, for no representation δ of a non-separable metric space the distance is $(\delta,\delta,\rho^>)$-computable [7]. In this article we prove a number of other results of this type.

The above theorems of the form "there is no representation such that ..." concern binary predicates or functions. We prove as a central lemma that every (γ,δ)-open set is a *countable* union of Cartesian products. So, if a set of pairs is not a *countable* union of Cartesian products it is not (γ,δ)-open for any representations γ,δ. Applying this lemma we prove for a variety of sets that they are not (γ,δ)-open and for some binary functions that they cannot be (γ,δ,ψ)-continuous for any representations γ,δ. Since no choice of representations can make them effective, such relations (for example equality on the real numbers) and functions can be called "absolutely non-effective". We obtain as immediate consequences that the sets cannot be (γ,δ)-r.e. or (γ,δ)-decidable and the functions cannot be (γ,δ,ψ)-computable for any representations γ,δ. Up to these consequences the results in this article are purely topological and require only the topological framework of TTE (Type 2 Theory of Effectivity [7]).

In Section 2 we summarize some basic definitions from TTE. In Section 3 we prove the central lemma on countable union of products and derive a number of "non-continuities" as consequences. In Section 4 we consider a σ-algebra $\mathcal{A}$ of measurable sets with infinite measures. which is generated by a σ-finite ring. We present a number of binary relations, which cannot be (γ,δ)-open, and several binary functions, which cannot be (γ,δ,ψ)-continuous for any representations γ,δ of the set $\mathcal{A}$.

2 Concepts from Computable Analysis

In this section we summarize some basic definitions from TTE, for details see [7]. Let Σ be a sufficiently large finite alphabet of symbols. Let Σ^* be the set of all finite strings and Σ^ω the set of all infinite sequences over Σ. For $w\in\Sigma^*$ let $w\Sigma^\omega\subseteq\Sigma^\omega$ be the set of all $p\in\Sigma^\omega$ such that w is a prefix of p. A partial function $f:\subseteq Y_1\times\ldots\times Y_k\to Y_0$ $(Y_0,\ldots,Y_k\in\{\Sigma^*,\Sigma^\omega\})$ is computable, if it can be computed by a Type-2 machines, i.e., a Turing machine with k input tapes and a one-way output tape for possibly one-way infinite inscriptions. For $(y_1,\ldots,y_k)\in Y_1\times\ldots\times Y_k$, $f(y_1,\ldots,y_k)=y_0$, iff on input $(y_1,\ldots,y_k)$ the machine halts with result $y_0\in\Sigma^*$ or the machine computes forever and writes $y_0\in\Sigma^\omega$, if $Y_0=\Sigma^*$ or $Y_0=\Sigma^\omega$, respectively. A set $Z\subseteq Y_1\times\ldots\times Y_k$ is recursively enumerable (r.e.), if $Z=\mathrm{dom}(f)$ for some computable function $f:\subseteq Y_1\times\ldots\times Y_k\to\Sigma^*$. Let $\iota:\Sigma^*\to\Sigma^*$, $a_1\ldots a_k\mapsto 110a_10\ldots0a_k011$ be the "wrapping function". The "pairing function" $\pi:\Sigma^\omega\times\Sigma^\omega\to\Sigma^\omega$,

$$\pi(a_0a_1\ldots,\,b_0b_1\ldots):=a_0b_0a_1b_1\ldots\quad(a_i,b_i\in\Sigma)$$

is bijective and computable and the projections of its inverse are computable. As usual we write $\langle p,q\rangle=\pi(p,q)$. The straightforward generalization of π to more than two arguments is denoted by $\langle\,\rangle$ as well.

On Σ^* we consider the discrete topology. On Σ^ω we consider Cantor topology generated by the base $\{w\Sigma^\omega \mid w \in \Sigma^*\}$ and on $(\Sigma^\omega)^k$ its product topology generated by the base $\{v_1\Sigma^\omega \times \ldots \times v_k\Sigma^\omega \mid v_1, \ldots, v_k \in \Sigma^*\}$. Every partial computable function $f : \subseteq Y_1 \times \ldots \times Y_k \to Y_0$ is continuous. Every r.e. set $Z \subseteq Y_1 \times \ldots \times Y_k$ is open.

A naming system of a set M is a surjective function $\delta : \subseteq Y \to M$ where $Y = \Sigma^*$ (notation) or $Y = \Sigma^\omega$ (representation). Let $\nu_{\mathbb{N}}$ and $\nu_{\mathbb{Q}}$ be canonical notations of the natural and the rational numbers, respectively. Let ρ be the standard representation of the set of real numbers $\mathbb{R}$ and let $\overline{\rho}_>$ be the upper representation of $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, \infty\}$ defined by: $\overline{\rho}_>(p) := \inf\{\nu_{\mathbb{Q}}(w) \mid \iota(w)$ is a subword of p and $w \in \mathrm{dom}(\nu_{\mathbb{Q}})\}$. Let $\rho_>$ be the restriction of $\overline{\rho}_>$ to $\mathbb{R}$.

For naming systems $\delta_i : \subseteq Y_i \to M_i$, a function $h : \subseteq Y_1 \times \ldots \times Y_k \to Y_0$ realizes a function $f : \subseteq M_1 \times \ldots \times M_k \to M_0$, with respect to $(\delta_1, \ldots, \delta_k, \delta_0)$, iff

$$f(\delta_1(y_1), \ldots, \delta_k(y_k)) = \delta_0 \circ h(y_1, \ldots, y_k)$$

whenever $f(\delta_1(y_1), \ldots, \delta_k(y_k))$ exists. The function f is $(\delta_1, \ldots, \delta_k, \delta_0)$-continuous (-computable), iff it has a continuous (computable) $(\delta_1, \ldots, \delta_k, \delta_0)$-realization. A set $X \subseteq M_1 \times \ldots \times M_k$ is called $(\delta_1, \ldots, \delta_k)$-open (r.e.), iff there is some open (r.e.) set $U \subseteq Y_1 \times \ldots \times Y_k$ such that

$$\{(y_1, \ldots, y_k) \mid (\delta_1(y_1), \ldots, \delta_k(y_k)) \in X\} = U \cap \mathrm{dom}(\delta_1) \times \ldots \times \mathrm{dom}(\delta_k)\,. \quad (1)$$

X is called $(\delta_1, \ldots, \delta_k)$-decidable, if X and its complement are $(\delta_1, \ldots, \delta_k)$-r.e.

For representations $\delta_1, \ldots, \delta_k$ the tupling $[\delta_1, \ldots, \delta_k] : \subseteq \Sigma^\omega \to M_1 \times \ldots \times M_k$ is defined by $[\delta_1, \ldots, \delta_k]\langle y_1, \ldots, y_k\rangle := (\delta_1(y_1), \ldots, \delta_k(y_k))$. In this article we need the definitions mainly for the cases $k = 1$ or $k = 2$. For representations $\gamma : \subseteq \Sigma^\omega \to M$, $\delta : \subseteq \Sigma^\omega \to M'$ and $\psi : \subseteq \Sigma^\omega \to M''$, and for $X \subseteq M \times M'$ and $f : \subseteq M \times M' \to M''$,

$$X \text{ is } (\gamma, \delta)\text{-decidable} \implies X \text{ is } (\gamma, \delta)\text{-r.e.} \implies X \text{ is } (\gamma, \delta)\text{-open}\,, \quad (2)$$

$$X \text{ is } (\gamma, \delta)\text{-PP} \iff X \text{ is } [\gamma, \delta]\text{-PP (for PP} \in \{\text{ decidable, r.e., open}\}), \quad (3)$$

$$f \text{ is } (\gamma, \delta, \psi)\text{-PP} \iff f \text{ is } ([\gamma, \delta], \psi)\text{-PP (for PP} \in \{\text{ computable, open}\}) \quad (4)$$

[7]. We will apply these properties without further mentioning. By $\delta|^N$ we denote the restriction of δ to range N.

Lemma 1. *Let $\delta : \subseteq \Sigma^\omega \to M$ and $\delta' : \subseteq \Sigma^\omega \to M'$ be representations and let $X \subseteq M$ be δ-open.*

1. *If $f : M' \to M$ is (δ', δ)-continuous then $f^{-1}[X]$ is δ'-open.*
2. *For any $N \subseteq M$, $X \cap N$ is $\delta|^N$-open.*

Proof. 1. Let $\gamma : A \to M$ and $\gamma' : A' \to M'$ be the restrictions of δ and δ' to $\mathrm{dom}(\delta)$ and $\mathrm{dom}(\delta')$, respectively. We consider the induced topologies on A and

A'. There is some continuous function $h : A' \to A$ such that $f\gamma' = \gamma h$. (Since γ, γ', f and h are total functions we can apply the usual transformation rules.) Then

$$\begin{aligned} X \text{ is } \delta\text{-open} &\implies \gamma^{-1}[X] \text{ is open (in } A) \implies h^{-1}\gamma^{-1}[X] \text{ is open} \\ &\implies (\gamma h)^{-1}[X] \text{ is open} \quad \implies (f\gamma')^{-1}[X] \text{ is open} \\ &\implies (\gamma')^{-1}f^{-1}[X] \text{ is open} \implies f^{-1}[X] \text{ is } \delta'\text{-open}\,. \end{aligned}$$

2. The embedding $f : N \to M$ is $(\delta|^N, \delta)$-continuous (realized by the identity on Σ^ω). Furthermore, $f^{-1}[X] = X \cap N$. Apply 1. for $\delta' := \delta|^N$. □

3 Absolutely Non-open Sets

The main results of this article are based on the following observation.

Lemma 2. *Let $\gamma : \subseteq \Sigma^\omega \to M$ and $\delta : \subseteq \Sigma^\omega \to M'$ be representations. Then for every non-empty (γ, δ)-open set $X \subseteq M \times M'$ there are sequences $(U_k)_{k\in\mathbb{N}}$ and $(V_k)_{k\in\mathbb{N}}$ of non-empty sets such that*

$$X = \bigcup_{k\in\mathbb{N}} U_k \times V_k\,. \tag{5}$$

Proof. Since X is (γ, δ)-open there is an open set $W \subseteq \Sigma^\omega \times \Sigma^\omega$ such that $\{(p, q) \mid (\gamma(p), \delta(q)) \in X\} = W \cap \mathrm{dom}(\gamma) \times \mathrm{dom}(\delta)$, therefore, $X = \{(\gamma(p), \delta(q)) \mid (p, q) \in W \cap \mathrm{dom}(\gamma) \times \mathrm{dom}(\delta)\}$. Since W is open and $X \neq \emptyset$, there are words u_k, v_k such that $W = \bigcup_{k\in\mathbb{N}} u_k\Sigma^\omega \times v_k\Sigma^\omega$, hence $X = \bigcup_{k\in\mathbb{N}} \gamma(u_k\Sigma^\omega) \times \delta(v_k\Sigma^\omega)$. Since $X \neq \emptyset$, $\gamma(u_k\Sigma^\omega) \times \delta(v_k\Sigma^\omega) \neq \emptyset$ for some $k \in \mathbb{N}$. Delete all empty elements from the family $(\gamma(u_k\Sigma^\omega) \times \delta(v_k\Sigma^\omega))_{k\in\mathbb{N}}$. If the resulting family is still infinite, we are finished. Otherwise, fill it up to a countable one by repeating one of its elements. □

By Equation (5) X is the *countable* union of Cartesian products. The lemma can be generalized straightforwardly to more than 2 factors.

Using Lemma 2 we will prove that several sets are not (γ, δ)-open and functions f are not (γ, δ, ψ)-continuous for *all* representations γ and δ. By Formulas (2, 3, 4) these results have a number of obvious implications, which we will not mention repeatedly. Many applications of Lemma 2 follow from the next lemma.

Lemma 3. *1. Let $\sim \subseteq M \times M$ be an equivalence relation such that the set $M/_\sim$ of equivalence classes is uncountable. Let $\prec \subseteq M \times M$ be an relation on M such that*

$$a \prec a \quad \text{and} \quad (a \prec b \quad \text{and} \quad b \prec a \implies a \sim b) \tag{6}$$

for all $a, b \in M$. Then there are no representations γ, δ of M such that $\prec$ is (γ, δ)-open.

2. Let $\sim \subseteq M \times M$ be an equivalence relation such that $M/_\sim$ is uncountable. Then there are no representations γ, δ of M such that $\sim$ is (γ, δ)-open.

Proof. 1. Suppose the relation $\prec$ is (γ,δ)-open. Then it can be written as $\bigcup_{k\in\mathbb{N}} U_k \times V_k$ with non-empty sets U_k, V_k. For any $a \in M$, $(a,a) \in \prec$ by assumption. Suppose, $(a,a),(b,b) \in U_k \times V_k$. Then $(a,b),(b,a) \in U_k \times V_k \subseteq \prec$, hence $a \sim b$. Therefore, for every $k \in \mathbb{N}$ there is some a such that

$$\{b \mid (b,b) \in U_k \times V_k\} \subseteq a/_\sim$$

and hence

$$\{b \mid (b,b) \in \bigcup_{k\in\mathbb{N}} U_k \times V_k\} \subseteq \bigcup \{a/_\sim \mid a \in A\}$$

for some countable set $A \subseteq M$. But

$$\{b \mid (b,b) \in \prec\} = M,$$

which is not a countable union of $\sim$-equivalence classes by assumption. Therefore, $\sim \neq \bigcup_{k\in\mathbb{N}} U_k \times V_k$, and so $\sim$ cannot be (γ,δ)-open.

2. Apply the first case with $\prec := \sim$. □

Let $A \Delta B := A \setminus B \cup B \setminus A$ be the symmetric difference of the sets A and B. The following theorem summarizes some more concrete results.

Theorem 4. *1. For an uncountable set M there are no representations γ,δ such that* $\mathrm{Eq} = \{(x,y) \mid x,y \in M, \quad x = y\}$ *is (γ,δ)-open.*

2. For the set $\mathbb{R}$ of real numbers there are no representations γ,δ such that $\mathrm{Leq} := \{(x,y) \mid x \leq y\}$ *is (γ,δ)-open.*

3. For an uncountable collection $\mathcal{S}$ of sets there are no representations γ,δ such that $\mathrm{Sub} := \{(A,B) \mid A,B \in \mathcal{S}, \quad A \subseteq B\}$ *is (γ,δ)-open.*

4. For a collection $\mathcal{S}$ of sets containing uncountably many singletons (i.e., the one-element sets) there are no representations γ,δ such that $\mathrm{Nei} := \{(A,B) \mid A,B \in \mathcal{S}, \quad A \cap B \neq \emptyset\}$ *is (γ,δ)-open.*

5. For each of the following binary relations T, "$A \cap B = \emptyset$", "$A \cap B$ is finite", "$A \setminus B = \emptyset$", "$A \setminus B$ is finite", "$A \Delta B = \emptyset$", "$A \Delta B$ is finite", on $2^{\mathbb{N}}$ there are no representations γ,δ of $2^{\mathbb{N}}$ such that T is (γ,δ)-open.

Proof. 1. Eq is an equivalence relation. Apply Lemma 3.2.

2. Apply Lemma 3.1 with $\leq$ and $=$ for $\prec$ and $\sim$, respectively.

3. Apply Lemma 3.1 with $\subseteq$ and $=$ for $\prec$ and $\sim$, respectively.

4. If Nei is (γ,δ)-open, then the uncountably many pairs $(\{x\},\{x\})$ must be elements of a countable union $\bigcup_i X_i \times Y_i$. If $(\{x\},\{x\}) \in X_i \times Y_i$ and $(\{y\},\{y\}) \in X_i \times Y_i$, then $(\{x\},\{y\}) \in X_i \times Y_i$, hence $x = y$. Therefore each $X_i \times Y_i$ contains at most one pair $(\{x\},\{x\})$. But there are uncountably many such pairs.

5. Let $\pi : \mathbb{N}^2 \to \mathbb{N}$ be a bijection. For $i \in \mathbb{N}$ let $C_i := \{\pi(i,j) \mid j \in \mathbb{N}\}$ and for $L \subseteq \mathbb{N}$ $(L \neq \emptyset, \mathbb{N})$ let $A_L := \bigcup_{i\in L} C_i$. Then all the A_L constitute an uncountable family of infinite sets such that A_K and A_L differ by an infinite set if $K \neq L$. If T is (γ,δ)-open, then by Lemma 2 it can be written as $T = \bigcup_{i\in\mathbb{N}} X_i \times Y_i$.

"$A \cap B = \emptyset$": For each L, $A_L \cap A_L^c = \emptyset$. If $(A_K, A_K^c),(A_L, A_L^c) \in X_i \times Y_i$, then $A_K \cap A_L^c = \emptyset$ and $A_L \cap A_K^c = \emptyset$, hence $A_K = A_L$. Therefore, each $X_i \times Y_i$ contains at most one pair (A_L, A_L^c). But the set of such pairs is uncountable.

"$A \cap B$ is finite": Proceed as above and observe that $A_K \cap A_L^c = \emptyset$ if $A_K \cap A_L^c$ is finite.

"$A/B = \emptyset$": For each L, $A_L \setminus A_L = \emptyset$. If $(A_K, A_K), (A_L, A_L) \in X_i \times Y_i$, then $A_K \setminus A_L = \emptyset$ and $A_L \setminus A_K = \emptyset$, hence $A_K = A_L$. Therefore, each $X_i \times Y_i$ contains at most one pair (A_L, A_L). But the set of such pairs is uncountable.

"$A \setminus B$ is finite": Proceed as above and observe that $A_K \setminus A_L = \emptyset$ if $A_K \setminus A_L$ is finite.

"$A \Delta B = \emptyset$": For each L, $A_L \Delta A_L = \emptyset$. If $(A_K, A_K), (A_L, A_L) \in X_i \times Y_i$, then $A_K \Delta A_L = \emptyset$, hence $A_K = A_L$. Therefore, each $X_i \times Y_i$ contains at most one pair (A_L, A_L). But the set of such pairs is uncountable.

"$A \Delta B$ is finite": Proceed as above and observe that $A_K \Delta A_L = \emptyset$ if $A_K \Delta A_L$ is finite. □

Theorem 4.1 applies, for example, to the real numbers (Thm 4.1.16 in [7]), to the powerset $2^{\mathbb{N}}$ of $\mathbb{N}$, to the various subset spaces of Euclidean space $\mathbb{R}^n$ and to spaces of continuous functions (Chapters 5.6 in [7]). Notice, however, that *inequality* may become r.e., for example, for the standard representation ρ of the real numbers $\{(x, y) \in \mathbb{R}^2 \mid x \neq x\}$ is (ρ, ρ)-r.e. Theorem 4.3 can be applied to $2^{\mathbb{N}}$ and to the open, the closed and the compact subsets of $\mathbb{R}^{\mathbb{N}}$. Theorem 4.4 can be applied to the closed and the compact subsets of $\mathbb{R}^{\mathbb{N}}$.

Every represented metric space with continuous distance is separable, that is, it has a countable dense subset. The proofs in [6] and of Lemma 8.1.1 in [7] use essentially Lemma 2. Here we prove a slight generalization.

Theorem 5. *Let γ, δ be representations of a metric space (M, d) such that the distance $d : M \times M \to \mathbb{R}$ is $(\gamma, \delta, \rho_>)$-continuous. Then the space is separable.*

Proof. For each $n \in \mathbb{N}$ define an open subset of Σ^ω by

$$U_n := \bigcup \{u\iota(v)\Sigma^\omega \mid u, v \in \Sigma^*,\ v \in \mathrm{dom}(\nu_\mathbb{Q}) \ \text{ and } \ \nu_\mathbb{Q}(v) < 2^{-n}\},$$

where recall that ι is the "wrapping function", and let $W_n := \{x \in \mathbb{R} \mid x < 2^{-n}\}$. Then $\rho_>^{-1}[W_n] = U_n \cap \mathrm{dom}(\rho_>)$, hence W_n is $\rho_>$-open. By assumption, d is $([\gamma, \delta], \rho_>)$-continuous. By Lemma 1.1, $d^{-1}[W_n]$ is (γ, δ)-open and so by Lemma 2 there are (non-empty) sets $X_k^n, Y_k^n \subseteq M$ such that

$$\{(a, b) \in M \times M \mid d(a, b) < 2^{-n}\} = d^{-1}[W_n] = \bigcup_{k \in \mathbb{N}} X_k^n \times Y_k^n\,.$$

For every $n, k \in \mathbb{N}$ choose some $x_k^n \in X_k^n$. Suppose $x \in M$ and $n \in \mathbb{N}$. Then for some k, $(x, x) \in X_k^n \times Y_k^n$, hence $(x_k^n, x) \in X_k^n \times Y_k^n$, that is, $d(x_k^n, x) < 2^{-n}$. Therefore, the countable set $\{x_k^n \mid n, k \in \mathbb{N}\}$ is dense in (M, d). □

Notice that d is $(\gamma, \delta, \rho_>)$-continuous if d is (γ, δ, ρ)-continuous.

By Theorem 5.1.3 in [7] intersection on the closed subsets of $\mathbb{R}^n$ is not $(\psi, \psi, \psi_<)$-continuous. We can conclude this from the observation that the set of non-empty closed subsets of $\mathbb{R}^n$ is $\psi_<$-open. More generally, from Theorem 4.4 we obtain:

Corollary 6. *Let ψ be a representation of a collection $\mathcal{S}$ of sets containing uncountably many singletons such that $\{A \in \mathcal{S} \mid A \neq \emptyset\}$ is ψ-open. Then there are no representations γ, δ such that intersection on $\mathcal{S}$ is (γ, δ, ψ)-continuous.*

Proof. Suppose, intersection on $\mathcal{S}$ is $([\gamma, \delta], \psi)$-continuous. Since Nem $:= \{A \in \mathcal{S} \mid A \neq \emptyset\}$ is ψ-open, $\cap^{-1}[\mathrm{Nem}] = \{(A, B) \in \mathcal{S} \times \mathcal{S} \mid A \cap B \neq \emptyset\} = \mathrm{Nei}$ is (γ, δ)-open by Lemma 1.1. But this is false by Theorem 4.4. □

The corollary is applicable to the set of closed subsets of an uncountable T_1-space since for every point x the set $\{x\}$ is closed [4].

4 Applications to Computable Measure Theory

Computable measure spaces have been introduced in [10]. We recall the main definitions. As references to Measure Theory see [1], [2] and [3].

A *ring* in a set Ω is a set $\mathcal{R}$ of subsets of Ω such that $\emptyset \in \mathcal{R}$ and $A \cup B \in \mathcal{R}$ and $A \setminus B \in \mathcal{R}$ if $A, B \in \mathcal{R}$. A *σ-algebra* in Ω is a set $\mathcal{A}$ of subsets of Ω such that $\Omega \in \mathcal{A}$, $\Omega \setminus A \in \mathcal{A}$ if $A \in \mathcal{A}$ and $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$, if $A_1, A_2, \ldots \in \mathcal{A}$. For any system $\mathcal{E}$ of subsets of Ω let $\sigma(\mathcal{E})$ be the smallest σ-algebra in Ω containing $\mathcal{E}$. A *premeasure* on a ring $\mathcal{R}$ is a function $\mu : \mathcal{R} \to \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ such that $\mu(\emptyset) = 0$, $\mu(A) \geq 0$ for $A \in \mathcal{R}$ and

$$\mu(\bigcup_{i=1}^{\infty} A_n) = \sum_{i=1}^{\infty} \mu(A_n)$$

if $A_1, A_2, \ldots \in \mathcal{A}$ are pairwise disjoint and $\bigcup_{i=1}^{\infty} A_n \in \mathcal{A}$. A premeasure μ on a σ-algebra $\mathcal{A}$ is called a *measure*. In this case, $(\Omega, \mathcal{A}, \mu)$ is called a *measure space*.

Definition 7. *A* computable measure space *is a quintuple* $\mathcal{M} = (\Omega, \mathcal{A}, \mu, \mathcal{R}, \alpha)$ *such that*

1. *$(\Omega, \mathcal{A}, \mu)$ is a measure space,*
2. *$\mathcal{R}$ is a countable ring such that $\mathcal{A} = \sigma(\mathcal{R})$, $\Omega = \bigcup \mathcal{R}$, and $\mu(A) < \infty$ for all $A \in \mathcal{R}$,*
3. *$\alpha : \subseteq \Sigma^* \to \mathcal{R}$ is a notation of $\mathcal{R}$ with recursive domain such that union and intersection are (α, α, α)-computable and μ is (α, ρ)-computable on $\mathcal{R}$.*

Since our negative results concern only continuity, we do not need the third axiom on computability. Therefore, in the following let $\mathcal{M} = (\Omega, \mathcal{A}, \mu, \mathcal{R})$ be a measure space such that 1. and 2. hold true. Since most of the non-effectivities occur for sets with infinite measure, We assume $\mu(\Omega) = \infty$. By Definition, $\Omega = \bigcup \mathcal{R}$. The whole set Ω can be exhausted also by a sequence of pairwise disjoint ring elements each of which has measure $\mu(D) > 1$ and by a sequence of pairwise disjoint algebra elements each of which has measure $\mu(G) = \infty$.

Lemma 8. *1. There is a sequence $(D_i)_{i \in \mathbb{N}}$ of ring elements such that $\mu(D_i) > 1$ and $D_i \cap D_j = \emptyset$ for $i \neq j$ and $\bigcup_i D_i = \Omega$.*

2. *There is a sequence* $(G_i)_{i\in\mathbb{N}}$ *of algebra elements such that* $\mu(G_i) = \infty$ *and* $G_i \cap G_j = \emptyset$ *for* $i \neq j$ *and* $\bigcup_i G_i = \Omega$.

Proof. 1. Let $B_0, B_1, \ldots$ be a numbering of $\mathcal{R}$. Define $C_i := B_i \setminus (B_0 \cup \ldots \cup B_{i-1})$. Then for $j < i$, $B_j \cap C_i = \emptyset$, hence $C_j \cap C_i = \emptyset$. By induction, $B_0 \cup \ldots \cup B_i = C_0 \cup \ldots \cup C_i$. Therefore, $\bigcup_i C_i = \bigcup_i B_i = \Omega$. There are numbers $0 = i_0 < i_1 < i_2 < \ldots$ such that $\mu \bigcup \{C_m \mid i_k \leq m < i_{k+1}\} > 1$ for all k. Define $D_k := \bigcup \{C_m \mid i_k \leq m < i_{k+1}\}$.

2. Let $\pi : \mathbb{N}^2 \to \mathbb{N}$ be a bijection. Define $G_i := \bigcup_{j\in\mathbb{N}} D_{\pi(i,j)}$. ($D_i$ from 1.) □

The proof can easily be changed to a computable version of the lemma, which, however, we do not need in this article. Concrete representations of the set $\mathcal{A}$ of a computable measure space have been discussed in [8,9].In the following we list several properties, which cannot be effective for any representations of this set.

For sets A, B let $A \Delta B := (A \setminus B) \cup (B \setminus A)$ be their symmetric difference. Let $\mathcal{A}_0$ be the set of all $A \in \mathcal{A}$ with finite measure and let $\mathcal{A}_{\infty\infty}$ be the set of all $A \in \mathcal{A}$ such that A and its complement A^c have infinite measure. Let $(G_i)_i$ be the fixed sequence from Lemma 8.2. For

$$G_L := \bigcup_{i\in L} G_i, \quad (L \subseteq \mathbb{N}),$$

$$\mu(G_L) = 0 \iff L = \emptyset, \tag{7}$$

$$G_L \in \mathcal{A}_{\infty\infty} \text{ for } L \notin \{\emptyset, \mathbb{N}\}, \tag{8}$$

$$G_L^c = G_{L^c} \text{ and } G_K \diamond G_L = G_{K \diamond L} \text{ for binary boolean operations } \diamond. \tag{9}$$

Since our ring $\mathcal{R}$ is countable, the cardinality of the generated σ-algebra $\mathcal{A} = \sigma(\mathcal{R})$ is not greater than the cardinality of Σ^ω. Therefore, the set $\mathcal{A}$ has a representation. We list a number of relations defined by the measure on the σ-Algebra which are not open for any representations.

Theorem 9. *Let* $E \in \mathcal{A}$ *such that* $\mu(E) < \infty$. *In each of the following 9 cases of* $Q(A, B)$ *there are no representations* γ, δ *of* $\mathcal{A}$ *such that*

$$X := \{(A, B) \mid A, B \in \mathcal{A}, \quad Q(A, B)\} \quad \textit{is } (\gamma, \delta)\textit{-open}:$$

1. $\mu(A \setminus B) = 0$, $\mu(A \setminus B \setminus E) = 0$, $\mu(A \setminus B) < \infty$.
2. $\mu(A \Delta B) = 0$, $\mu((A \Delta B) \setminus E) = 0$, $\mu(A \Delta B) < \infty$,
3. $\mu(A \cap B) = 0$, $\mu((A \cap B) \setminus E) = 0$, $\mu(A \cap B) < \infty$.

The 9 statements hold accordingly for representations γ, δ *of* $\mathcal{A}_{\infty\infty}$ *and the sets* $Y := \{(A, B) \mid A, B \in \mathcal{A}_{\infty\infty}, \quad Q(A, B)\}$.

Proof. First we prove the theorem for sets from the universe $\mathcal{B} := \{G_L \mid L \subseteq \mathbb{N}, \ L \neq \emptyset \text{ and } L \neq \mathbb{N}\} \subseteq \mathcal{A}_{\infty\infty} \subseteq \mathcal{A}$. Let γ', δ' be representations of $\mathcal{B}$.

1. **Case** $\mu(A \setminus B) = 0$**:** Let $A \prec B \iff \mu(A \setminus B) = 0$ and $A \sim B \iff \mu(A \Delta B) = 0$. Then (6) is true and the set of $\sim$-equivalence classes is uncountable

since G_K and G_L are not equivalent, if $K \neq L$. By Lemma 3.1, $\{(A,B) \in \mathcal{B}\times\mathcal{B} \mid \mu(A \setminus B) = 0\}$ is not (γ', δ')-open.

Case $\mu(A \setminus B \setminus E) = 0$**:** Let $A \prec B \iff \mu(A \setminus B \setminus E) = 0$ and $A \sim B \iff \mu(A \Delta B) = 0$. Since $\mu(A \setminus B \setminus E) = 0$ implies $\mu(A \setminus B) = 0$ for $A, B \in \mathcal{B}$, we can continue as in the first case.

Case $\mu(A \setminus B) < \infty$**:** Let $A \prec B \iff \mu(A \setminus B) < \infty$. and $A \sim B \iff \mu(A \Delta B) = 0$. Since $\mu(A \setminus B) < \infty$ implies $\mu(A \setminus B) = 0$ for $A, B \in \mathcal{B}$, we can continue as in the first case.

2. In each of the three subcases replace "\" by "Δ" in 1.

3. We apply Lemma 2.

Case $\mu(A \cap B) = 0$**:** Assume that the set X is (γ', δ')-open. Then there are sets U_k, V_k such that $X = \bigcup_k U_k \times V_k$. For each $L \notin \{\emptyset, \mathbb{N}\}$, $(G_L, G_L^c) \in X$. Suppose $(G_K, G_K^c), (G_L, G_L^c) \in U_k \times V_k$. Then $\mu(G_K \cap G_L^c) = 0$, hence $K \cap L^c = \emptyset$ and correspondingly $L \cap K^c = \emptyset$, hence $K = L$. Since there are uncountably many pairs (G_K, G_K^c), countably many Cartesian products cannot cover them. Contradiction.

Case $\mu((A \cap B) \setminus E) = 0$**:** Proceed as in the case $\mu(A \cap B) = 0$. Notice that $\mu(G_K \cap G_L^c \setminus E) = 0$ implies $K = L$.

Case $\mu(A \cap B) < \infty$**:** Proceed as in the case $\mu(A \cap B) = 0$. Notice that $\mu(G_K \cap G_L^c) < \infty$ implies $K = L$.

The 9 results for the sets X and for the sets Y follow by Lemma 1.2: Suppose, X is (γ, δ)-open, hence $[\gamma, \delta]$-open. By Lemma 1.2 for $N := \mathcal{B}\times\mathcal{B}$, the set $X \cap \mathcal{B}\times\mathcal{B}$ is $[\gamma, \delta]|^{\mathcal{B}\times\mathcal{B}}$-open, hence $(\gamma|^{\mathcal{B}}, \delta|^{\mathcal{B}})$-open. But this is impossible as we have shown. The argument for the sets Y is similar. □

In measure theory usually sets which differ by a set of measure 0 are not distinguished. For $A, B \in \mathcal{A}$ let $A \equiv B \iff \mu(A \Delta B) = 0$. Then $\equiv$ is an equivalence relation. If $A \equiv B$ and $A' \equiv B'$ then $A^c \equiv B^c$ and $A \diamond A' \equiv B \diamond B'$ for every binary boolean operation $\diamond$. Therefore, on the set of equivalence classes, the factorization $\mathcal{A}/_{\equiv}$, the boolean functions are well-defined. Furthermore, a measure $\mu/_{\equiv}$ on $\mathcal{A}/_{\equiv}$ is well-defined by $\mu/_{\equiv}(A/_{\equiv}) := \mu(A)$. We will consider a modification of Theorem 9 where $\mathcal{A}$ and μ are replaced by $\mathcal{A}/_{\equiv}$ and $\mu/_{\equiv}$, respectively. We will apply the following lemma.

Lemma 10. *For every representation $\delta : \subseteq \Sigma^\omega \to \mathcal{A}/_{\equiv}$ there is a representation $\beta : \subseteq \Sigma^\omega \to \mathcal{A}$ such that the factorization* fac $: A \mapsto A/_{\equiv}$ *is (β, δ)-computable.*

Proof. Let $\gamma \subseteq \Sigma^\omega \to \mathcal{A}$ be any representation of $\mathcal{A}$. Define a representation β by

$$\beta\langle p, q\rangle = A :\iff \gamma(p) = A \text{ and } \gamma(p) \in \delta(q)\,.$$

Then the computable function $\langle p, q\rangle \mapsto q$ realizes the factorization w.r.t. (β, δ). □

Corollary 11. *Theorem 9 holds true accordingly for $\mathcal{A}/_{\equiv}$ and $\mu/_{\equiv}$ replacing $\mathcal{A}$ and μ.*

Proof. As an example consider the set

$$X = \{(R, S) \mid R, S \in \mathcal{A}/_{\equiv} \text{ and } \mu/_{\equiv}(R \cap S) < \infty\}\,.$$

Assume that there are representations γ, δ such that X is (γ, δ)-open, hence $[\gamma, \delta]$-open. By Lemma 10 there are representations γ', δ' of $\mathcal{A}$ such that factorization is (γ', γ)-computable and (δ', δ)-computable. Therefore, the total function $F : (A, B) \mapsto (A/_{\equiv}, B/_{\equiv})$ is $([\gamma', \delta'], [\gamma, \delta])$-continuous. By Lemma 8, $F^{-1}[X]$ is $[\gamma', \delta']$-open, hence (γ', δ')-open. But since

$$\begin{aligned} F^{-1}[X] &= \{(A, B) \mid (A/_{\equiv}, B/_{\equiv}) \in X\} \\ &= \{(A, B) \mid \mu_{\equiv}(A/_{\equiv} \cap B/_{\equiv}) < \infty\} \\ &= \{(A, B) \mid \mu(A \cap B) < \infty\,,\} \end{aligned}$$

$F^{-1}[X]$ cannot be (γ', δ')-open by Theorem 9. Contradiction.

The proofs for the other 8 cases are similar. Also the proofs for the cases $\mathcal{A}_{\infty\infty}$ are similar. □

References

1. Heinz Bauer. *Wahrscheinlichkeitstheorie und Grundzüge der Maßtheorie*. Walter de Gruyter, Berlin, 1974.
2. Heinz Bauer. *Measure and Integration Theory*. Walter de Gruyter, Berlin, 2001.
3. Donald L. Cohn. *Measure Theory*. Birkhäuser, Boston, 1980.
4. Ryszard Engelking. *General Topology*. Vol.6 of Sigma series in pure mathematics. Heldermann, Berlin, 1989.
5. Peter Hertling. A real number structure that is effectively categorical. *Mathematical Logic Quarterly*,45(2):147–182, 1999.
6. Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113:191–210, 1993.
7. Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.
8. Yongcheng Wu and Decheng Ding. Computability of measurable sets via effective metrics. *Mathematical Logic Quarterly*, 51(6):543–559, 2005.
9. Yongcheng Wu and Decheng Ding. Computability of measurable sets via effective topologies. *Archive for Mathematical Logic*, 45:365–379, 2006.
10. Yongcheng Wu and Klaus Weihrauch. A computable version of the Daniell-Stone theorem on integration and linear functionals. *Theoretical Computer Science*, 359(1-3):28-42, 2006.

Linear-Size Log-Depth Negation-Limited Inverter for k-Tonic Binary Sequences

Hiroki Morizumi[1] and Jun Tarui[2]

[1] Graduate School of Informatics, Kyoto University
Kyoto 606-8501 Japan
morizumi@kuis.kyoto-u.ac.jp
[2] Department of Info and Comm Eng, University of Electro-Comm
Chofu, Tokyo 182-8585 Japan
tarui@ice.uec.ac.jp

Abstract. A zero-one sequence $x_1, \ldots, x_n$ is *k-tonic* if the number of i's such that $x_i \neq x_{i+1}$ is at most k. The notion generalizes well-known *bitonic* sequences. In *negation-limited* complexity, one considers circuits with a limited number of NOT gates, being motivated by the gap in our understanding of monotone versus general circuit complexity, and hoping to better understand the power of NOT gates. In this context, the study of *inverters*, i.e., circuits with inputs $x_1, \ldots, x_n$ and outputs $\neg x_1, \ldots, \neg x_n$, is fundamental since an inverter with r NOTs can be used to convert a general circuit to one with only r NOTs. In particular, if linear-size log-depth inverter with r NOTs exists, we do not lose generality by only considering circuits with at most r NOTs when we seek superlinear size lower bounds or superlogarithmic depth lower bounds. Markov [JACM1958] showed that the minimum number of NOT gates necessary in an n-inverter is $\lceil \log_2(n+1) \rceil$. Beals, Nishino, and Tanaka [SICOMP98–STOC95] gave a construction of an n-inverter with size $O(n \log n)$, depth $O(\log n)$, and $\lceil \log_2(n+1) \rceil$ NOTs. We give a construction of circuits inverting k-tonic sequences with size $O((\log k)\, n)$ and depth $O(\log k \, \log\log n + \log n)$ using $\log_2 n + \log_2 \log_2 \log_2 n + O(1)$ NOTs. In particular, for the case where $k = O(1)$, our k-tonic inverter achieves asymptotically optimal linear size and logarithmic depth. Our construction improves all the parameters of the k-tonic inverter by Sato, Amano, and Maruoka [COCOON06] with size $O(kn)$, depth $O(k \log^2 n)$, and $O(k \log n)$ NOTs. We also give a construction of k-tonic *sorters* achieving linear size and logarithmic depth with $\log_2 \log_2 n + \log_2 \log_2 \log_2 n + O(1)$ NOT gates for the case where $k = O(1)$. The following question by Turán remains open: Is the size of any depth-$O(\log n)$ inverter with $O(\log n)$ NOT gates superlinear?

Keywords: circuit complexity, negation-limited circuit, inverter, k-tonic.

1 Introduction and Summary

Although exponential lower bounds are known for the monotone circuit size [12], [8], [5], at present we cannot prove a superlinear lower bound for the size of

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 605–615, 2007.
© Springer-Verlag Berlin Heidelberg 2007

circuits computing an explicit Boolean function. It is natural to ask: What happens if we allow a limited number of NOT gates? The hope is that by the study of *negation-limited* complexity of Boolean functions under various scenarios [6], [7], [4], [3], [2], [13], [9], we obtain a better understanding about the power of NOT gates.

As explained in the abstract, the study of inverters is fundamental in this context. We consider circuits consisting of AND/OR/NOT gates, and the *size* of a circuit is the number of gates in it. The best known construction of a general inverter is due to Beals, Nishino, and Tanaka [4]. Their inverter has size $O(n \log n)$ and depth $O(\log n)$ and uses $\lceil \log_2(n+1) \rceil$ NOT gates. . In a recent paper [13], Sato, Amano, and Maruoka considered circuits that is guaranteed to invert a restricted class of inputs, and gave a construction for a *k-tonic inverter*, i.e., a circuit that inverts all k-tonic 0/1 sequences, with size $O(kn)$, depth $O(k \log^2 n)$, and $O(k \log n)$ NOTs. We give a new, different construction of a k-tonic inverter achieving improvements of all the three parameters. In particular, for $k = O(1)$, we achieve asymptotically optimal linear size and logarithmic depth using only slightly more than $\log_2 n$ NOT gates. Improvements are shown in Table 1.

Theorem 1. *There is a k-tonic inverter that has size $O((\log k)n)$ and depth $O(\log k \log\log n + \log n)$, and uses $\log_2 n + \log_2 \log_2 \log_2 n + O(1)$ NOT gates.*

Table 1. The parameters of the k-tonic inverters of Sato et al [13] and this paper

	Sato et al [13]	this paper
size	$O(kn)$	$O((\log k)n)$
depth	$O(k \log^2 n)$	$O(\log k \log\log n + \log n)$
# of NOTs	$O(k \log n)$	$\log_2 n + \log_2 \log_2 \log_2 n + O(1)$

Amano, Maruoka, and Tarui [3] considered the minimum size of a circuit that merges two 0/1 sequences using t NOT gates, and they showed that it is $\Theta(n \log n/2^t)$, thus demonstrating a smooth trade-off of size versus the number of NOTs from the monotone case of $\Theta(n \log n)$ to the general case of $\Theta(n)$. Their merging circuit actually works for any bitonic sequence. Sato, Amano, Maruoka [13] also considered a generalized scenario in terms of k-tonic sequences and, for $t \leq \log_2 n$ and $k = O(\log n)$, they gave a construction of a *k-tonic sorter*, i.e., a circuit that sorts all k-tonic binary sequences, that has size $O(kn + (n \log n)/2^t)$ and uses $O(tk^2)$ NOT gates. The design principle and the analysis of our k-tonic inverter immediately yields an improved k-tonic sorter:

Theorem 2. *There is a k-tonic sorter that uses t NOT gates and has size $O\left((\log k)n + (n \log n)(t/2^t)\right)$ and depth $O((\log k) \cdot t + \log n)$.*

2 Component Circuits/Networks

In this section we explain the components that we use in our circuits. The constructions in Sections 2.1 and 2.2 are due to Beals, Nishino, and Tanaka [4].

The reader may choose to skip this section and come back to it after seeing how components are assembled and used in our circuits.

2.1 Inverting the Inputs of a Comparator Network

Let N_1 be a *comparator network* (see, e.g., Knuth [10]) with inputs $v_1, \ldots, v_n$ and outputs $w_1, \ldots, w_n$. Assume that N_1 has depth d and contains s comparators. Consider the case where inputs are Boolean. In the Boolean case, each comparator can be considered as a pair of one AND gate and one OR gate (Figure 1), and thus N_1 can be considered as a depth-d size-$2s$ monotone circuit.

Assume that the negations of the *outputs* of N_1, i.e., $\neg w_1, \ldots, \neg w_n$ are computed by another circuit and are available. Then, we can construct a circuit N_2 that outputs the negations of the *inputs* $\neg v_1, \ldots, \neg v_n$ as follows. For each comparator c with inputs x_1 and x_2 and outputs y_1 and y_2, we can compute $\neg x_1$ and $\neg x_2$ from $x_1, x_2, \neg y_1, \neg y_2$ as shown in Figure 1. Repeatedly apply this construction considering comparators one by one from the outputs of N_1 towards the inputs, and obtain the network N_2. The circuit N_2 has depth $2d$ and consists of $2s$ ANDs and $2s$ ORs.

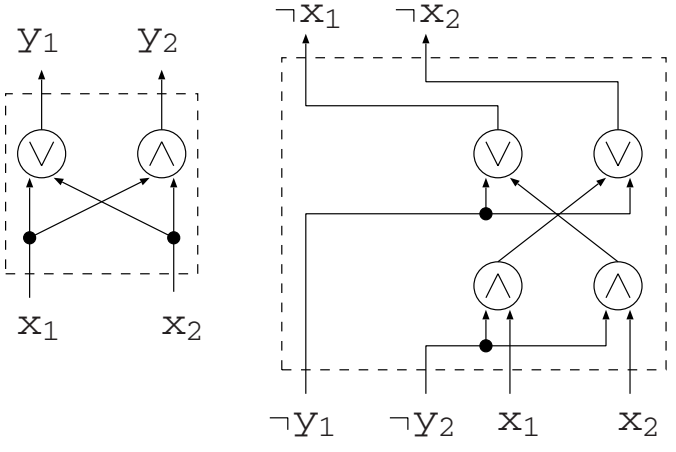


Fig. 1. computing the negations of inputs using the negations of outputs

2.2 The Beals-Nishino-Tanaka Inverter

The inverter operates as follows. Sort $x_1, \ldots, x_n$ by the (upside-down) AKS n-sorting network [1] with depth $O(\log n)$ and size $O(n)$, and obtain $y_1 \geq \cdots \geq y_n$. Apply Fischer's network M_n [6], [7], [4], and obtain $\neg y_1, \ldots, \neg y_n$. Finally, apply the network explained in Section 2.1 that outputs $\neg x_1, \ldots, \neg x_n$ using $\neg y_1, \ldots, \neg y_n$. Here M_n is a network that inverts a sorted 0/1-sequence $y_1 \geq \cdots \geq y_n$ with size $O(n)$ and depth $O(\log n)$ using $\lceil \log_2(n+1) \rceil$ NOT gates. (More precisely, for $n = 2^r - 1$, M_n has size $4n - 3r$; this is the minimum

size [9] of circuits inverting n sorted inputs with r NOTS.) The inverter uses $\lceil \log_2(n+1) \rceil$ NOT gates and has depth $O(\log n)$ and size $O(n \log n)$.

2.3 Conditional Shifter

Let $p \in \{0,1\}$. Assume that $\delta \leq \alpha$ and let $y_1, \ldots, y_{\alpha+\delta}$ be a 0/1-sequence. Suppose that we want to let $z_1, \ldots, z_\alpha$ respectively be $y_1, \ldots, y_\alpha$ if $p = 1$ and $y_{\delta+1}, \ldots, y_{\alpha+\delta}$ if $p = 0$. In other words, we want to either (1) discard the last δ y_j's, or (2) discard the first δ y_j's and then shift by δ. This can be easily be done using p and $\neg p$ as follows: For $j = 1, \ldots, \alpha$, compute

$$z_j = (p \wedge y_j) \vee (\neg p \wedge y_{j+\delta}).$$

Let $y'_1 \leq \cdots \leq y'_{\alpha+\delta}$ and $z'_1 \leq \cdots \leq z'_\alpha$ respectively be the 0/1-sequences obtained by sorting $y_1, \ldots, y_{\alpha+\delta}$ and $z_1, \ldots, z_\alpha$. Assume further that the following condition holds:

$$p = 0 \Longrightarrow y_1 = \cdots = y_\delta = 0;$$
$$p = 1 \Longrightarrow y_{\alpha+1} = \cdots = y_{\alpha+\delta} = 1.$$

Then, we can easily compute $\neg y_1, \ldots, \neg y_{\alpha+\delta}$ and $y'_1, \ldots, y'_{\alpha+\delta}$ as follows.

$$\neg y_j = \begin{cases} (p \wedge \neg z_j) \vee \neg p & \text{for } j = 1, \ldots, \delta; \\ (p \wedge \neg z_j) \vee (\neg p \wedge \neg z_{j-\delta}) & \text{for } j = \delta+1, \ldots, \alpha; \\ \neg p \wedge \neg z_{j-\delta} & \text{for } j = \alpha+1, \ldots, \alpha+\delta. \end{cases}$$

$$y'_j = \begin{cases} p \wedge z'_i & \text{for } j = 1, \ldots, \delta; \\ (p \wedge z'_j) \vee (\neg p \wedge z'_{j-\delta}) & \text{for } j = \delta+1, \ldots, \alpha; \\ p \vee (\neg p \wedge z'_{j-\delta}) & \text{for } j = \alpha+1, \ldots, \alpha+\delta. \end{cases}$$

3 Negation-Limited k-Tonic Inverter

Most of this section is devoted to an explanation of our k-tonic inverter claimed in Theorem 1. In Section 3.1 we explain the overall structure of our k-tonic inverter. For the sake of exposition, we first consider computing *pivot bits* in a naive way, and we provide rough analysis for the number of NOT gates needed for reducing the problem size. In Section 3.2 we explain how we actually compute pivot bits in our circuit to achieve the claimed depth. It turns out that most work we do is giving appropriate *definitions* and developing an appropriate *framework* for analysis. In Section 3.3 we explain how we can achieve the number of NOT gates as claimed in Theorem 1 by a simple finer analysis. In Section 3.4 we explain how we can obtain our k-tonic sorter in a similar way.

3.1 Overall Structure of the k-Tonic Inverter

We first explain using a general algorithmic language and then explain in terms of circuits. We consider a k-tonic binary input sequence of length n and assume

that $n = k2^r$ for some integer r. If n is not of this form, we can pad an input sequence $x = \langle x_1, \ldots, x_n \rangle$ with trailing 1's and obtain the sequence $x' = \langle x_1, \ldots, x_n, 1, \ldots, 1 \rangle$ whose length is the minimum $n' \geq n$ of the form $n' = k2^r$, apply the inverter for the $(k+1)$-tonic sequence x', and discard the last $n' - n$ outputs.

Let $x = \langle x_1, \ldots, x_n \rangle$ be a k-tonic 0/1 sequence of length $n = 4km$. Think of x_i's as entries of a $4k \times m$ matrix M as follows. (We won't be doing any linear algebra; we can equally speak in terms of a rectangular array or a two-dimensional grid.)

$$M = \begin{pmatrix} x_1 & x_2 & \cdots & x_{m-1} & x_m \\ x_{m+1} & x_{m+2} & \cdots & x_{2m-1} & x_{2m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(4k-1)m+1} & x_{(4k-1)m+2} & \cdots & x_{4km-1} & x_{4km} \end{pmatrix}.$$

A row is *dirty* if it contains both 0 and 1; otherwise it is *clean*; an all-0 row is *0-clean* and an all-1 row is *1-clean*.

Since the sequence x is k-tonic, among the $4k$ rows of M, at most k rows are dirty. Sort each *column* of M with smaller entries up, and obtain the matrix $\overline{M_0}$. The matrix $\overline{M_0}$ has at most k dirty rows, and all of them as middle rows. Thus either the bottom $(4k-k)/2 = 3k/2$ rows are all 1-clean, or else the top $3k/2$ rows are all 0-clean. For now, we use the following weaker form: Either (1) all the bottom k rows are 1-clean or (2) all the top k rows are 0-clean.

Define *pivot bit* p as $p =$ AND of the km entries in the bottom k rows of $\overline{M_0}$. Use one NOT gate and obtain $\neg p$. Using p and $\neg p$ discard either the bottom k 1-clean rows or the top k 0-clean rows according to whether $p = 1$ or 0, i.e., whether (1) holds or not. The remaining $3k$-row matrix $\overline{M_1}$ has at most k dirty rows, and all of them as middle rows. Again discard the bottom $(3k-k)/2 = k$ rows or the top k rows using the pivot bit for the bottom k rows together with one NOT gate.

We are left with a $2k \times m$ matrix $\overline{M_2}$. Split each row of $\overline{M_2}$ into the first half and the last half. Let L be the $4k \times \frac{m}{2}$ matrix whose $4k$ rows are the $4k$ halves of the rows of $\overline{M_2}$ *stacked* on top of one another for each row. At most k rows of L are dirty. Thus using two NOT gates we have halved the problem size: We can apply the same operation and arguments for L, i.e., sort each column and discard $2k$ clean rows using two NOT gates.

We now explain in terms of circuits. We start over with the $4k \times m$ matrix M above. To sort each column of M, apply AKS sorting networks each sorting $4k$ elements; use m separate networks for m columns in parallel. Now consider the column-sorted matrix $\overline{M_0}$, and let $y_1, \ldots, y_n$ be the entries in its first row through its last row.

Consider, for now, computing the pivot bit p naively as $p = \wedge_{j=3km+1}^{4km} y_j$. Using one NOT gate compute $\neg p$. Discard the top k rows or the bottom k rows by the conditional shifter in Section 2.3: For $j = 1, \ldots, 3km$, compute $z_j = (p \wedge y_j) \vee (\neg p \wedge y_{j+k})$. The z_j's are the entries of the $3k \times m$ matrix $\overline{M_1}$. Assume

that $\neg z_1, \ldots, \neg z_{3km}$ are the outputs of the our subcircuit inverting $z_1, \ldots, z_{3km}$. We can use the conditional shifter for negations in Section 2.3 and obtain $\neg y_j$'s.

Continue halving the problem size $\nu = \log_2 \log_2 n$ times so that the size is $n' = n/\log_2 n$, and then use the Beals-Nishino-Tanaka inverter for n' inputs.

Our circuit consists of three parts (Figure 2):

subcircuit 1 computing pivot bits and reducing the problem size from n to n'

subcircuit 2: the Beals-Nishino-Tanaka inverter for $n' = n/\log_2 n$ inputs

subcircuit 3, which shifts the outputs appropriately using the the pivot bits

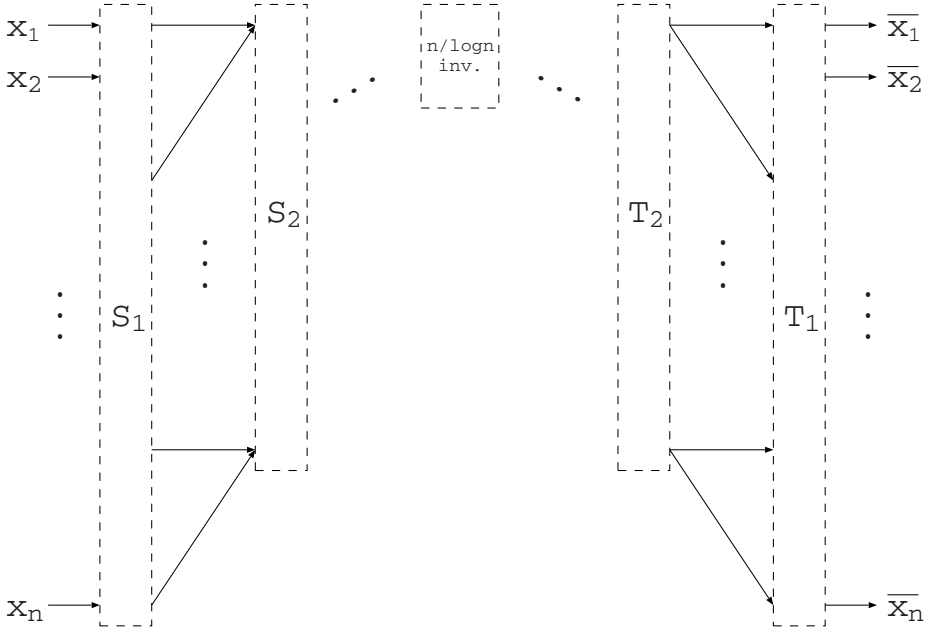


Fig. 2. Overall structure of the inverter

The inverter explained so far has the parameters shown in Table 2. The parameters of subcircuits 2 and 3 are readily seen. We provide some explanation for the parameters of subcircuit 1.

For $n = km$, consider the first parallel application of m separate AKS sorting networks each sorting k elements. This first part has size $O((\log k)n)$. Since the value of n geometrically decreases by a constant factor, the size of subcircuit 1 is dominated by the size of this first part. If we compute each pivot bit naively as taking the AND of some y_j's as above, this takes depth $O(\log n)$; we repeat this ν times; thus the depth of subcircuit 1 will be $O(\log n \cdot \nu) = O(\log n \log\log n)$. In the consideration above we halve the problem size by two NOT gates; thus a total of $2\nu = 2\log_2 \log_2 n$ NOTs are used for the problem size reduction.

Table 2. The parameters of subcircuits 1, 2, 3

	subcircuit 1	subcircuit 2	subcircuit 3
size	$O((\log k)n)$	$O(n)$	$O(n)$
depth	$O(\log n \log\log n)$	$O(\log n)$	$O(\log n)$
# of NOTs	$2\log_2\log_2 n + O(1)$	$\log_2 n - \log_2\log_2 n + O(1)$	0

In Section 3.2 and 3.3 we explain how to reduce the depth of subcircuit 1 to $O(\log k \log\log n)$ and the number of NOTs in subcircuit 1 to $\log_2\log_2 n + \log_2\log_2\log_2 n + O(1)$, and thus obtain an k-tonic inverter with the parameters claimed in Theorem 1.

3.2 Reducing the Depth to $O(\log k \log\log n)$

Let M be an $l \times m$ 0/1-matrix. Let t be a nonnegative integer such that 2^t divides m, i.e., we can divide each row into 2^t consecutive parts of equal size. The parameter t represents the number of pivot bits in our circuit, which equals the number of times that we shrink the problem size by a constant factor, which also equals the number of NOT gates that we use.

For $s = 0, 1, \ldots, t$, we define an *s-block* of M as follows. Each row itself forms a 0-block; there are l 0-blocks. Split each 0-block, i.e., split each row into the first half and the last half. These halves are the $2l$ 1-blocks. Similarly, splitting each $(s-1)$-block yields two s-blocks; there are $2^s l$ s-blocks. Thus each row forms 2^s s-blocks for $s = 0, \ldots, t$, and hence forms a total of $u = \sum_{s=0}^{t} 2^s = 2^{t+1} - 1$ blocks. For each row, order these u blocks as follows. The first block is the 0-block, i.e., the whole row. Then comes the two 1-blocks, i.e., the first half and the last half, in this order. Then comes the four 2-blocks, i.e, the first quarter up to the last quarter; and so on.

Let $F = (f_{ij})$ be an $l \times u$ 0/1-matrix, where u is as above. Our intention will roughly be to let the equality $f_{ij} = 1$ represent the fact that the j-th block in the i-th row of M is 1-clean, i.e., all-1.

In our circuit, we call f_{ij} a *flag bit*. We compute flag bits f_{ij}'s *just once* as follows. For each t-block b, which is a smallest block, compute the flag bit f_b for block b as $f_b = \wedge_{x_i \in b}\, x_i$. For $s = t-1, \ldots, 0$, each s-block b contains two $(s+1)$-blocks b_1 and b_2; compute f_b as $f_b = f_{b_1} \wedge f_{b_2}$. Thus all flag bits are initially computed using only ANDs in depth $\log_2 m$. After initial computation, we *sort* flag bits column-wise using AKS sorting networks and discard bottom or top rows of flag bits as we discard top or bottom rows of input bits.

Right after the initial computation, the flag bit f_b for a block b is 1 iff the block b is 1-clean. After sorting f_b's, this may not hold: it is possible that a block b is 1-clean but $f_b = 0$. But sorting maintains the property that if $f_b = 1$, then b is 1-clean (we later call this 1-conservative), and we show how this suffices for our purposes. We discard the bottom k rows of input bits if the AND of the corresponding k flag bits is 1, computing the AND in depth $\lceil \log_2 k \rceil$. In other

words, the first pivot is computed in depth $\lceil \log_2 n \rceil$, and thereafter each pivot bit is computed in depth $\lceil \log_2 k \rceil$. This is how we obtain the claimed depth.

We proceed to show the correctness of the method above. We give definitions of key properties; Lemma 2 says that the properties hold after the initial computation of flag bits; Lemma 3 says that the properties are maintained by the operations above.

For two matrices M and F as above, the pair (M, F) is *1-conservative* if the following holds: For $1 \leq i \leq l$ and $1 \leq j \leq u$, if $f_{ij} = 1$ then the j-th block in the i-th row of M is 1-clean. The j-th block b in the i-th row of M is *good* if either (1) b is 1-clean *and* $f_{ij} = 1$ or (2) b is 0-clean; otherwise b is a *bad* block. Say that (M, F) is *k-mixed* if there are at most k bad s-blocks for each $s = 0, \ldots, t$. Note that the definitions of 1-conservative and k-mixed are with respect to the parameter t. When appropriate, we make this dependence explicit by saying, e.g., *k-mixed with respect to t subdivisions*.

Let (M, F) be as above: M is an $l \times m$ matrix and F is an $l \times u$ matrix, where $u = \sum_{s=0}^{t} 2^s = 2^{t+1} - 1$ for a parameter $t \leq 2^m$. *Stacking* (M, F) yields the pair of matrices $(\widehat{M}, \widehat{F})$, where $\widehat{M}$ is an $2l \times (m/2)$ matrix and $\widehat{F}$ is an $2l \times ((u-1)/2)$ matrix obtained as follows. Split each row of M into the first half and the last half. Stack the $2l$ halves thus obtained on top of one another, and obtain a $2l \times m/2$ matrix $\widehat{M}$. In other words, the first half and the last half of the i-th row of M is respectively the $(2i-1)$-th row and the $2i$-th row of $\widehat{M}$. As for $\widehat{F}$: Throw away the first column of F, which corresponds to 0-blocks of M; the 0-blocks have been thrown away by stacking M into $\widehat{M}$. Put the second column of F on top to the third column; put the 4th and the 5th column on top of the 6th and the 7th column respectively; in general put the $(2^s + r)$-th column on top of the $(2^s + 2^{s-1} + r)$-th column $(0 \leq r < 2^s, 1 \leq s \leq t)$, and obtain $\widehat{F}$.

Lemma 2. Let $x_1, \ldots, x_n$ be a k-tonic 0/1-sequence of length $n = lm$. Let M be the $l \times m$ matrix having x_i's in a row-major form: e.g., the first row is $x_1, \ldots, x_m$. Consider s-blocks of M for $0 \leq s \leq t$. Let $F = (f_{ij})$ be the $l \times m$ 0/1-matrix such that $f_{ij} =$ AND of all x_r's in the j-th block of the i-th row. Then, (M, F) is 1-conservative and k-mixed with respect to t subdivisions.

Proof. By definition of f_{ij}'s, the pair (M, F) is 1-conservative; furthermore, clearly there is no 1-clean block with the corresponding flag bit f_{ij} being 0: In the setting above a block b is bad iff it is dirty. A 0-1 change in the sequence $x_1, \ldots, x_n$ produces at most one dirty s-block for each $s = 0, \ldots, t$. The lemma follows. □

Lemma 3. Assume that (M, F) is 1-conservative and is k-mixed with respect to t subdivisions. Let $\overline{M}$ and $\overline{F}$ respectively denote the matrix obtained by sorting columns of M and F. Further, let $\widehat{\overline{M}}$ and $\widehat{\overline{F}}$ respectively denote the matrix obtained by stacking $\overline{M}$ and $\overline{F}$. Then, $(\overline{M}, \overline{F})$ is 1-conservative and k-mixed with respect to t subdivisions. Further, $(\widehat{\overline{M}}, \widehat{\overline{F}})$ is 1-conservative and k-mixed with respect to $t - 1$ subdivisions.

Proof. For simplicity, first consider 0-blocks, i.e, rows of M and the corresponding first column $c = (c_{i1})$ of F. Assume that c contains α 1's, and that $c_{i_1 1} = c_{i_2 1} = \cdots = c_{i_\alpha} 1 = 1$.

Since (M, F) is 1-conservative, the α rows of M, rows $i_1, i_2, \ldots, i_\alpha$, are 1-clean. After column-sorting, the first column $\overline{c_{i1}}$ of $\overline{F}$ contains α 1's at the bottom, and the bottom α rows of $\overline{M}$ are 1-clean. Thus the condition of being 1-conservative holds with respect to the first column of F and the corresponding blocks, i.e., all the 0-blocks. Exactly the same argument applies for any column and the corresponding blocks. Hence (M, F) is 1-conservative.

Assume that M and F have l rows and that M is k-mixed. To see that $(\overline{M}, \overline{F})$ is k-mixed, again first consider 0-blocks, i.e., rows of M and the corresponding first column $c = (c_{i1})$ of F. Assume that M has α 1-clean good rows and β 0-clean good rows. By definition of goodness, the column c contains α 1's. After column-sorting, the bottom α rows are 1-clean and the bottom α entries of $\overline{c}$ are 1's; thus there are α 1-clean good blocks. The top β rows are 0-clean, and hence they are good. Thus there are as many good 0-blocks in $(\overline{M}, \overline{F})$ as in (M, F).

Now consider the 1-blocks of M, i.e., the first halves and the last halves of the rows. Assume that there are k_1 bad 1-blocks among the first halves and k_2 bad 1-blocks among the last halves, where $k_1 + k_2 \leq k$. We can apply the above argument for 0-blocks separately for the first halves and for the last halves. In each of the two cases, after column-sorting there are as many good 1-blocks as before. Hence the assertion holds for 1-blocks. For the s-blocks, we can argue similarly separately considering 2^s groups of s-blocks. This completes our proof that $(\overline{M}, \overline{F})$ is k-mixed.

Finally, stacking does not destroy being 1-conservative nor does it introduce any new bad block. □

In our circuit stacking simply corresponds to rearranging the ordering of the intermediate gates; it does not need any gate. This completes our proof of the claimed depth reduction.

3.3 Reducing the Number of NOTs

Consider, as in Section 3,1, a $4k$-row matrix M consisting of a k-tonic 0/1-sequence, and the matrix $\overline{M_0}$ obtained from M by sorting each column. Discard k rows using one NOT gate. Now, instead of again discarding as explained in Section 3.1, consider processing the remaining $6k$ 1-blocks; i.e., halve the $3k$ rows, stack the $6k$ halves, and consider the resulting $6k$-row matrix. At most k rows are bad. Discard $2k$ rows. (Actually we can discard $(6k - k)/2 = (5/2)k$ rows; this does not yield an asymptotic improvement.)

Further halve and stack to obtain $2(6k-2k) = 8k$ rows, discard $3k$ rows, obtain $2(8k - 3k) = 10k$ rows, discard $4k$ rows, and so on. In general, at iteration s, discard sk rows out of $(2s+2)k$ rows; halve and stack to obtain $((2s + 2)k - sk) \times 2 = 2sk + 4k = (2(s + 1) + 2)\, k$ rows. Thus with t NOT gates we reduce the problem as follows. We assume that $t \geq 2$.

$$\frac{3}{4}\cdot\frac{4}{6}\cdot\frac{5}{8}\cdot\frac{6}{10}\cdot\cdots\cdot\frac{t+2}{2t+2}=\prod_{s=1}^{t}\frac{s+2}{2s+2}=(1/2)^{t-1}\,\frac{t+2}{4}\le(1/2)^t\cdot t,$$

where we have the second equality since each denominator is twice the previous numerator. Thus we can reduce the size to $1/\log_2 n$ using t NOT gates with t satisfying $2^t/t \ge \log_2 n$, and hence with $t = \log_2\log_2 n + \log_2\log_2\log_2 n + O(1)$.

In the scheme above, the column size in column-sorting increases, and we use AKS k'-sorting networks for increasing k'. This increases the depth of subnetwork 1, but we can easily see that the asymptotic depth does not change. This completes the description of our k-tonic inverter as claimed in Theorem 1, and thus the proof of Theorem 1.

3.4 Negation-Limited k-Tonic Sorter

We can obtain a k-tonic sorter in Theorem 2 as follows. Use exactly the same design as above to reduce the problem size to $n' = n\ (t/2^t)$ with t NOT gates. Then, instead of the Beals-Nishino-Tanaka inverter use the AKS n'-sorting network. Apply the shifter for sorted outputs described in Section 2.3.

4 Open Problems

The following question by Turán remains open: Is the size of any depth-$O(\log n)$ inverter with $O(\log n)$ NOT gates superlinear?

For $k = O(1)$, can we reduce the number of NOT gates in a k-tonic inverter to $\log_2 n + O(1)$ while maintaining size $O(n)$ and depth $O(\log n)$? This question may be of interest somewhat beyond its minor technical appearance: in one way it asks whether we can utilize *each* NOT gate to *exactly halve* the problem size.

Acknowledgement

We thank the anonymous referees for the valuable comments and suggestions.

References

1. M. Ajtai, J. Komlós and E. Szemerédi, An $O(n \log n)$ Sorting Network, *Proceedings of STOC83: the 15th Annual ACM Symposium on Theory of Computing* (1983), pp. 1–9.
2. K. Amano and A. Maruoka, A Superpolynomial Lower Bound for a Circuit Computing the Clique Function with at most $(1/6)\log\log n$ Negation Gates, *SIAM Journal on Computing* (2005) **35(1)**, pp. 201–216.
3. K. Amano, A. Maruoka and J. Tarui, On the Negation-Limited Circuit Complexity of Merging, *Discrete Applied Mathematics* (2003) **126(1)**, pp. 3–8.

4. R. Beals, T. Nishino and K. Tanaka, On the Complexity of Negation-Limited Boolean Networks. *SIAM Journal on Computing*, (1998) **27(5)**, pp. 1334–1347; a preliminary version appears in: *Proceedings of STOC95: the 27th Annual ACM Symposium on Theory of Computing* (1995), pp. 585–595.
5. R. Boppana and M. Sipser, The Complexity of Finite Functions, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, J. v. Leeuwen editor, Elsevier/MIT Press (1990), pp. 757–804.
6. M. Fischer, The Complexity of Negation-Limited Networks - A Brief Survey, *Lecture Notes in Computer Science* (1975) vol. **33**, pp. 71–82.
7. M. Fischer, Lectures on Network Complexity. *Technical Report* 1104, (1974, revised 1996) CS Department, Yale University, (available on the web).
8. D. Harnik and R. Raz, Higher Lower Bounds on Monotone Size, *Proceedings of STOC00: the 32nd Annual ACM Symposium on Theory of Computing* (2000), pp. 378–387.
9. K. Iwama, H. Morizumi and J. Tarui, Negation-Limited Complexity of Parity and Inverters, *Proceedings of ISAAC06: the 17th International Symposium on Algorithms and Computation* (2006), Lecture Notes in Computer Science vol. 4280, pp. 223–232.
10. D. Knuth, *The Art of Computer Programming vol. 3: Sorting and Searching*, 2nd edition (1998), Addison-Wesley.
11. A. Markov, On the Inversion Complexity of a System of Functions, *Journal of the ACM* (1958) **5(4)**, pp. 331–334.
12. A. Razborov, Lower Bounds on the Monotone Complexity of Some Boolean Functions, *Dokl. Akad. Nauk SSSR* (1985) **281(4)**, pp. 798–801 (in Russian); English translation in: *Soviet Math. Dokl.* (1985) **31**, pp. 354–357.
13. T. Sato, K. Amano, and A. Maruoka, On the Negation-Limited Circuit Complexity of Sorting and Inverting k-tonic Sequences, *Proceedings of COCOON06: the 12th Annual International Computing and Combinatorics Conference* (2006), Lecture Notes in Computer Science vol. 4112, pp. 104–115.

The Existence of Unsatisfiable Formulas in k-LCNF for $k \geq 3$*

Qingshun Zhang and Daoyun Xu

Department of Computer Science,
Guizhou University (550025), Guiyang, P.R. China
zhangqingshunnb@126.com, dyxu@gzu.edu.cn

Abstract. A CNF formula F is linear if any distinct clauses in F contain at most one common variable. A CNF formula F is exact linear if any distinct clauses in F contain exactly one common variable. All exact linear formulas are satisfiable [4], and for the class LCNF of linear formulas , the decision problem LSAT remains NP-complete. For the subclasses LCNF of $\text{LCNF}_{\geq \text{k}}$, in which formulas have only clauses of length at least k, the decision problem $\text{LSAT}_{\geq \text{k}}$ remains NP-complete if there exists an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$ [3,5]. Therefore, the NP-completeness of SAT for $\text{LCNF}_{\geq \text{k}}$ ($\text{k} \geq 3$) is the question whether there exists an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$. In [3,5], it is shown that both $\text{LCNF}_{\geq 3}$ and $\text{LCNF}_{\geq 4}$ contain unsatisfiable formulas by the constructions of hypergraphs and latin squares. It leaves the open question whether for each $k \geq 5$ there is an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$. In this paper, we present a simple and general method to construct minimal unsatisfiable formulas in k-LCNF for each $k \geq 3$.

1 Introduction

A literal is a propositional variable or a negated propositional variable. A clause C is a disjunction of literals, $C = (L_1 \vee \cdots \vee L_m)$, or a set $\{L_1, \cdots, L_m\}$ of literals. A formula F in conjunctive normal form (CNF) is a conjunction of clauses, $F = (C_1 \wedge \cdots \wedge C_n)$, or a set $\{C_1, \cdots, C_n\}$ of clauses, or a list $[C_1, \cdots, C_n]$ of clauses. $var(F)$ is the set of variables occurring in the formula F and $var(C)$ is the set of the variables in the clause C. We denote $\#cl(F)$ as the number of clauses of F and $\#var(F)$ (or $|var(F)|$) as the number of variables occurring in F. CNF(n, m) is the class of CNF formulas with n variables and m clauses. The *deficiency* of a formula F is defined as $\#cl(F) - \#var(F)$, denoted by $d(F)$. A formula F is minimal unsatisfiable (MU) if F is unsatisfiable and $F - \{C\}$ is satisfiable for any clause $C \in F$. It is well known that F is not minimal unsatisfiable if $d(F) \leq 0$ [1]. So, we denote MU(k) as the set of minimal unsatisfiable formulas with deficiency $k \geq 1$. Whether or not a formula belongs to MU(k) can be decided in polynomial time [2].

* The work is supported by the National Natural Science Foundation of China (No. 60463001, No. 10410638, No. 60310213).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 616–623, 2007.
© Springer-Verlag Berlin Heidelberg 2007

A CNF formula F is linear if any two distinct clauses in F contain at most one common variable. A CNF formula F is exact linear if any two distinct clauses in F contain exactly one common variable. We define k-CNF := {F ∈ CNF| (∀C ∈ F)(|C| = k)}, LCNF := {F ∈ CNF| F is linear}, $\text{LCNF}_{\geq \text{k}}$:= {F ∈ LCNF| (∀C ∈ F)(|C| ≥ k)}, and k-LCNF := {F ∈ LCNF| (∀C ∈ F)(|C| = k)}. The decision problems of satisfiability are denoted as k-SAT, LSAT, and k-LSAT for restricted instances to the corresponding to subclasses, respectively.

It is shown that every exact linear formulas is satisfiable [4], but LSAT remains NP-completeness [3,4,5]. For the subclasses $\text{LCNF}_{\geq \text{k}}$, $\text{LSAT}_{\geq \text{k}}$ remains NP-completeness if there exists an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$ [3,4,5]. Therefore, the NP-completeness of $\text{LSAT}_{\geq \text{k}}$ for $k \geq 3$ is the question whether there exists an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$. We are interested in some NP-complete problems for linear formulas, and get some simplified NP-complete problem by constructing unsatisfiable linear formulas. It is helpful to analyze complexity of resolutions, and to find some effective algorithm for satisfiability.

In [3,5], by the constructions of hypergraphs and latin squares, the unsatisfiable formulas in $\text{LCNF}_{\geq \text{k}}$ ($\text{k} = 3, 4$) are constructed, respectively. But, the method is too complex and has no generalization. In [5], it leaves the open question whether for each $k \geq 5$ there is an unsatisfiable formula in $\text{LCNF}_{\geq \text{k}}$.

In this paper, based on the characterization of minimal unsatisfiable formulas, we introduce a generalize method in Lemma 1 and Lemma 2, which we can transform a CNF formula into a required CNF formula by constructing proper minimal unsatisfiable formulas. By this method, we present a simple and general method to construct unsatisfiable formulas in k-LCNF for each $k \geq 3$ by the application of minimal unsatisfiable formulas and the induction. It is shown for each $k \geq 3$ that there exist minimal unsatisfiable formulas in k-LCNF.

2 Minimal Unsatisfiable Formulas and Its Applications

A clause $C = (L_1 \vee L_2 \vee \cdots \vee L_n)$ can be presented as a set $\{L_1, L_2, \cdots, L_n\}$ of literals. Similarly, A CNF formulas $F = (C_1 \wedge C_2 \wedge \cdots \wedge C_m)$ can be presented as a set $\{C_1, C_2, \cdots, C_m\}$ of clauses, or a list $[C_1, C_2, \cdots, C_m]$ of clauses. $var(F)$ is the set of variables occurring in the formula F and $var(C)$ is the set of the variables in the clause C. We define $|F| = \sum_{1 \leq i \leq m} |C_i|$ as the size of F. In this paper, the formulas mean CNF formulas.

A formula $F = [C_1, \cdots, C_m]$ with n variables $x_1, \cdots, x_n$ in $CNF(n, m)$ can be represented as the following $n \times m$ matrix $(a_{i,j})$, called the representation matrix of F, where $a_{ij} = +$ if $x_i \in C_j$, $a_{ij} = -$ if $\neg x_i \in C_j$, otherwise $a_{ij} = 0$ (or, blank).

A formula F is called *minimal unsatisfiable* if F is unsatisfiable, and for any clause $f \in F$, $F - \{f\}$ is satisfiable. We denote MU as the class of minimal unsatisfiable formulas, and MU(k) as the class of minimal unsatisfiable formulas with deficiency k. Let $C = (L_1 \vee \cdots \vee L_n)$ be a clause. We view a clause

as a set of literals. The collection $C_1, \cdots, C_m$ of subsets of C (as a set) is a partition of C, where $C = \bigcup_{1 \le i \le m} C$ and $C_i \cap C_j = \phi$ for any $1 \le i \ne j \le m$, which corresponds to a formula $F_C = C_1 \wedge \cdots \wedge C_m$. We call F_C as a partition formula of C. Specially, the collection $\{L_1\}, \cdots \{L_n\}$ of singleton subsets of C is called the simple partition of C, and the formula $[L_1, \cdots, L_n] = L_1 \wedge \cdots \wedge L_n$ is called the *simple partition formula* of C.

Let $F_1 = [f_1, \cdots, f_m]$ and $F_2 = [g_1, \cdots, g_m]$ be formulas. We denote $F_1 \vee_{cl} F_2 = [f_1 \vee g_1, \cdots, f_m \vee g_m]$. Similarly, let C be a clause and $F = [f_1, \cdots, f_m]$ a formula, denote $C \vee_{cl} F = [(C \vee f_1), \cdots, (C \vee_{cl} f_m)]$.

In the transformation from a CNF formula to a 3-CNF formula, we found a basic application of minimal unsatisfiable: for a clause $C = (L_1 \vee L_2 \vee \cdots \vee L_p)$ $(p > 3)$ one can introduce $(p-3)$ new $y_1, y_2, \cdots, y_{p-3}$ variables, and split C into a partition $\{L_1, L_2\}, \{L_3\}, \cdots, \{L_{p-2}\}, \{L_{p-1}, L_p\}$ of C, and then construct $(p-2)$ clauses $(L_1 \vee L_2 \vee y_1), (L_3 \vee \neg y_1 \vee y_2), \cdots, (L_{p-2} \vee \neg y_{p-4} \vee y_{p-3}), (L_{p-1} \vee L_p \vee \neg y_{p-3})$. In fact, $[y_1, (\neg y_1 \vee y_2), \cdots, (\neg y_{p-4} \vee y_{p-3}), \neg y_{p-3}]$ is a minimal unsatisfiable in MU(1), and the partition $\{L_1, L_2\}, \{L_3\}, \cdots, \{L_{p-3}\}, \{L_{p-1}, L_p\}$ of C corresponds to a CNF formula $[(L_1 \vee L_2), L_3, \cdots, L_{p-2}, (L_{p-1} \vee L_p)]$. Thus, the formula $[(L_1 \vee L_2 \vee y_1), (L_3 \vee \neg y_1 \vee y_2), \cdots, (L_{p-2} \vee \neg y_{p-4} \vee y_{p-3}), (L_{p-1} \vee L_p \vee \neg y_{p-3})]$ is viewed as *clauses-disjunction* of $[(L_1 \vee L_2), L_3, \cdots, L_{p-2}, (L_{p-1} \vee L_p)]$ and $[y_1, (\neg y_1 \vee y_2), \cdots, (\neg y_{p-4} \vee y_{p-3}), \neg y_{p-3}]$ at corresponding positions of clauses, respectively. Additionally, an unit clause L corresponds to the formula $[(L \vee y \vee z), (L \vee y \vee \neg z), (L \vee \neg y \vee z), (L \vee \neg y \vee \neg z)]$, where $[(y \vee z), (y \vee \neg z), (\neg y \vee z), (\neg y \vee \neg z)]$ is a minimal unsatisfiable formula MU(2), and a clause $(L_1 \vee L_2)$ corresponds to the formula $[(L_1 \vee L_2 \vee y), (L_1 \vee L_2 \vee \neg y)]$, where $[y, \neg y] = y \wedge \neg y$ is is a minimal unsatisfiable formula MU(1). It implies that a subclause of the original clause can be copied.

Based on this observation and the characterization of minimal unsatisfiable formulas, we introduce a generalize method as follows.

Lemma 1. *Let $C = (L_1 \vee \cdots \vee L_n)$ $(n \ge 2)$ be a clause and $F_C = [C_1, \cdots, C_m]$ $(m \ge 2)$ a partition formula of C. For any MU formula $H = [f_1, \cdots, f_m]$ with $var(C) \cap var(H) = \phi$, if a truth assignment ν satisfies the formula $F_C \vee_{cl} H$, then $\nu(C) = 1$. Conversely, for any truth assignment ν_0 satisfying C, ν_0 can be extended into a truth assignment ν satisfying $F_C \vee_{cl} H$.*

Proof. Let $C = (L_1 \vee \cdots \vee L_n)$ be a clause and $F_C = [C_1, \cdots, C_m]$ $(m \ge 2)$ a partition formula of C. Without losses of generality (w.l.o.g.), we assume $C_1 = (L_1 \vee \cdots \vee L_{i_1})$, $C_2 = (L_{i_1+1} \vee \cdots \vee L_{i_2})$, $\cdots$, $C_m = (L_{i_{m-1}+1} \vee \cdots \vee L_n)$.

Let ν be a truth assignment satisfying $F_C \vee_{cl} H$. Since H is minimal unsatisfiable, we have $\nu(f_k) = 0$ for some $(1 \le k \le m)$. It must be $\nu(C_k) = 1$. It implies $\nu(C) = 1$ since C_k is a subclause of C.

Conversely, suppose that C is satisfied by a truth assignment ν_0. Since C is disjunction of literals $L_1, \cdots, L_n$, there exists some k $(1 \le k \le n)$ such that $\nu_0(L_k) = 1$. W.l.o.g., we assume $\nu_0(L_1) = 1$, then $\nu_0(C_1) = 1$. Since H is minimal

unsatisfiable, we have $H-\{f_1\}$ is satisfiable, thus there exists a truth assignment ν_1 such that $\nu_1(H-\{f_1\})=1$. Please note that $var(C) \cap var(H) = \phi$, we can join into a truth assignment ν from ν_0 and ν_1, which for $x \in var(C) \cup var(H)$, $\nu(x) = \nu_0(x)$ for $x \in var(C)$, and $\nu(x) = \nu_1(x)$ for $x \in var(H)$. It is clear that ν is a truth assignment satisfying $F_C \vee_{cl} H$.

Example 1. Let $C = (x_1 \vee \cdots \vee x_8)$ be a clause, and $F_C = [(x_1 \vee x_2), (x_3 \vee x_4), (x_5 \vee x_6), (x_7 \vee x_8)]$. We take a minimal unsatisfiable formula $H = [(y_1 \vee y_2), (\neg y_1 \vee y_2), (\neg y_2 \vee y_3), (\neg y_2 \vee \neg y_3)]$, and generate the formula $F_C \vee_{cl} H = [(x_1 \vee x_2 \vee y_1 \vee y_2), (x_3 \vee x_4 \vee \neg y_1 \vee y_2), (x_5 \vee x_6 \vee \neg y_2 \vee y_3), (x_7 \vee x_8 \vee \neg y_2 \vee \neg y_3)]$.

Based on the method in Lemma 1 for a clause, we have the following Lemma 2. It presents a method constructing required formulas.

Lemma 2. *Let $F = C_1 \wedge \cdots \wedge C_n$ be a formula with $|C_i| \geq 2$ for $1 \leq i \leq m$. Suppose that for each $1 \leq i \leq n$, F_i is a partition formula of C_i and $\#cl(F_i) = m_i \geq 2$. Let $H_1, \cdots, H_n$ be* MU *formulas satisfying the following conditions:*

(1) For each $1 \leq i \leq n$, $\#cl(H_i) = m_i$.

(2) $(\bigcup_{1 \leq i \leq n} var(H_i)) \cap var(F) = \emptyset$.

(3) For any $1 \leq i \neq j \leq n$, $var(H_i) \cap var(H_j) = \emptyset$.

We define $F^* := (F_1 \vee_{cl} H_1) \wedge (F_2 \vee_{cl} H_2) \wedge \cdots \wedge (F_n \vee_{cl} H_n)$. Then, F is satisfiable iff F^* is satisfiable.

Proof. ($\Rightarrow$) Assume that F is satisfiable. We have a truth assignment ν_0 over $var(F)$ such that $\nu_0(F) = 1$. It implies $\nu_0(C_i) = 1$ for each $1 \leq i \leq n$. By the proof of Lemma 1, we can extend ν_0 into a truth assignment ν_i over $var(F) \cup var(H_i)$ such that $\nu_i(F_i \vee_{cl} H_i) = 1$. By the condition (3), we can combine $\nu_1, \cdots, \nu_n$ into a truth assignment ν^* over $var(F) \cup var(H_1) \cup \cdots \cup var(H_n)$ such that $\nu^*(F_i \vee_{cl} H_i) = 1$ for each $1 \leq i \leq n$, where $\nu^*(x) := \nu_0(x)$ for $x \in var(F)$ and $\nu^*(x) := \nu_i(x)$ for $x \in var(H_i)$ $(1 \leq i \leq n)$. It means that F^* is satisfiable.

($\Leftarrow$) Assume that F^* is satisfiable. We have a truth assignment ν over $var(F) \cup var(H_1) \cup \cdots \cup var(H_n)$ such that $\nu(F^*) = 1$. It implies $\nu(F_i \vee_{cl} H_i) = 1$ for each $1 \leq i \leq n$. Please note for each $1 \leq i \leq n$ that H_i is minimal unsatisfiable and $\#cl(H_i) = \#cl(F_i) = m_i$. We have $\nu_i(H_i) = 0$ for each $1 \leq i \leq n$, where ν_i is the restriction of ν over $var(H_i)$. By the definition of $F_i \vee_{cl} H_i$ and $\nu(F_i \vee_{cl} H_i) = 1$, there exists some clause $C_{i,j}$ of F_i such that $\nu_0(C_{i,j}) = 1$, where ν_0 is the restriction of ν over $var(F)$. Since $C_{i,j}$ is a subclause of C_i, we have $\nu_0(C_i) = 1$. So, we have $\nu_0(C_i) = 1$ for each $1 \leq i \leq n$. It means that F is satisfiable.

Example 2. Let $F = [(x_1 \vee \cdots \vee x_8), (\neg x_1 \vee \cdots \vee \neg x_8), (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4)]$ be a formula. We take a minimal unsatisfiable formula $H = [(y_1 \vee y_2), (\neg y_1 \vee y_2), (\neg y_2 \vee y_3), (\neg y_2 \vee \neg y_3)]$, and copy H into two minimal unsatisfiable formulas H_1 and H_2 with $var(H_1) \cap var(H_2) = \emptyset$, and take $H_3 = [(w_1 \vee w_2), (\neg w_1 \vee w_2), \neg w_2]$. Finally, we generate the formula F^* as follows.

$$
\begin{array}{c}
x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ \hline y_1 \\ y_2 \\ y_3 \\ \hline z_1 \\ z_2 \\ z_3 \\ \hline w_1 \\ w_2
\end{array}
\left(\begin{array}{cccc|cccc|cccc}
+ & & & & - & & & & + & & & \\
+ & & & & - & & & & & - & & \\
 & + & & & & - & & & & & & + \\
 & + & & & & - & & & & & & - \\
 & & + & & & & - & & & & & \\
 & & + & & & & - & & & & & \\
 & & & + & & & & - & & & & \\
 & & & + & & & & - & & & & \\ \hline
+ & - & & & & & & & & & & \\
+ & + & - & - & & & & & & & & \\
 & & + & - & & & & & & & & \\ \hline
 & & & & + & - & & & & & & \\
 & & & & + & + & - & - & & & & \\
 & & & & & & + & - & & & & \\ \hline
 & & & & & & & & + & - & & \\
 & & & & & & & & + & + & - &
\end{array}\right)
$$

We now introduce the following four MU formulas.

(1) $A_n = [(x_1 \vee \cdots \vee x_n), (\neg x_1 \vee x_2), (\neg x_2 \vee x_3), \cdots, (\neg x_{n-1} \vee x_n), (\neg x_n \vee x_1), (\neg x_1 \vee \cdots \vee \neg x_n)] \in \mathrm{MU}(2)$. Its representation matrix is:

$$
\begin{array}{c}
x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n
\end{array}
\left(\begin{array}{ccccc}
+ & - & & & + \; - \\
+ & + & - & & - \\
\vdots & & + & & \vdots \\
\vdots & & & \cdots & \vdots \\
 & & & - & \\
+ & & & + & - \; -
\end{array}\right)
$$

We take a formula $A_n^c = [(\neg x_1 \vee x_2), (\neg x_2 \vee x_3), \cdots, (\neg x_{n-1} \vee x_n), (\neg x_n \vee x_1)]$. Clearly, both $A_n^c + \{(x_1 \vee \cdots \vee x_n)\}$ and $A_n^c + \{(\neg x_1 \vee \cdots \vee \neg x_n)\}$ are satisfiable, and $A_n^c + \{(x_1 \vee \cdots \vee x_n) \models (x_1 \wedge \cdots \wedge x_n)$ and $A_n^c + \{(\neg x_1 \vee \cdots \vee \neg x_n) \models (\neg x_1 \wedge \cdots \wedge \neg x_n)$.

Clearly, the subformula A_n^c of A_n is satisfiable, and for any truth assignment τ satisfying A_n^c it holds that $\tau(x_1) = \cdots \tau(x_n)$. The formula A_n^c presents a cycle of implication: $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n \rightarrow x_1$.

(2) $B_n = [(x_1 \vee x_3), (\neg x_1 \vee x_2), \cdots, (\neg x_s \vee x_{s+1}), \cdots, (\neg x_{n-2} \vee x_{n-1}), (\neg x_{n-1} \vee \neg x_3)] \in \mathrm{MU}(1)$, where $n \geq 6$. The representation matrix of B_6 is:

$$
\begin{array}{c}
x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5
\end{array}
\left(\begin{array}{ccccc}
+ & - & & & \\
 & + & - & & \\
+ & & + & - & - \\
 & & & + & - \\
 & & & & + \; -
\end{array}\right)
$$

Please note that $\#cl(B_n) = n$ and $\#var(B_n) = n - 1$, and B_n is a linear formula for $n \geq 6$.

(3) The standard MU formulas S_n with n variables, $x_1, \cdots, x_n$, is defined by

$$S_n = \bigwedge_{(\epsilon_1, \cdots, \epsilon_n) \in \{0,1\}^n} (x_1^{\epsilon_1} \vee \cdots \vee x_n^{\epsilon_n})$$

where $x_i^0 = x_i$ and $x_i^1 = \neg x_i$ for $1 \leq i \leq n$. Denote the clause $X_{\epsilon_1 \cdots \epsilon_n} = x_1^{\epsilon_1} \vee \cdots \vee x_n^{\epsilon_n}$.

The representation matrix of S_3 is:

$$\begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} \begin{pmatrix} + & + & + & + & - & - & - & - \\ + & + & - & - & + & + & - & - \\ + & - & + & - & + & - & + & - \end{pmatrix}.$$

The above MU formulas are useful in constructions of required formulas in this paper.

3 Construction of Linear Minimal Unsatisfiable Formulas

In this section, we introduce a subclass of CNF, called linear CNF formulas, and present a general constructing method of linear MU formulas.

Definition 1. *(1) A formula* $F \in$ CNF *is called linear if*

(a) F contains no pair of complementary unit clauses, and

(b) For all $C_i, C_j \in F$ with $C_i \neq C_j$, $|var(C_i) \cap var(C_j)| \leq 1$.

Let $LCNF$ denote the class of all linear formulas.

(2) A formula $F \in$ CNF is called exact linear if F is linear, and for all $C_i, C_j \in F$ with $C_i \neq C_j$, $|var(C_i) \cap var(C_j)| = 1$.

For example, the formula B_n is linear for $n \geq 6$. Let (XLCNF) LCNF denote the class of all (exact) linear formulas. Similarly, denote by (XLCNF$_{\geq k}$) LCNF$_{\geq k}$ the class of all (exact) linear formulas, in which formulas have only clauses of length at least $k \in N$.

Lemma 3. *Let $F = [C_1, \cdots, C_m]$ be a MU formula with $|C_i| = l_i \geq 2$ for each $1 \leq i \leq m$, and let $\{G_i = [f_1^i, \cdots, f_{l_i}^i]\}$ is a collection of linear* MU *formula where $1 \leq i \leq m$, where $var(G_i) \cap var(G_j) = \phi$ for any $1 \leq i \neq j \leq m$. Then, the formula $F^* := \bigwedge_{1 \leq i \leq m} (F_{C_i} \vee_{cl} G_i)$ is a linear* MU *formula, where F_{C_i} is the simple partition formula of clause C_i for $1 \leq i \leq m$, and $var(F) \cap (\bigcup_{1 \leq i \leq m} var(G_i)) = \phi$.*

Proof. Let $F = [C_1, \cdots, C_m]$ be a MU formula with $|C_i| = l_i \geq 2$ for each $1 \leq i \leq m$. For $1 \leq i \leq m$, we assume that $C_i = (L_{i,1} \vee \cdots \vee L_{i,l_i})$, and define a *block* formula: $F_{C_i} \vee_{cl} G_i := [(L_{i,1} \vee f_1^i), \cdots, (L_{i,l_i} \vee f_{l_i}^i)]$, where $F_{C_i} = [L_{i,1}, \cdots, L_{i,l_i}]$, and the the formula: $F^* := \bigwedge_{1 \leq i \leq m} (F_{C_i} \vee_{cl} G_i)$.

(1) F^* is minimal unsatisfiable.

Firstly, by Lemma 2, F^* is unsatisfiable since F is unsatisfiable and $G_1, \cdots, G_m$ are minimal unsatisfiable.

Secondly, F^* is minimal unsatisfiable. For any clause $g \in F^*$, w.l.o.g., we assume $g = (L_{1,1} \vee f_1^1)$, and consider the satisfiability of $F^* - \{g\}$.

Since F is minimal unsatisfiable, there exists a truth assignment τ_0 over $var(F)$ satisfying $[C_2, \cdots, C_m]$, and τ_0 forces each literal in C_1 to be false, i.e., $\tau_0(L_{1,1}) = \cdots = \tau_0(L_{1,l_1}) = 0$, and $\tau_0(C_2) = \cdots = \tau_0(C_m) = 1$. Since G_1 is minimal unsatisfiable, there exists a truth assignment τ_1 over $var(G1)$ satisfying $G_1 - \{f_1^1\}$. Thus, we have a truth assignment τ_1^* satisfying $(F_{C_1} \vee_{cl} G_1) - \{(L_{1,1} \vee f_1^1)\}$ by joining τ_0 and τ_1, where $\tau_1^*(x) = \tau_0(x)$ for $x \in var(F)$ and $\tau_1^*(x) = \tau_1(x)$ for $x \in var(G_1)$.

For each $2 \leq k \leq m$, since $\tau_0(C_k) = 1$, there is a literal L_{k,j_k} $(1 \leq j_k \leq l_k)$ such that $\tau_0(L_{k,j_k}) = 1$. By the minimal satisfiability of G_k, we have that $G_k - \{f_{j_k}^k\}$ is satisfiable. Therefore, we have a truth assignment τ_k over $var(G_k)$ satisfying $G_k - \{f_{j_k}^k\}$. Thus, we have a truth assignment τ_k^* satisfying $(F_{C_k} \vee_{cl} G_k)$ by joining τ_0 and τ_k, where $\tau_k^*(x) = \tau_0(x)$ for $x \in var(F)$ and $\tau_k^*(x) = \tau_k(x)$ for $x \in var(G_k)$.

Finally, we have a truth assignment τ^* satisfying $F^* - \{g\}$ by combining $\tau_0, \tau_1, \cdots, \tau_m$, where $\tau^*(x) = \tau_0(x)$ for $x \in var(F)$ and $\tau^*(x) = \tau_k(x)$ for $x \in var(G_k)$ $(1 \leq k \leq m)$.

(2) F^* is linear.

For any distinct clauses $f, g \in F^*$, we consider the following cases.

Case 1. Both f and g are in the same block formula.

There exists some k $(1 \leq k \leq m)$ such that $f = (L_{k,s} \vee f_s^k)$ and $g = (L_{k,s'} \vee f_{s'}^k)$ for some $1 \leq s \neq s' \leq l_k$. By $s \neq s'$, $var(f) \cap var(g) \subseteq var(f_s^k) \cap var(f_{s'}^k)$. Since G_k is linear, we have $|var(f_s^k) \cap var(f_{s'}^k)| \leq 1$. Thus, $|var(f) \cap var(g)| \leq 1$.

Case 2. f and g are in the different block formulas.

There exist some k and k' $(1 \leq k \neq k' \leq m)$ such that $f \in (F_{C_k} \vee_{cl} G_k)$ and $g \in (F_{C_k'} \vee_{cl} G_{k'})$. By constructions of block formulas, we have $f = (L_{k,s} \vee f_s^k)$ for some $1 \leq s \leq l_k$ and $g = (L_{k',s'} \vee f_{s'}^{k'})$ for some $1 \leq s' \leq l_{k'}$. By $k \neq k'$, we have $var(G_k) \cap var(G_{k'}) = \phi$. Thus, $var(f) \cap var(g) \subseteq var(L_{k,s}) \cap var(L_{k',s'})$. It implies that $|var(f) \cap var(g)| \leq 1$.

In Lemma 3, we present a method constructing MU formulas $k-$LCNF for $k \geq 3$ by S_n and B_n $(n \geq 6)$.

We consider firstly the construction of formulas for the case of $k = 3$.

We take MU formulas S_6 and B_6 with $var(S_6) \cap var(B_6) = \phi$ in Section 2. Please note that B_6 is a linear MU formula, and $|C| = 6$ for each $C \in S_6$, and $|C| = 2$ for each $C \in B_6$.

For each clause $X_{\epsilon_1 \cdots\ \epsilon_6} = (x_1^{\epsilon_1} \vee \cdots \vee x_6^{\epsilon_6}) \in S_6$, we take the simple partition formula $F_{\epsilon_1 \cdots\ \epsilon_6} = [x_1^{\epsilon_1}, \cdots, x_6^{\epsilon_6}] = x_1^{\epsilon_1} \wedge \cdots \wedge x_6^{\epsilon_6}$ of $X_{\epsilon_1 \cdots\ \epsilon_6}$, and take a copy of B_6, denoted by $B_6^{\epsilon_1 \cdots\ \epsilon_6}$, and define a formula $(F_{\epsilon_1 \cdots\ \epsilon_6} \vee_{cl} B_6^{\epsilon_1 \cdots\ \epsilon_6})$.

It restricts $var(B_6^{\epsilon_1 \cdots\ \epsilon_6}) \cap var(B_6^{\epsilon_1' \cdots \epsilon_6'}) = \phi$ for any distinct $(\epsilon_1, \cdots,\ \epsilon_6), (\epsilon_1', \cdots,\ \epsilon_6') \in \{0,1\}^6$, and $var(B_6^{\epsilon_1 \cdots\ \epsilon_6}) \cap var(S_6) = \phi$ for any $(\epsilon_1, \cdots,\ \epsilon_6) \in \{0,1\}^6$.

We now define the following formula:

$$SL_3 := \bigwedge_{(\epsilon_1, \cdots,\ \epsilon_6) \in \{0,1\}^6} (F_{\epsilon_1 \cdots\ \epsilon_6} \vee_{cl} B_6^{\epsilon_1 \cdots\ \epsilon_6}).$$

SL_3 is a linear MU formula by Lemma 3.

Please note that $\#cl(SL_3) = 6 \cdot 2^6$, and $|C| = 3$ for each $C \in SL_3$.

We define inductively a counting functions of clauses $cl(k)$ for $k \geq 3$: $cl(3) = 6 \cdot 2^6$ and $cl(k+1) = cl(k) \cdot 2^{cl(k)}$ for $k \geq 3$.

For the case of $k \geq 3$, suppose that the linear formula SL_k has been constructed such that SL_k is a linear MU formula, and the length of each clause in SL_k equals to k.

By Lemma 3, we define inductively the following linear MU formula:

$$SL_{k+1} := \bigwedge_{(\epsilon_1,\cdots,\ \epsilon_{cl(k)})\in\{0,1\}^{cl(k)}} (F_{\epsilon_1\cdots\ \epsilon_{cl(k)}} \vee_{cl} SL_k^{\epsilon_1\cdots\ \epsilon_{cl(k)}})$$

where, for $(\epsilon_1, \cdots,\ \epsilon_{cl(k)}) \in \{0,1\}^{cl(k)}$

(a) $F_{\epsilon_1\cdots\ \epsilon_{cl(k)}}$ is the simple partition formula of clause $X_{\epsilon_1\cdots\ \epsilon_{cl(k)}} \in S_{cl(k)}$.

(b) $SL_k^{\epsilon_1\cdots\ \epsilon_{cl(k)}}$ is a copy SL_k with new variables.

$S_{cl(k)}$ is minimal unsatisfiable, SL_k is both minimal unsatisfiable and linear. By Lemma 3, SL_{k+1} is a linear MU formula.

Thus, we have the following result:

Theorem 1. *For each positive integer $k \geq 3$, k-*LCNF *contains minimal unsatisfiable formulas.*

4 Conclusions and Future Works

Based on the application of minimal unsatisfiable formulas and the induction, we present a simple and general method to construct some linear formulas minimal unsatisfiable in k-CNF for each $k \geq 3$, which is stronger than the open problem whether or not there are unsatisfiable formulas in $\text{LCNF}_{\geq k}$ [3,**?**]. Based on existences of minimal unsatisfiable formula in k-LCNF for $k \geq 3$, we can show that the decision problem k-LSAT is NP-complete for $k \geq 3$. The future works is to investigate deeply structures and characterizations of linear formulas, and to apply linear formulas to analyzing complexity of resolutions and modifying effective algorithms for satisfiability.

References

1. G. Davydov, I. Davydova, H. Kleine Büning, An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF, *Annals of Mathematics and Artificial Intelligence*, 23 (1998), pp. 229-245.
2. H. Fleischner, O. Kullmann, S. Szeider, Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference,*Theoretical Computer Science*, 289(1)(2002), pp. 503-516.
3. S. Porschen, and E. Speckenmeyer, Linear CNF formulas and satisfiability, Tech. Report zaik2006-520, University Köln, 2006.
4. S. Porschen, E. Speckenmeyer, and B. Randerath, On linear CNF formulas, in: A. Biere, C. P. Gomes (eds.), Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006), LNCS 4121, 2006, pp. 212-225. (Springer, New York 2006)
5. S. Porschen, and E. Speckenmeyer, NP-completeness of SAT for restricted linear formulas classes, Proceedings of Guangzhou Symposium on Satisfiability in Logic-Based Modeling, pp. 111-123.

Improved Exponential Time Lower Bound of Knapsack Problem Under BT Model*,**

Xin Li[1], Tian Liu[1,***], Han Peng[1], Liyan Qian[1], Hongtao Sun[1], Jin Xu[1], Ke Xu[2], and Jiaqi Zhu[1]

[1] School of EECS, Beijing University, Beijing 100871, China
lt@pku.edu.cn, jxu@pku.edu.cn
[2] Department of Computer Science, Beihang University, Beijing 100083, China
kexu@nlsde.buaa.edu.cn

Abstract. M. Alekhnovich et al. have recently proposed a model of algorithms, called BT model, which covers Greedy, Backtracking and Simple Dynamic Programming algorithms and can be further divided into three kinds of fixed, adaptive and fully adaptive ones, and have proved exponential time lower bounds of exact and approximation algorithms under adaptive BT model for Knapsack problem which are about $\Omega(2^{0.5n}/\sqrt{n})$ and $\Omega((1/\epsilon)^{0.315})$(for approximation ratio $1-\epsilon$), respectively (M. Alekhovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi, Toward a Model for Backtracking and Dynamic Programming, *Proceedings of Twentieth Annual IEEE Conference on Computational Complexity*, pp308-322, 2005). In this short note, we slightly improve their lower bounds to $\Omega(2^{0.66n}/\sqrt{n})$ and $\Omega((1/\epsilon)^{0.420})$, respectively, through more complicated combinatorial arguments, and propose as an open question what is the best achievable lower bound for Knapsack problem under the adaptive BT model.

1 Introduction

Many combinatorial optimization problems are NP-complete and probably have no polynomial time algorithms [1]. It is presumed, yet has not been proved that there are only exponential time algorithms for these problems. It is very hard to prove exponential time lower bound for these problems under universal model of algorithms, unless proving $P \neq NP$. Nevertheless, under some restricted model of algorithms, it is possible to prove exponential lower bounds for NP-complete problems. Alekhnovich et al have proposed a restricted model of algorithms, called BT model, which covers greedy, backtracking and simple

* In memory of Michael Alekhnovich (1978-2006).

** This work was partially supported by National 973 Program of China (Grant Nos. G1999032701, 2002CB312004 and 2005CB321901), the National Science Foundation of China (Grant Nos. 60403003 and 30670540) and FANEDD(Grant No.200241). An earlier version of this paper appeared as arxiv:cs.CC/0606064 in June 2006.

*** Corresponding author.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 624–631, 2007.
© Springer-Verlag Berlin Heidelberg 2007

dynamic programming algorithms and can be further divided into three kinds of fixed, adaptive and fully adaptive ones , and have proved exponential lower bound for Knapsack, Vertex Cover and SAT under these models [2].

In this short note, we slightly improve the above mentioned exponential time lower bounds of exact and approximation algorithms for Knapsack problem under adaptive BT model. Although the improvement looks small and somewhat technical, it is still non-trivial, involving some more complicated combinatorial arguments, and seems to be the first one that provides such an improvement. We also propose as an open question what is the best achievable lower bound for Knapsack problem under the adaptive BT model.

The remaining parts of this note are as follows. In Section 2 we briefly introduce the BT model. In Section 3 we improve the lower bounds of knapsack problem under adaptive BT model. In Section 4 we propose as an open question about the optimality of our result.

2 A Glimpse on the BT Model

In this section we quickly recall the relevant notions from the BT model which are necessary to present our results. For more explaining and details, please refer to the original paper [2].

2.1 A Description of the BT Model

A combinatorial optimization problem P is represented by a set of data items D and a set of choices H. Each data item represents a partial structure of an instance. The set of choices contains all the possible choices which can be applied to data items. For example, in Knapsack problem, each item can be a data item and "chosen to be in the knapsack" and "chosen not to be in the knapsack" can be the set of choices. In the following, $O(S)$ denotes the set of all the orderings of all the elements in S.

A BT algorithm A to a problem P is comprised of an ordering function r_A^k and a choice function c_A^k, where

$$r_A^k : D^k \times H^k \mapsto O(D)$$

is the ordering of unprocessed data items made by algorithm after the first k data items has been processed, and

$$c_A^k : D^{k+1} \times H^k \mapsto O(H \cup \{\perp\})$$

represents the constraints made by algorithm when it makes choice for the $(k+1)$-th item according to the processed k data items and the choices for them. The algorithm consider only the choices before $\perp$. If r_A^k is a constant function and does not depend on D^k or H^k, then it's called fixed. If r_A^k depends on D^k but not on H^k, then it's called adaptive. If r_A^k depends on both D^k and H^k, then it's called fully adaptive. In this note, only adaptive BT algorithm is considered.

2.2 Lower Bound Strategies for the BT Model

The lower bound strategies for BT model takes the form of a game between the adversary and the solver (the BT algorithm) [2,3]. In the game, since at the beginning the solver cannot see all the input data items, so the adversary always tries to produce a difficult problem instance for the solver. The game can be viewed as a series of rounds. The ith round is composed of three parts: P_i, PI_i and $T_i (i = 0, 1, 2...)$. P_i is the finite set of data items owned by the adversary which cannot be seen by the solver in the ith pattern. PI_i is the set of data items representing a partial instance of the problem which have been seen by the solver in the ith pattern. And T_i is the set of partial solutions to PI_i. See Fig. 1.

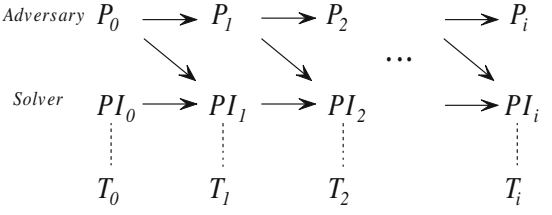


Fig. 1. Model of solver-adversary

At the beginning of the game, the solver gives an ordering rule on the input data items. And the adversary constructs rule of deleting data items according to the solver's rule and gives P_0. In this pattern, PI_0 is empty and T_0 is also empty. In the ith$(i \geq 1)$ pattern, solver picks a data item a from P_{i-1} and add it to PI_{i-1}, and gets $PI_i = PI_{i-1} \cup \{a\}$, and then computes T_i in which every solution is an extension of some solution in T_{i-1}. Then adversary deletes a from P_{i-1} and some other data items and gets P_i. This process continues until P_i is empty. In the last pattern, if PI_i is not a valid instance or T_i contains the optimal solution of PI_i, then solver wins, otherwise adversary wins.

If the set of all the solutions to PI_i can be classified into some equivalent classes and for any partial solution PS to any equivalent class, there exists an instance $A \subseteq PI_i \cup P_i$ so that every optimal solution of A contains PS, then PS is called "indispensable". If there exists a pattern in which the number of all the indispensable solutions is exponential with the scale of the problem, then the exponential lower bound of the problem can be achieved under the fixed or adaptive BT model.

3 Improving Lower Bounds of Knapsack

In this section we improve known exponential lower bounds of exact algorithm and approximation algorithm on Knapsack under adaptive BT model from [2]. We first define Knapsack Problems formally as:

Knapsack Problem

Input: n pairs of non-negative integers, $(x_1, p_1), ..., (x_n, p_n)$ and a positive integer N. x_i represents the weight of the ith item and p_i represents the value of the ith item. N is the volume of the knapsack.

Output: $S \subseteq \{1, 2, ..., n\}$, such that $\sum_{i \in S} p_i$ is maximized with respect to $\sum_{i \in S} x_i \leq N$.

Simple Knapsack Problem

Input: n non-negative integers $\{x_1, ..., x_n\}$ and a positive integer N. x_i is the weight and value of the ith item and N is the volume of the knapsack.

Output: $S \subseteq \{1, 2, ..., n\}$, such that $\sum_{i \in S} x_i$ is maximized with respect to $\sum_{i \in S} x_i \leq N$.

Simple Knapsack problem is also NP complete [1]. So it is only needed to prove the exponential lower bound for simple Knapsack. In the following, we denote a simple Knapsack problem with n items and knapsack volume of N with (n, N).

3.1 Lower Bound of Exact Algorithms

M. Alekhnovich et. al have proved the following theorem in [2].

Theorem 1. For simple Knapsack problem (n, N), the time complexity of any adaptive BT algorithm is at least

$$\binom{n/2}{n/4} = \Omega(2^{0.5n}/\sqrt{n}).$$

We will prove the following theorem, following the same line of the previous work but setting parameters differently in the proof and involving some more complicated combinatorial arguments.

Theorem 2. For simple Knapsack problem (n, N), for any $0 < \epsilon \leq 1/2$, there exists N_0, such that whenever $N > N_0$, the time complexity of any adaptive BT algorithm is at least

$$\binom{(2-\epsilon)n/3}{(2-\epsilon)n/6} = \Omega(2^{(2-\epsilon)n/3}/\sqrt{n}) \approx \Omega(2^{0.66n}/\sqrt{n}).$$

Proof: Let $I = \{0, 1, ..., \frac{3N}{n}\}$, all the weights of the items are from I.

This is an constructive proof containing three steps.

Step 1

Solver chooses the first $(2 - \epsilon)n/3$ items. After each, adversary deletes some items from the remaining by the following two rules. Let S be the set of the items which have been seen by the solver.

(1) For any $S_1, S_2 \subseteq S$, delete items with weight of $|\sum_{x \in S_1} x - \sum_{x \in S_2} x|$.

(2) For any $S_1 \subseteq S$, delete items with weight of $|N - \sum_{x \in S_1} x|$.

Step 2
Let P be the set of the first $(2-\epsilon)n/3$ items chosen by solver. Now we prove that: for all $Q \subset P$ and $|Q| = (2-\epsilon)n/6$, there exists $R \subset I - P$, such that $|R| = (1+\epsilon)n/3$ and $\sum_{x\in Q} x + \sum_{x\in R} x = N$. See Fig. 2:

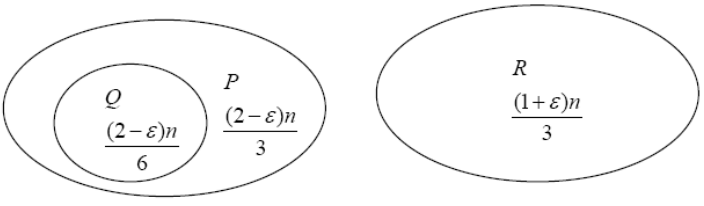


Fig. 2. Constructing an indispensable partial solution

For all $Q \subset P$, let $U = 3^n$, $a = \frac{3}{(1+\epsilon)n}(N - \sum_{x\in Q} x)$, $J = \{i | i \in \mathbb{Z} \wedge i \in [a-U, a+U]\}$ (later we will prove that $J \subset I$). Adversary chooses $\frac{(1+\epsilon)n}{3} - 2$ items in J. After each choice, adversary will delete some items from the remaining by the following two rules. Let S be the set of items chosen by solver and adversary.

(1) For all $S_1, S_2 \subseteq S$, delete items with weight of $|\sum_{x\in S_1} x - \sum_{x\in S_2} x|$.
(2) For all $S_1 \subseteq S$, delete items with weight of $|N - \sum_{x\in S_1} x|$.

Let the sum of the weighs of these $\frac{(1+\epsilon)n}{3} - 2$ items be w. Then the adversary can choose items with weight bigger or smaller than a so that

$$|w - a(n/2 - 2)| \leq U.$$

Step 3
Adversary choose a pair of items from the following $U + 1$ pairs (later we will prove that they are in I), such that the weight of these two items and the weight of the previously chosen items sum to N:

$$(\frac{v}{2} - i, \frac{v}{2} + i), i = 1, 2, ..., U + 1$$

in which $v = N - \sum_{x\in Q} x - w$.

Since in the process of choosing items, the number of all the items does not exceed n, let this number be n_t. Order these n_t items by a fixed order, then every item corresponds to a single bit of a n_t-bit 0-1-(-1) string. And every such string corresponds to a deleted item in the previous process. In detail, let the sum of the weights of items corresponding to 1 in the string be s_1, and let the sum of the weights of items corresponding to -1 in the string be s_{-1}. If $s_1 - s_{-1} > 0$, then delete item with weight of $s_1 - s_{-1} > 0$; if $s_1 = 0$, then delete item with weight of $N - s_{-1} > 0$. So the number of the deleted items does not exceed U.

So in these $U+1$ pairs of items, there must be one pair which is not deleted. Then we can choose this pair in the third step.

Next we prove all the weights of the chosen items are in I.

Because

$$a = \frac{3}{(1+\epsilon)n}(N - \sum_{x \in Q} x)$$

and

$$0 \leq \sum_{x \in Q} x \leq \frac{(2-\epsilon)n}{6} \cdot \frac{3N}{n} = \frac{(2-\epsilon)3N}{6},$$

so

$$\frac{3\epsilon N}{2(1+\epsilon)n} \leq a \leq \frac{3N}{(1+\epsilon)n},$$

clearly,

$$0 \leq \frac{3\epsilon N}{2(1+\epsilon)n} \leq a \leq \frac{3N}{(1+\epsilon)n} \leq \frac{3N}{n}.$$

Since

$$|w - a(n/2-2)| \leq U,$$

$$a = \frac{3}{(1+\epsilon)n}(N - \sum_{x \in Q} x)$$

and

$$v = N - \sum_{x \in Q} x - w,$$

so

$$|v - 2a| \leq U,$$

that is

$$a - \frac{U}{2} \leq \frac{v}{2} \leq a + \frac{U}{2}.$$

So in the third step, the possible lightest and heaviest items are $a - \frac{U}{2} - U - 1$ and $a + \frac{U}{2} + U + 1$. So in order that all the items chosen in the third step are from I, it is only needed that

$$\begin{cases} a - \frac{U}{2} - U - 1 \geq 0 \\ a + \frac{U}{2} + U + 1 \leq \frac{3N}{n} \end{cases},$$

that is

$$\begin{cases} N \geq \frac{3(1+\epsilon)n3^n + 2(1+\epsilon)n}{3\epsilon} \\ N \geq \frac{3(1+\epsilon)n3^n + 2(1+\epsilon)n}{6\epsilon} \end{cases},$$

So it is only need that $N \geq \lceil \frac{1+\epsilon}{\epsilon} \rceil n3^n$. In fact, $N \geq \lceil \frac{1+\epsilon}{\epsilon} \rceil n3^n$ is also sufficient to $J \subset I$.

Next we prove that solver must keep all Q in P as partial solution in the computation tree, otherwise adversary can keep the solver from finding the optimal solution.

We use the reduction to absurdity. Let $Q_1, Q_2 \subset P$ and $|Q_1| = |Q_2| = (2 - \epsilon)n/6$. R_1 and R_2 correspond to Q_1 and Q_2 respectively. If solver does not keep Q_1, then adversary deletes all the items except R_1. If solver keeps Q_2 to get the optimal solution, then there are only three cases (see Fig. 3):

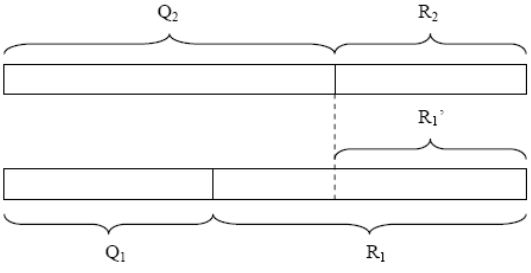


Fig. 3. Proof of indispensable of partial solution of Knapsack

(1) $\sum_{x \in Q_2} x + \sum_{x \in R_1} x < N$.

The optimal solution N cannot be achieved in this case.

(2) $\sum_{x \in Q_2} x + \sum_{x \in R_1} x = N$.

So $\sum_{x \in Q_1} x = \sum_{x \in Q_2} x$. This is impossible. Since according to the first rule in the first step, the last chosen item in the first step which is contained in Q_1 or Q_2 but not both cannot be chosen in the first step.

(3) $\sum_{x \in Q_2} x + \sum_{x \in R_1} x > N$, that is $\sum_{x \in Q_2} x > \sum_{x \in Q_1} x$.

Assume R_1' is a subset of R_1 and it leads to the optimal solution with Q_2. Then there are two subcases:

(3.1) there is only one item in $R_1 - R_1'$.

According to the first rule in the first step, this item should be deleted.

(3.2) there are two or more items in $R_1 - R_1'$.

Among the two or more items in $R_1 - R_1'$, if there is only one item which is chosen in the third step, then according to the first rule in the second step, it should be deleted.

If there are two items which are chosen in the third step, then all the items in R_1' are chosen in the second step. Let the choosing order of these items be $i_1, ..., i_s$, then according to the second rule in the second step, the last item i_s should be deleted.

So every Q in P is indispensable. So the time complexity is at least

$$\binom{(2-\epsilon)n/3}{(2-\epsilon)n/6} = \Omega(2^{(2-\epsilon)n/3}/\sqrt{n}) \approx \Omega(2^{0.66n}/\sqrt{n}).$$

■

3.2 Lower Bound of Approximation Algorithms

M.Alekhnovich et. al have proved the following theorem in [2].

Theorem 3. For simple Knapsack problem, using adaptive BT algorithm, to achieve $1-\epsilon$ approximation ratio, the time complexity is at least $\Omega((1/\epsilon)^{1/3.17})$.

We prove the following theorem:

Theorem 4. For simple Knapsack problem, using adaptive BT algorithm, to achieve $1-\epsilon$ approximation ratio, the time complexity is at least $\Omega((1/\epsilon)^{1/2.38})$.
Proof: In theorem 2 we have proved that for given n, for any $0 < \delta < 0.5$, if $N = \lceil \frac{1+\delta}{\delta} \rceil n3^n$, then time complexity of simple Knapsack problem (n, N) under adaptive BT algorithms is $\Omega(2^{\frac{(2-\delta)}{3}n}/\sqrt{n})$, so the optimal solution cannot be achieved by adaptive BT algorithm with complexity of $\gamma = o(2^{\frac{(2-\delta)}{3}n}/\sqrt{n})$. Since the weight of item is integer, so the upper bound of approximation ratio is

$$\frac{N-1}{N} \sim 1 - \frac{1}{\lceil \frac{1+\delta}{\delta} \rceil n3^n} \sim 1 - \widetilde{O}(\gamma^{-\frac{3}{2-\delta}\log_2 3}) \sim 1-\epsilon,$$

so

$$\gamma = \Omega((1/\epsilon)^{\frac{2-\delta}{4.75}}) \approx \Omega((1/\epsilon)^{1/2.38}).$$

For arbitrary n and N , we can find n_0 such that $N = \lceil \frac{1+\delta}{\delta} \rceil n_0 3^{n_0}$, and set the weights of $n - n_0$ items in the n items to be 0. Then the above lower bound works to any number of items. ■

4 Discussion

In this note, we have slightly improved the exponential time lower bounds of exact and approximation algorithms under adaptive BT model for Knapsack problem which are $\Omega(2^{(2-\epsilon)n/3}/\sqrt{n}) \approx \Omega(2^{0.66n}/\sqrt{n})$ and $\Omega((1/\epsilon)^{1/2.38}) \approx \Omega((1/\epsilon)^{0.420})$, respectively.

We do not know whether our lower bounds are optimal. An interesting question is: what is the best achievable lower bound for Knapsack problem under adaptive BT model?

Acknowledgments. We thank the anonymous reviewers of TAMC07 whose comments have greatly helped us to improve our presentations.

References

1. Garey, M.R., Johnson,D. S.: Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman and Sons, New York (1979)
2. Alekhnovich, M., Borodin, A., Buresh-Oppenheim,J., Impagliazzo, R. , Magen,A. and Pitassi,T.: Toward a Model for Backtracking and Dynamic Programming, Proceedings of Twentieth Annual IEEE Conference on Computational Complexity (2005) 308-322
3. Pudlak, P.: Proofs as games. American Math. Monthly, **23** (2000) 541–550

Phase Transition of Multivariate Polynomial Systems

Giordano Fusco and Eric Bach

Computer Sciences Department
University of Wisconsin-Madison
{fusco, bach}@cs.wisc.edu

Abstract. A random multivariate polynomial system with more equations than variables is likely to be unsolvable. On the other hand if there are more variables than equations, the system has at least one solution with high probability. In this paper we study in detail the phase transition between these two regimes, which occurs when the number of equations equals the number of variables. In particular the limiting probability for no solution is $1/e$ at the phase transition, over a prime field.

We also study the probability of having exactly s solutions, with $s \geq 1$. In particular, the probability of a unique solution is asymptotically $1/e$ if the number of equations equals the number of variables. The probability decreases very rapidly if the number of equations increases or decreases.

Our motivation is that many cryptographic systems can be expressed as large multivariate polynomial systems (usually quadratic) over a finite field. Since decoding is unique, the solution of the system must also be unique. Knowing the probability of having exactly one solution may help us to understand more about these cryptographic systems. For example, whether attacks should be evaluated by trying them against random systems depends very much on the likelihood of a unique solution.

1 Introduction

A random multivariate quadratic system in n variables is composed of m equations of the form

$$a_{11}x_1^2 + a_{12}x_1x_2 + \cdots + b_1x_1 + \cdots + b_nx_n = c,$$

where the coefficients are independently and uniformly distributed on $GF(p)$ (in the case of $p = 2$ the square terms are not present). More generally, a multivariate polynomial system can have terms up to degree d.

In this paper we study the probability that a multivariate polynomial system has no solutions. If the number of equations is much greater than the number of variables, it is very likely that the system has no solution. On the other hand if there are more variables than equations we expect to have at least one solution. For $n + \alpha$ random equations in n variables over $GF(p)$ with p prime, we show that the asymptotic probability that they have no common solution is $e^{-p^{-\alpha}}$.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 632–645, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The phase transition occurs when the number of equations equals the number of variables. The asymptotic probability in that case is $1/e$.

We also study the probability that a multivariate polynomial system has exactly s solutions, with $s \geq 1$. Asymptotically, this probability follows the Poisson distribution $\lambda^s e^{-\lambda}/s!$, where $\lambda = e^{-\alpha \log p}$. Its highest value is $e^{-1}/s!$, which is attained when the number of equations equals the number of variables. As the number of equations increases or decreases, the probability decays very rapidly.

The motivation for studying the probability of exactly s solutions comes from recent developments in cryptography. Many attacks on cryptosystems have been based on solving a large multivariate polynomial system over a finite field (some of them are [1] [5] [6]). The idea is to express the cryptosystem as a quadratic or cubic system, and then to use an ad-hoc method to solve it. The solution of this system is unique because it represents the decoded text. One of the methods used to solve these systems is called XL and it was first proposed in [5]. In [5] and in subsequent papers, it has been argued that XL takes advantage of the uniqueness of the solution. Knowing the probability of having exactly one solution, we can understand how often XL has the claimed advantage, if applied to random quadratic systems.

The quadratic systems from cryptography are not perfectly random, but in absence of a better theory we would like to get some insight by assuming that they are. In particular, the asymptotic probability that a random quadratic system has exactly one solution is $1/e$, if the number of equations equals the number of variables, and decays very rapidly if the number of equations increases or decreases.

We ran a large set of experiments to confirm the validity of our results, including some cases that are not covered by our proofs. We found that the variance of the distance between our formulas and the experimental data is small in most of the cases.

In order to apply our formulas to polynomial systems from cryptanalysis, we consider also particular configurations that occur in that case. Polynomial systems from cryptanalysis have two important properties: their equations are linearly independent and the systems are sparse. Experimental results confirm that our formulas remain valid also in this case of linearly independent equations. We generated different types of sparse systems and our formulas matched the experimental results in most of the cases.

Finally we show the results of the application of our formulas to the quadratic systems of some real cryptographic systems. Using the dimensions of those systems we determine the probability of having exactly one solution. This probability is extremely small, but on the other hand there is a huge number of possible quadratic systems of that size.

This paper is organized as follows. Section 2 gives a brief overview of related work. The probability formulas are derived in section 3. Section 4 contains the results of some experiments that confirm the general validity of our formulas. In section 5, we apply our formulas to some cryptographic systems.

2 Related Work

Given a quadratic system there is a well known procedure to determine the number of solutions. The outline of the method is the following. A single quadratic equation can be transformed into canonical form, as described by Jordan [13] for p odd, and Dickson [7] for $p = 2$. From this form it is easy to count the solutions. Then, a system of quadratic equations can be handled by counting the solutions to a number of single equations. A detailed description of this procedure for $GF(2)$ is given in [16]. This method requires exponential time.

This is not surprising, as Valiant proved in [15] that it is #P-complete to count the number of solutions of a multivariate polynomial system of degree 2 or higher.

The problem we study in this paper is different. We are not computing the number of solutions of given quadratic systems, but we are determining the probability that a random system has no solutions or exactly s solutions.

Recently, much attention has been given to unsatisfiable systems, as there is a direct connection between tautologies and unsatisfiable systems (see for example [2] [4] [3] [14]). The focus of those papers is to study proof complexity, in particular to determine under which conditions a system is unsatisfiable. Here instead we determine the probability that a random system is unsatisfiable given its size.

Woods in [16] shows that there exists a phase transition on multivariate polynomial systems, by showing that a system is unsatisfiable when the difference between the number of equations and the number of variables goes to infinity, and that the system has at least one solution when the difference between the number of variables and the number of equations goes to infinity. In this paper we study the phase transition in more detail, in particular we determine the point in which the phase transition occurs and the limiting value of the probability near the transition point.

To our knowledge this is the first detailed study of phase transitions in polynomial systems. However, there is a well known phase transition between satisfiability and unsatisfiability for boolean formulas, which has been studied both theoretically and experimentally. We believe that Friegut [8] was the first one to prove the existence of a phase transition in boolean formulas. More details have been found experimentally, and surveys of this work have been given by Franco, in [9] and [10]. Our results do say something about boolean formulas, since a boolean formula in conjunctive normal form can be easily transformed into a quadratic system over $GF(2)$ (see for example [12]). However, they are more general, in that we consider polynomial systems of any degree and for any prime field. We also have rigorous theorems to support our experimental observations.

3 Probability of No Solutions and of Exactly s Solutions

The following theorems comprise our main result. Theorem 1 is a special case of theorem 2, but we preferred to state it separately to emphasize the phase transition.

Theorem 1. *Let $d \geq 2$ and p be a prime number. Given a multivariate polynomial system of $n + \alpha$ random equations of degree-d in n variables over $GF(p)$, the probability that the system has no solution is $e^{-p^{-\alpha}}$, asymptotically in n.*

Corollary 1. *For a system as in theorem 1, the probability of no solution is e^{-1}, if the number of equations equals the number of variables (i.e. $\alpha = 0$).*

Theorem 2. *Let $d \geq 2$ and p be a prime number and $\lambda = e^{-\alpha \log p}$. Given a multivariate polynomial system of $n + \alpha$ random equations of degree-d in n variables in $GF(p)$, the probability that the system has exactly $s \geq 1$ solutions follows the Poisson distribution $\lambda^s e^{-\lambda}/s!$ asymptotically in n.*

Corollary 2. *For a system as in theorem 2, the probability that the system has exactly $s \geq 1$ solutions is $e^{-1}/s!$, if the number of equations equals the number of variables (i.e. $\alpha = 0$).*

The rest of this section contains the proofs of these results.

Proof of theorem 1. Let p be a prime, and for an n-tuple $x = (x_1, \ldots, x_n)$ of elements from $GF(p)$ let $R_x = (1, \cdots, x_r, \cdots, x_r x_s, \cdots)$. For a system of degree d, R_x contains the monomials up to degree d. Let G be the $p^n \times \nu$ matrix whose rows are the R_x for distinct x, where $\nu \approx n^d$ is the number of coefficients in each equation.

Consider the indicator variable

$$Z_x = \begin{cases} 1, & \text{if } x \text{ is a solution to all equations;} \\ 0, & \text{otherwise} \end{cases}$$

Its expectation is $E[Z_x] = p^{-(n+\alpha)}$, and the probability that there is no common solution is

$$E[\prod_x (1 - Z_x)].$$

By the inclusion-exclusion principle we have

$$\prod_x (1 - Z_x) \geq 1 - \sum_x Z_x,$$
$$\prod_x (1 - Z_x) \leq 1 - \sum_x Z_x + \sum_{x,y} Z_x Z_y,$$
$$\prod_x (1 - Z_x) \geq 1 - \sum_x Z_x + \sum_{x,y} Z_x Z_y - \sum_{x,y,z} Z_x Z_y Z_z,$$

and so on. Any partial sum with an even (resp. odd) number of terms provides a lower (resp. upper) bound. Also, in these sums, the indices x, y, z, etc. refer to distinct n-tuples, so each term is effectively a sum over subsets.

Now consider a typical term in the above sum, such as

$$\sum_{x^{(1)}, \ldots, x^{(k)}} \prod_i Z_{x^{(i)}}$$

Its expected value is

$$\sum_{x^{(1)},\ldots,x^{(k)}} E\left[\prod_i Z_{x^{(i)}}\right] \tag{1}$$

A subset for which the corresponding $Z's$ are stochastically independent will contribute $p^{-k(n+\alpha)}$ to the sum. We need to show that most of the subsets are of this type. We say that a subset $\{x^{(1)},\ldots,x^{(k)}\}$ is in *general position* if the extended vectors $(1,x^{(1)}),\ldots,(1,x^{(k)})$ are linearly independent. Observe that for any general position subset, the random variables $Z_{x^{(i)}}$, are stochastically independent. The number of general position subsets is

$$\frac{p^n(p^n-1)(p^n-p)\ldots(p^n-p^{k-2})}{k!}$$

Hence, the general position subsets contribute, for large n, the value

$$\frac{p^{nk}}{k!}p^{-k(n+\alpha)} = \frac{p^{-\alpha k}}{k!}$$

If all the rows of G were linearly independent then all subsets would be in general position. Unfortunately this is not true. However, by lemma 1 below, the contribution from subsets not in general position is insignificant compared to this.

Let k^* be the largest odd integer not bigger than ν. For quadratic systems, k^* is approximatively $n^2/2$, and in general, k^* goes to infinity with n. Then, if

$$\delta = \Pr[\text{ no solution }] - \sum_{k=0}^{k^*-2}\frac{p^{-\alpha k}}{k!},$$

we have

$$-\frac{p^{-\alpha(k^*-1)}}{(k^*-1)!}(1+o(1)) \le \delta \le \frac{p^{-\alpha k^*}}{k^*!}(1+o(1))$$

By Stirling's formula, the upper and lower bounds go to 0 as $n\to\infty$, and the sum is the Taylor series for the (entire) exponential function, so the limit of δ is 0, and we conclude

$$\lim_{n\to\infty}\Pr[\text{ no solution }] = e^{-p^{-\alpha}}$$ □

Proof of theorem 2. The indicator for exactly s solutions is

$$I = \sum_{x^{(1)}} Z_{x^{(1)}} \sum_{x^{(2)}\neq x^{(1)}} Z_{x^{(2)}} \cdots \sum_{\substack{x^{(s)}\neq x^{(1)} \\ \cdots \\ x^{(s)}\neq x^{(s-1)}}} Z_{x^{(s)}} \prod_{\substack{y\neq x^{(1)} \\ \cdots \\ y\neq x^{(s)}}} (1-Z_y).$$

If we expand this and collect terms, we get

$$\sum_{k\ge 0}(-1)^k\binom{s+k}{k}\sum_{x^{(1)},\ldots,x^{(s+k)}} Z_{x^{(1)}}\cdots Z_{x^{(s+k)}}.$$

As before, the k-th inner sum is over the subsets of $GF(p)^n$ of size $(s+k)$.

For each n, this expansion of I is a finite sum. Furthermore, the Z's are all non-negative, so taking its expectation produces an alternating series. We can therefore evaluate the limit of these expectations by computing limits termwise as we did in the proof of theorem 1.

So let us consider a particular value of k. The number of general position subsets of size $s+k$ is given by

$$\frac{p^n(p^n-1)\cdots(p^n-p^{s+k-2})}{(s+k)!}$$

Therefore, their contribution to the expectation is asymptotically

$$\frac{p^{n(s+k)}p^{-(s+k)(n+\alpha)}}{(s+k)!} = \frac{p^{-(s+k)\alpha}}{(s+k)!}$$

Using lemma 1 below with k replaced by $s+k$, we see that including subsets not in general position will not change the value of this limit.

Arguing as before, the expectation of I has the limit

$$\sum_{k\geq 0}(-1)^k\binom{s+k}{k}\frac{p^{-(s+k)\alpha}}{(s+k)!} = \frac{p^{-s\alpha}}{s!}\sum_{k\geq 0}(-1)^k\frac{p^{-k\alpha}}{k!}$$

The value of the last sum is $e^{-p^{-\alpha}} = e^{-\lambda}$, and this gives the desired result. □

The following lemma is used in the proof of theorems 1 and 2. It is the key device for our analysis, as it allows us to compute limiting probabilities as if we had full independence.

Lemma 1. *Let p be a prime, and for an n-tuple $x = (x_1, \ldots, x_n)$ of elements from $GF(p)$ let $R_x = (1, \cdots, x_r, \cdots, x_r x_s, \cdots)$. Let G be the $p^n \times \nu$ matrix whose rows are the R_x for distinct x. For fixed k and $n \to \infty$, the contribution to (1) from subsets not in general position goes to 0.*

Proof of lemma 1. The points $x^{(0)}, \ldots, x^{(\ell)}$ in $GF(p)^n$ are *affinely independent* if the differences $x^{(1)} - x^{(0)}, \ldots, x^{(\ell)} - x^{(0)}$ are linearly independent. For distinct points, this happens iff the corresponding subset is in general position.

Let S be a particular k-subset of the rows of G, corresponding to a set of k points. Let $\ell+1$ be the maximum number of points in S that are affinely independent. We may choose coordinates so that the rows for these points are

$$\begin{array}{cccccccccccccc}
 & x_1 & & x_\ell & x_{\ell+1} & & x_n & x_1^2 & & x_n^2 & & x_i x_j & \\
 & & & & & & & & & & & & \\
1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots \\
1 & 1 & \cdots & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots \\
 & & & & & \vdots & & & & & & & \\
1 & 0 & \cdots & 1 & 0 & \cdots & 0 & 0 & \cdots & 1 & \cdots & 0 & \cdots
\end{array} \tag{2}$$

for a quadratic system. In general, for a degree-d system the coordinate would follow a similar pattern.

Assume that $\ell + 1 < k$, that is, the subset S is degenerate. We claim that the rank of S cannot be $\ell + 1$. (It is obviously at least this large.) If it were, then any other row would be of the form

$$1\ w_1 \cdots\ w_\ell\ 0\ \cdots\ 0\ w_1^2 \cdots\ w_n^2 \cdots\ w_i w_j \cdots$$

Since it is a linear combination of rows from (2), we must have all $w_i w_j = 0$. This means that at most one w_i, say w_1, is nonzero. Then we would have (from inspection of the x_1 and x_1^2 columns)

$$w_1 \cdot 1 = w_1^2$$

So $w_1 \in \{0, 1\}$, but this is impossible since the rows came from distinct points. Hence, the rank is at least $\ell + 2$.

For a fixed value of ℓ, there will be at most

$$p^n(p^n - 1)(p^n - p) \cdots (p^n - p^{\ell-1}) \times p^{k^2}$$

such degenerate subsets of rows. The first factor is an upper bound for the number of ways to choose $\ell+1$ affinely independent points, and the second factor follows from $\ell \leq k$ and affine independence. (Once we have chosen coordinates, only w_i with $i \leq \ell$ are eligible to be nonzero.) As $n \to \infty$, we have

$$p^n(p^n - 1) \cdots (p^n - p^{\ell-1}) \times p^{k^2} \sim p^{(\ell+1)n+k^2}$$

Now, if a collection of rows has rank r, the probability of choosing coefficients so that a degree-d function vanishes at the points corresponding to those rows is p^{-r}. Similarly, the probability of choosing m sets of coefficients independently with the same property is p^{-mr}. So, for any fixed ℓ, the contribution of degenerate subsets to (1) is at most

$$\frac{p^{(\ell+1)n+k^2}}{p^{mr}} \leq \frac{p^{(\ell+1)n+k^2}}{p^{m(\ell+2)}}$$

This is because of our rank estimate. If we substitute $m = n + \alpha$, this becomes

$$\frac{p^{k^2-(\ell+2)\alpha}}{p^n}$$

which has the limit 0 as n goes to infinity. □

3.1 Extension of the Results to $\mathbb{Z}/(pq)$

In this section we derive the probability formulas for $\mathbb{Z}/(pq)$ where p and q are distinct primes.

Theorem 3. *Given a multivariate polynomial system of $n+\alpha$ random equations of degree-d in n variables in $\mathbb{Z}/(pq)$ with p and q distinct primes, the probability that the system has no solution is $e^{-p^{-\alpha}} + e^{-q^{-\alpha}} - e^{-(p^{-\alpha}+q^{-\alpha})}$, asymptotically in n.*

Corollary 3. *For a system as in theorem 3, the limiting probability of no solution is* $2e^{-1} - e^{-2}$*, if the number of equations equals the number of variables (i.e.* $\alpha = 0$*).*

Theorem 4. *Let* $\lambda = e^{-\alpha \log p}$ *and* $\mu = e^{-\alpha \log q}$*. Given a multivariate polynomial system of* $n + \alpha$ *random equations of degree-d in* n *variables in* $\mathbb{Z}/(pq)$ *with* p *and* q *distinct primes, the limiting probability that the system has exactly* $s \geq 1$ *solutions is*

$$e^{-\lambda-\mu} \sum_{\substack{uv=s \\ u,v\geq 1}} \frac{\lambda^u \mu^v}{u!\, v!}$$

asymptotically in n*.*

Corollary 4. *For a system as in theorem 4, the limiting probability that the system has exactly* $s \geq 1$ *solutions is* $e^{-2} \sum_{u,v\geq 1}^{uv=s} \frac{1}{u!\, v!}$*, if the number of equations equals the number variables (i.e.* $\alpha = 0$*).*

Proof of theorem 3. A solution does not satisfy the system modulo pq if it does not satisfy it modulo p or it does not satisfy it modulo q. But this way, we are double counting the probability that it does not satisfy it both modulo p and modulo q.

The probability that there are no solutions modulo p and no solutions modulo q is the product of these two probabilities:

$$e^{-p^{-\alpha}} \cdot e^{-q^{-\alpha}} = e^{-(p^{-\alpha}+q^{-\alpha})}$$

The probability that there are no solutions modulo pq is the sum of the probability of having no solutions modulo p and no solutions modulo q, minus the probability of no solutions modulo p and modulo q together.

$$e^{-p^{-\alpha}} + e^{-q^{-\alpha}} - e^{-(p^{-\alpha}+q^{-\alpha})} \qquad \square$$

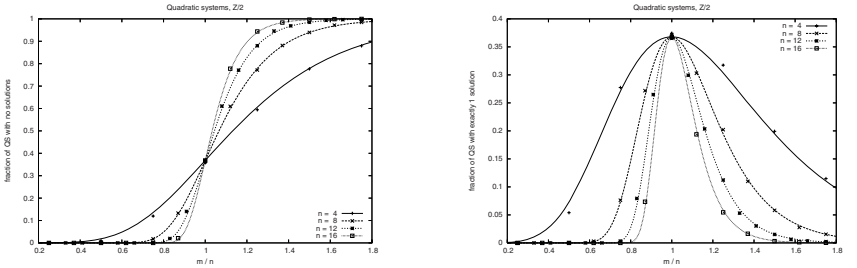
Note that the previous result can be further extended to products of many distinct primes, by using the inclusion-exclusion principle.

Proof of theorem 4. To have exactly s solutions modulo pq, we must have u solutions mod p and v solutions mod q, where $uv = s$. For different factorizations of s, these events are disjoint. Therefore, for n going to infinity, the probability is

$$e^{-\lambda-\mu} \sum_{\substack{uv=s \\ u,v\geq 1}} \frac{\lambda^u \mu^v}{u!\, v!} \qquad \square$$

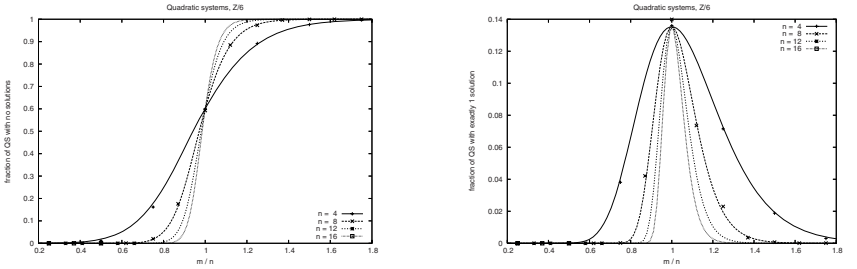
4 Experimental Results

We ran a large set of experiments, which confirm the validity of our results also in the cases that are not covered by our proofs. We generated 10,000 random polynomial systems for each configuration and we counted the number of solutions in



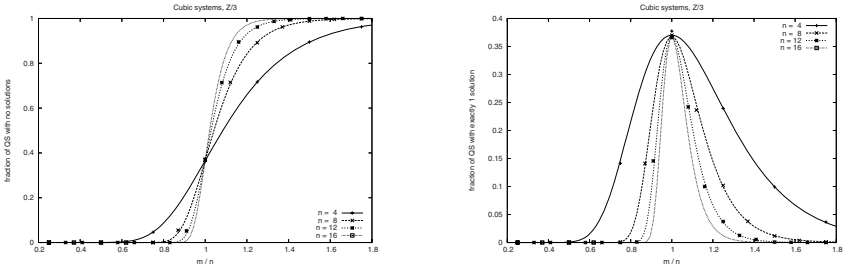
(*a*) (*b*)

Fig. 1. Fraction of quadratic systems with (*a*) no solutions, and (*b*) exactly one solution in $\mathbb{Z}_2$



(*a*) (*b*)

Fig. 2. Fraction of quadratic systems with (*a*) no solutions, and (*b*) exactly one solution in $\mathbb{Z}_6$



(*a*) (*b*)

Fig. 3. Fraction of cubic systems with (*a*) no solutions, and (*b*) exactly one solution in $\mathbb{Z}_3$

each case. Figure 1*a* shows the fraction of quadratic systems with no solutions in $\mathbb{Z}_2$. Figure 1*b* shows the fraction of quadratic systems with exactly one solution in $\mathbb{Z}_2$. The continuous line represents the value of the functions described in the previous section, while the discrete symbols give results from the experiments. We can see that the experimental results are consistent with the formulas even in the case of a small number of variables. This is better than what we were

Table 1. Variance of experimental values with respect to the formulas for (a) uniform at random equation, (b) linearly independent equations

	no solutions	1 solution
$\mathbb{Z}_2$	$1.66 \cdot 10^{-5}$	$1.95 \cdot 10^{-5}$
$\mathbb{Z}_3$	$7.30 \cdot 10^{-6}$	$7.56 \cdot 10^{-6}$
$\mathbb{Z}_5$	$2.48 \cdot 10^{-6}$	$1.74 \cdot 10^{-6}$
$\mathbb{Z}_6$	$1.54 \cdot 10^{-5}$	$1.60 \cdot 10^{-6}$
$\mathbb{Z}_7$	$2.00 \cdot 10^{-6}$	$2.78 \cdot 10^{-6}$

(a)

	no solutions	1 solution
$\mathbb{Z}_2$	$1.79 \cdot 10^{-5}$	$2.09 \cdot 10^{-5}$
$\mathbb{Z}_3$	$7.30 \cdot 10^{-6}$	$7.56 \cdot 10^{-6}$
$\mathbb{Z}_5$	$2.58 \cdot 10^{-6}$	$1.82 \cdot 10^{-6}$
$\mathbb{Z}_6$	$1.65 \cdot 10^{-5}$	$1.72 \cdot 10^{-6}$
$\mathbb{Z}_7$	$2.89 \cdot 10^{-6}$	$4.02 \cdot 10^{-6}$

(b)

expecting, because the formulas were derived for n going to infinity. Figures 2a and 2b show similar results for $\mathbb{Z}_6$. Figures 3a and 3b show that similar results hold for cubic systems. Table 1a shows the variance of the experimental values with respect to the formulas for the quadratic systems. The data of this table is obtained varying n from 4 to 16 and m from 1 to 28.

These experiments were designed to investigate a range of applicability wider than the one considered in our theorems. The fact that the variance is small makes us believe that our theorems are valid more generally than our proofs would indicate.

4.1 Linearly Independent Equations

We considered the case of non-linear systems with linearly independent equations. This is motivated by the fact that the quadratic systems used in cryptanalysis have only linearly independent equations.

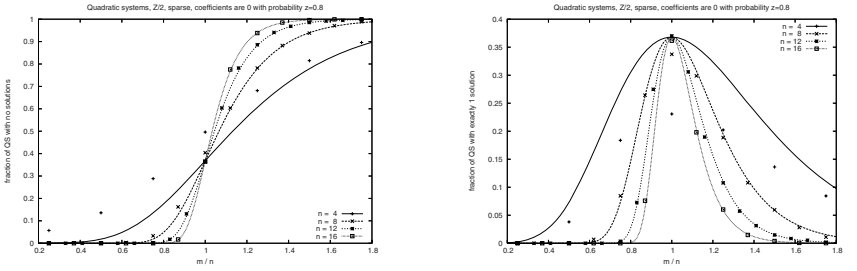
The formulas derived in section 3 hold in the case of linearly independent equations also. This is because the equations of a random polynomial system are linearly independent with very high probability. In fact a system of degree q with n variables has more than n^q coefficients, which implies that the matrix of the coefficients is rectangular even when we consider $m > n$. As shown in [11], it is very likely that a random rectangular matrix has maximal rank.

This is confirmed by the experimental data. We ran the same experiment as the one described at the beginning of section 4, but enforcing that the equations must be linearly independent, by eliminating the linearly dependent equations. As we can see from table 1b the variance is very small in this case also.

4.2 Sparse Systems

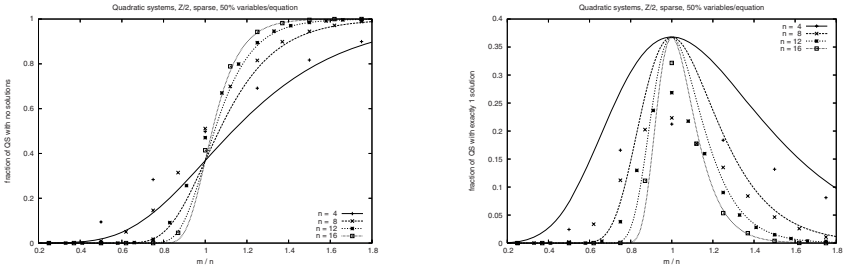
In this section we check our formulas on sparse systems. Again the motivation comes from cryptanalysis, where the quadratic systems are usually sparse. In order to simulate the sparseness, we consider three kind of sparse systems:

1. Each coefficient can be 0 with probability z and non zero with probability $1 - z$. Note that the known term can still assume any value with equal probability.
2. Each equation contains exactly a fraction f of the variables, i.e. the coefficients of the remaining variables are 0.



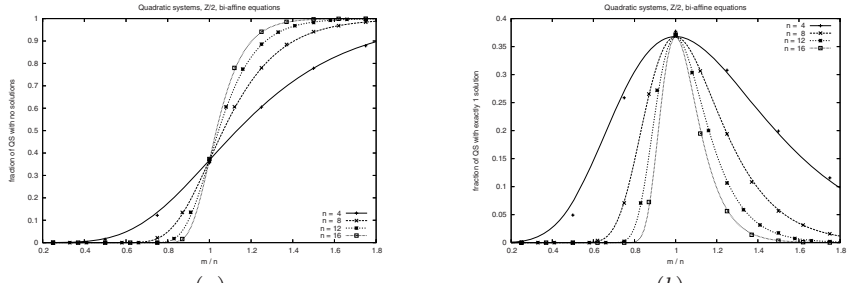
(*a*) (*b*)

Fig. 4. Fraction of quadratic systems with (*a*) no solutions, and (*b*) exactly one solution, in $\mathbb{Z}_2$ with coefficients set to 0 with probability $z = 2/3$



(*a*) (*b*)

Fig. 5. Fraction of quadratic systems with (*a*) no solutions, and (*b*) exactly one solution, in $\mathbb{Z}_2$ with exactly 50% variables per equation



(*a*) (*b*)

Fig. 6. Fraction of quadratic systems with (*a*) no solutions, and (*b*) exactly one solution, in $\mathbb{Z}_2$ with bi-affine equations

3. Bi-affine equations. These are the type of equations used in the cryptanalysis of Rijndael (see for example [6]).

Case 1: the coefficients have higher probability to be 0. This is the most generic type of sparseness that we are considering. The variance between the formulas from section 3 and the experimental results is very small for values of z up to 0.7.

Table 2. Variance of experimental values with respect to the formulas for (a) different values of z with random system in $\mathbb{Z}_3$, and (b) for different fields when the coefficients are zero with probability $z = 2/3$

z	no solutions	1 solution
0.5	$3.66 \cdot 10^{-6}$	$6.30 \cdot 10^{-6}$
0.6	$1.62 \cdot 10^{-5}$	$9.04 \cdot 10^{-6}$
0.7	$7.14 \cdot 10^{-5}$	$3.21 \cdot 10^{-5}$
0.8	$1.82 \cdot 10^{-3}$	$6.81 \cdot 10^{-4}$
0.9	$1.32 \cdot 10^{-2}$	$3.31 \cdot 10^{-3}$

(a)

	no solutions	1 solution
$\mathbb{Z}_2$	$3.74 \cdot 10^{-5}$	$3.27 \cdot 10^{-5}$
$\mathbb{Z}_3$	$8.05 \cdot 10^{-5}$	$3.62 \cdot 10^{-5}$
$\mathbb{Z}_5$	$1.54 \cdot 10^{-4}$	$7.81 \cdot 10^{-5}$
$\mathbb{Z}_6$	$1.19 \cdot 10^{-3}$	$7.17 \cdot 10^{-5}$
$\mathbb{Z}_7$	$1.27 \cdot 10^{-4}$	$4.00 \cdot 10^{-5}$

(b)

Table 3. Variance of experimental values with respect to the formulas for (a) random system in $\mathbb{Z}_3$ generated varying f from 0.1 to 0.5, and (b) for different fields when f is fixed to 0.5. (c) Variance of experimental values with respect to the formulas for different fields with bi-affine equations.

f	no solutions	1 solution
0.1	$4.16 \cdot 10^{-1}$	$1.74 \cdot 10^{-2}$
0.2	$2.58 \cdot 10^{-1}$	$1.65 \cdot 10^{-2}$
0.3	$8.78 \cdot 10^{-2}$	$1.45 \cdot 10^{-2}$
0.4	$3.65 \cdot 10^{-3}$	$9.54 \cdot 10^{-3}$
0.5	$5.21 \cdot 10^{-3}$	$2.87 \cdot 10^{-3}$

(a)

	no solutions	1 solution
$\mathbb{Z}_2$	$3.34 \cdot 10^{-3}$	$2.56 \cdot 10^{-3}$
$\mathbb{Z}_3$	$5.33 \cdot 10^{-3}$	$2.94 \cdot 10^{-3}$
$\mathbb{Z}_5$	$9.17 \cdot 10^{-2}$	$4.11 \cdot 10^{-3}$
$\mathbb{Z}_6$	$1.87 \cdot 10^{-2}$	$1.02 \cdot 10^{-3}$
$\mathbb{Z}_7$	$1.91 \cdot 10^{-2}$	$7.90 \cdot 10^{-3}$

(b)

	no solutions	1 solution
$\mathbb{Z}_2$	$9.17 \cdot 10^{-6}$	$2.00 \cdot 10^{-5}$
$\mathbb{Z}_3$	$2.63 \cdot 10^{-5}$	$3.99 \cdot 10^{-5}$
$\mathbb{Z}_5$	$1.24 \cdot 10^{-6}$	$9.04 \cdot 10^{-6}$
$\mathbb{Z}_6$	$4.14 \cdot 10^{-5}$	$1.23 \cdot 10^{-5}$
$\mathbb{Z}_7$	$4.10 \cdot 10^{-6}$	$5.10 \cdot 10^{-6}$

(c)

Table 2a shows how the variance varies using different values of z with random system in $\mathbb{Z}_3$. A similar situation is obtained in other prime fields. Table 2b shows the value of the variance of random systems in different fields where the coefficients are zero with probability $z = 2/3$.

If z is smaller than 0.7, the results are very similar to figures 1a and 1b. Figures 4a and 4b show the result obtained with random systems in $\mathbb{Z}_2$ where the coefficients are zero with probability $z = 0.8$. As we can see in the plot, the formula does not approximate well a system with $n = 4$ variables, but it still works for bigger values of n.

Case 2: each equation contains exactly a fraction f of the variables. In this case the variance from the experiments is much higher. Table 2a shows the values of the variance of random system in $\mathbb{Z}_3$ generated varying f from 0.1 to 0.5. Similar results are obtained in other fields where f is fixed to 0.5, as shown in table 2b.

An explanation of these results is that this model reduces the freedom of the random equations, which in fact are no longer perfectly uniform at random. For this reason the formulas no longer exactly describe the phenomenon and the variance from the experiment is much higher. This is also evident from figures 5a and 5b.

Case 3: bi-affine equations. Bi-affine equations are used only for quadratic systems. The variables are partitioned into two sets of equal size. Each quadratic

Table 4. (a) Sizes of quadratic systems from cryptography. (b) Total number of quadratic systems and probability of exactly 1 solution.

Cryptosystem	n	m	α
Khazad	6464	7664	1200
Misty1	3856	3856	0
Kasumi	4264	4264	0
Camellia-128	3584	6224	2640
Rijndael-128	3296	6296	3000
Serpent-128	16640	17680	1040

(a)

Cryptosystem	Total # of systems	Pr[1 solution]
Khazad	$6.86 \cdot 10^{6249185}$	$5.81 \cdot 10^{-362}$
Misty1	$1.68 \cdot 10^{2239709}$	$1/e$
Kasumi	$4.20 \cdot 10^{2738543}$	$1/e$
Camellia-128	$1.64 \cdot 10^{1934992}$	$1.91 \cdot 10^{-795}$
Rijndael-128	$5.40 \cdot 10^{1636625}$	$8.13 \cdot 10^{-904}$
Serpent-128	$3.58 \cdot 10^{41683551}$	$8.49 \cdot 10^{-314}$

(b)

term is composed of a variable from the first set and one from the second (i.e. two variables from the same set never appear multiplied together). The variance in this case is small as we can see from table 3c. The results for $\mathbb{Z}_2$ are plotted in figures 6a and 6b.

5 Equations from Cryptographic Systems

In this section we apply the formula for exactly one solution to the sizes of quadratic systems for some well known cryptographic systems. The results obtained with the experimental data (see section 4) give us confidence in using the formula in this case, even if this is not a case covered by our proofs. The data in the table 4a is from [1]. All the equations are in $\mathbb{Z}_2$. For the quadratic systems of Misty1 and Kasumi, the parameters m and n are in the range of applicability of our formulas.

In table 4b we can see that for many systems the probability of having exactly one solution is extremely small. However the number of systems with exactly one solution is not that small, if we consider that the total number of possible systems is huge.

One inference that can be drawn from this study is that quadratic systems with unique solutions are relatively rare, so rare that in most cases, studying the performance of solution algorithms for random systems might not tell us much about their efficacy in attacking specific cryptosystems.

6 Conclusions and Open Problems

We showed that the probability that a random polynomial system has no solution has a phase transition when the number of equations equals the number of variables. The value of the probability at the phase transition is $1/e$ if the computation is over a prime field.

We showed that probability of having exactly s solution, $s \geq 1$, follows a Poisson distribution with parameter $\lambda = e^{-\alpha \log p}$, for prime fields.

We extended the result to $\mathbb{Z}/(pq)$, with p and q distinct primes. It is an open problem to extend the result to the case of $\mathbb{Z}/(p^r)$, with p prime.

It is an open problem to adapt the formulas in the case of sparse systems where each equation contains exactly a fixed number of variables.

Acknowledgments

We would like to thank Dieter van Melkebeek and Janos Simon for their helpful comments. We also would like to thank NSF grant CCF-0523680.

References

1. A. Biryukov, C. De Cannière, *Block ciphers and systems of quadratic equations*, Proc. FSE 2003, LNCS 2887, pp. 274–289, 2003.
2. P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, *Lower bounds on Hilbert's Nullstellensatz and propositional proofs*, Proc. London Math. Soc., n. 73, pp. 1–26, 1996.
3. S. Buss, R. Impagliazzo, J. Krajíček, A.A. Razborov, J. Sgall, *Proof complexity in algebraic systems and bounded depth Frege systems with modular counting*, Comput. Complex., n. 6, pp. 256–298, 1997.
4. M. Clegg, J. Edmonds, R. Impagliazzo, *Using the Groebner basis algorithm to find proofs of unsatisfiability*, Proc. 28th Ann. ACM Symp. Theory Comput., pp. 174–183, 1996
5. N. Courtois, A. Klimov, J. Patarin, A. Shamir, *Efficient algorithms for solving overdefined systems of multivariate polynomial equations*, Proc. Eurocrypt 2000, LNCS 1807, pp. 392–407, 2000.
6. N. Courtois, J. Pierpzyk, *Cryptanalysis of block ciphers with overdefined systems of equations*, Proc. Asiacrypt 2002, LNCS 2501, pp. 267–287, 2002.
7. L. E. Dickson, *Determination of the structure of all linear homogeneous groups in a Galois field which are defined by a quadratic invariant*, Amer. J. Math., v. 21, pp. 193–256, 1899.
8. E. Friegut, *Necessary and sufficient conditions for sharp thresholds of graph properties and the k-SAT problem*, Amer. J. Math., v. 12, pp. 1017–1054, 1999.
9. J. Franco, *Results related to threshold phenomena research in satisfiability: lower bounds*, Theoret. Comput. Sci., v. 265, n. 1–2, pp. 147–157, 2001.
10. J. Franco, *Typical case complexity of satisfiability algorithms and the threshold phenomenon* Disc. Appl. Math., v. 153, n. 1–3: pp. 89–123, 2005.
11. F. Gerth III, *Limit probabilities for coranks of matrices over GF(q)*, Lin. Multilin. Alg., v. 19, pp. 79–93, 1986.
12. J. Håstad, S. Phillips, S. Safra, *A well-characterized approximation problem*, Inf. Proc. Lett., v. 47, n. 6, pp. 301–305, 1993.
13. C. Jordan, *Sur la forme canonique des congruences du second degré et le nombre de leurs solutions*, J. Math. Pures. Appls. (2), v. 17, pp. 368–402, 1872. [Abstract of results in C. R. Acad. Sci. Paris, v. 74, pp. 1093–1095, 1872.]
14. T. Pitassi, *Algebraic propositional proof systems*, Descriptive Complexity and Finite Models, v. 31 of DIMACS Ser. Discrete Math. Thoret. Comput. Sci. pp. 215–244, 1997.
15. L.G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput. v. 8, pp. 4120–421, 1979.
16. A. R. Woods, *Unsatisfiable systems of equations, over a finite field*, Proc. 39th Ann. Symp. Found. Comput. Sci., pp. 202–211, 1998.

Approximation Algorithms for Maximum Edge Coloring Problem*

Wangsen Feng, Li'ang Zhang, Wanling Qu, and Hanpin Wang

School of Electronic Engineering and Computer Science
Peking University, Beijing 100871, P.R. China
{fengws,zliang,qwl,whpxhy}@pku.edu.cn

Abstract. We propose polynomial time approximation algorithms for a novel maximum edge coloring problem which arises from the field of wireless mesh networks [8]. The problem is about coloring all the edges in a graph and finding a coloring solution which uses the maximum number of colors with the constraint, for every vertex in the graph, all the edges incident to it are colored with no more than $q(q \in \mathbb{Z},\ q \geq 2)$ colors. The case $q = 2$ is of great importance in practice. In this paper, we design approximation algorithms for cases $q = 2$ and $q > 2$ with approximation ratio 2.5 and $1 + \frac{4q-2}{3q^2-5q+2}$ respectively. The algorithms can give practically usable estimations on the upper bounds of the numbers of the channels used in wireless mesh networks.

1 Introduction

1.1 Problem Definition

Graph coloring problems occupy an important place in graph theory. Generally, there are two types of coloring: vertex coloring and edge coloring. For vertex coloring, Brooks [1] states that $\chi(G) \leq \Delta(G)$ for any graph G except complete graphs K_n and odd circles C_{2k+1}, where *chromatic number* $\chi(G)$ is the minimum number of colors needed in a vertex coloring of graph G. Karp [2] proves that to determine $\chi(G)$ is an NP-hard problem . If P $\neq$ NP holds, Garey and Johnson [3] point out that there is even no polynomial time approximation algorithm with ratio 2. However, Turner [4] designs an algorithm of complexity $O(|V|+|E|logk)$ and with probability almost 1 to color a given k-colorable graph with k colors for the case that k is not too large relative to $|V|$. For edge coloring, Vizing [5] states that for any graph G, either $\chi^{'}(G) = \Delta(G)$ or $\chi^{'}(G) = \Delta(G)+1$, where *chromatic index* $\chi^{'}(G)$ is the minimum number of colors needed in an edge coloring of G. Holyer [6] proves that it is also an NP-hard problem to determine $\chi^{'}(G)$. The proof of Vizing Theorem yields an approximation algorithm for this problem

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312004 and the National High-tech Research and Development 863 Program of China under Grant No. 2006AA01Z160.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 646–658, 2007.
© Springer-Verlag Berlin Heidelberg 2007

which finds an edge coloring solution using $\Delta(G)+1$ kinds of colors within one of optimal. Recently, Uriel Feige et al. [7] have investigated the maximum edge t-coloring problem in multigraphs. The problem is to color as many edges as possible using t colors, such that no pairs of adjacent edges are colored with the same color. They show that the problem is NP-hard and further design constant factor approximation algorithms for it.

The problems mentioned above are all traditional coloring ones, they obey the same rule: no pairs of adjacent vertices(edges) are colored with the same color. In this paper, we study a novel kind of edge coloring problem. The problem is also about edge coloring, but to use as many colors as possible to satisfy the constraint: for every vertex in the graph, all the edges incident to it are colored with no more than $q(q \in \mathbb{Z},\ q \geq 2)$ colors. Usually, q is equal to 2. We call it "maximum edge coloring problem". It can be defined formally as follows:

Maximum edge coloring problem: Given a connected undirected simple graph $G=(V,E)$ and a positive integer $q \geq 2$, color all the edges in E, with the constraint: for every vertex in V, all the edges incident to it are colored with no more than q colors, ask for a solution which uses maximum number of colors.

We want to emphasize two points in the definition. Firstly, the input graph is restricted to be connected in the definition. In fact, general graphs can also be solved if we can find a solution to any connected graph. Secondly, q is required to be no less than 2. Since if $q=1$, it is easy to see that all the edges must be colored with only one color because of the connectivity of the graph. So it is a trivial case.

1.2 Motivation

In 2005, Ashish Raniwala and Tzi-cker Chiueh [8] proposed a multi-channel wireless mesh network architecture (called *Hyacinth*) that equips each mesh network node with multiple 802.11 network interface cards (NICs). They point out that intelligent channel assignment is critical to *Hyacinth*'s performance. A series of experiments are carried and the results show that with 2 NICs on each node, it is possible to improve the network throughput by a factor of 6 to 7 when compared with the conventional single-channel ad hoc network architecture. For more details, readers are referred to [8,9]. Actually, in such kind of channel assignment problem in multi-channel wireless mesh networks, a novel computational problem is involved. The wireless mesh network can be modeled as a network graph $G_N=(V_N,E_N,q,m)$, where V_N is the set of mesh routers in the mesh network, E_N is the set of pairs of mesh routers which can communicate directly, q denotes the number of network interface cards each node owns, m is the number of non-overlapping channels provided by the network. Clearly, G_N is a connected, undirected, simple graph and the number of channels assigned to each node can't exceed the number of its NICs: q. If we omit m and take $G_N=(V_N,E_N,q)$ as the input of the above maximum edge coloring problem, then the solution provides an upper bound of the number of channels the mesh network can use. This bound is an important parameter for wireless mesh network researchers.

Because the mesh routers in a wireless mesh network often have two network interface cards, the case $q = 2$ is very important. We design approximation algorithms for this special case and case $q > 2$ is also considered. For more knowledge on approximation algorithms, readers are referred to [10].

The rest of the paper is organized as follows. In section 2, two important properties on the problem are introduced. In section 3 and section 4, we discusses the approximation algorithms and the corresponding approximation ratios for case $q = 2$ and case $q > 2$, respectively. In section 5, we conclude the paper and propose future research direction.

2 Preliminaries

$ALG(G)$ is used to denote the number of colors used in the solution given by our algorithm on the input graph G; $OPT(G)$ to denote the number of colors used in an optimal coloring solution of G.

Lemma 1: *Given an arbitrary connected graph $G = (V, E)$, suppose the optimal solutions use m colors: $1, 2, ..., m$. Based on the color of each edge, we can divide the edge set into m subsets: $E_1, E_2, ..., E_m$. Each subset E_i denotes the set of edges colored with color i. If we choose one edge from each subset, the subgraph H induced by these m edges satisfies:*

1) $\Delta(H) \leq q$;
2) If $q = 2$, then H consists of paths and cycles;
3) If $q = 2$, $OPT(G) \leq n$, where $n = |V(G)|$.

Proof: 1) Because the colors of the edges in H are different from each other, noting the q-constraint for optimal solutions, the degree of each vertex v in H satisfies $1 \leq d_H(v) \leq q$, that means $\Delta(H) \leq q$.
2) If $q = 2$, it is clear that H is a set of paths and cycles, i.e. each connected component of H is either a path or a cycle.
3) According to 2), $OPT(G) = m = |E(H)| \leq |V(H)| \leq n$.

Lemma 2: *Given a vertex cover V^* of a graph G with $|V^*| = k$, let H be the subgraph induced by V^* in G. Then:*

1) $OPT(G) \leq kq$;
2) If H has a matching of size m, then $OPT(G) \leq kq - m$;
3) If $q = 2$ and H is connected, then $OPT(G) \leq k + 1$;
4) If $q = 2$ and H has l connected components $(1 \leq l \leq k)$, then $OPT(G) \leq k+l$.

Proof: 1) Since V^* is a vertex cover, every edge of G is incident to a vertex in V^* at least. On the other hand, the edges incident to V^* can be colored with $|V^*|q$ colors at most based on the q-constraint. Thus, $OPT(G) \leq |V^*|q = kq$.
2) Let M_H be a matching in H of size m. $E(G)$ can be divided into two non-intersecting parts: M_H and $E(G) - M_H$. It is clear that $E(G) - M_H$ can be colored with $(k - 2m)q + 2m(q - 1) = kq - 2m$ new colors at most, no matter

how M_H is colored. On the other hand, M_H can be colored with m colors at most. So, $OPT(G) \leq (kq - 2m) + m = kq - m$.
3) Suppose H is colored with x colors. According to Lemma 1, $1 \leq x \leq k$. If H is connected, then there are at least $x - 1$ vertices in V^* incident to two edges colored with different colors in H. Thus, $E(G) - E(H)$ can be colored with $[k - (x-1)](q-1) + (x-1)(q-2) = k - x + 1$ new colors at most. It yields that $OPT(G) \leq (k - x + 1) + x = k + 1$.
4) Denote by $c_1, c_2, ..., c_l$ $(\sum_{i=1}^{l} c_i = k)$ the numbers of the vertices in the l connected components of H respectively. According to 3), $OPT(G) \leq \sum_{i=1}^{l}(c_i (q-1) + 1) = k(q-1) + l = k + l$.

3 Approximation Algorithms for Case $q = 2$

Clearly, if $\Delta(G) \leq q$, the number of colors used by an optimal solution is equal to $|E|$. According to this fact, we adopt the greedy strategy to design the following approximation algorithm. The main idea is that, first, we find a maximal subgraph H of G with $\Delta(H) \leq q - 1$, then we assign a new color to each edge of H. When dealing with the rest edges, we must be more careful, because this procedure may lead to conflict, which means the constraint is broken. To avoid the conflict, we employ a simple trick as follows: delete the edges of H from the original graph G and just let every non-isolated vertex connected component of the residual graph G' share one new color.

1. Compute a maximal matching M of G;
2. Assign a new color to each edge in M;
3. Delete the edges in M from the original graph G and for each connected component of the residual graph G' which is not an isolated vertex, assign to it a new color;
4. Output each edge with the color assigned to it.

Fig. 1. Algorithm 1

Obviously, the solution given by Algorithm 1 is feasible, because it satisfies the q-constraint. A maximal matching can be found in $O(|E|)$ time. Thus, the time complexity of Algorithm 1 is $O(|E|)$. Now we consider the approximation ratio of the algorithm.

Theorem 1: *For any connected graph G, Algorithm 1 achieves an approximation factor of 3.*

Proof: As we all know, the vertex set V^* matched by the maximal matching M is a vertex cover of G. Clearly, $|V^*| = 2|M|$ and M is also a matching of the subgraph induced by V^*. Base on Lemma 2, we can easily draw the conclusion:

$OPT(G) \leq 2|M|\ \ 2-|M| = 3|M|$. On the other hand, $ALG(G) \geq |M|+1$ (Here, we assume that the residual graph $G^{'} = G - M$ has at least one edge. Because if $G^{'}$ has no edge, $M = G$. Thus $\Delta(G) < 2$. This case is trivial: $ALG(G) = OPT(G) = |E|$, Theorem 1 follows immediately.). So the approximation ratio is:

$$\frac{OPT(G)}{ALG(G)} \leq \frac{3|M|}{|M|+1} \leq 3 \tag{1}$$

The following graph gives a tight example for Algorithm 1.

Example 1: In the graph shown in Figure 2, the set of vertical edges is a maximal matching of G; on the other hand, G can be colored with $3m$ kinds of colors at most. So, $ALG(G) = m + 2$, and $OPT(G) = 3m$.

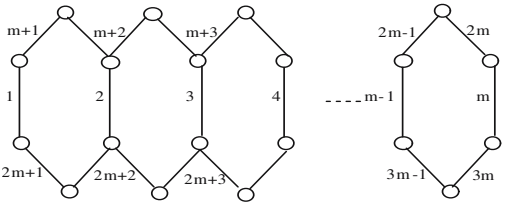


Fig. 2. Tight example for Algorithm 1

Intuitively, if we find a maximum matching rather than finding a maximal matching in step 1, the algorithm would be better. Can this change bring the improvement of the approximation ratio? It can easily be seen that, if the graph is chosen to be a bipartite, the modified algorithm is a factor 2 approximation algorithm for it. Because in bipartite graphs, there exists $max_{matching\ M}|M| = min_{vertex\ cover\ U}|U|$. Combined with Lemma 2, we have

$$\frac{OPT(G)}{ALG(G)} \leq \frac{2|U_{min}|}{|M_{max}|} \leq 2 \tag{2}$$

In fact, for general graphs, we can modify Algorithm 1 as above to get an improved algorithm with ratio 2.5.

Algorithm 2 is shown in Figure 3. A maximum matching can be found in $O(|V|^{\frac{1}{2}}|E|)$ time ([11]). So, the time complexity of Algorithm 2 is $O(|V|^{\frac{1}{2}}|E|)$. Next, we begin to consider the approximation ratio of Algorithm 2. Based on Algorithm 2, the edge set of an input graph G can be divided into three parts. E_1: the maximum matching M; E_2: the set of edges between the unsaturated vertices of M and the saturated vertices; E_3: the set of rest edges. Denote by $n(E_1)$, $n(E_2)$ and $n(E_3)$ the maximum number of different colors held by E_1, E_2 and E_3 respectively. Then $OPT(G)$ is bounded by the sum of $n(E_1)$, $n(E_2)$ and $n(E_3)$. Obviously, this upper bound is too big. To tighten the upper bound, we need to consider how many kinds of new colors can be held in E_3 at most after E_1 and E_2 have been colored. This is the main idea in the proof of Theorem 2.

1. Compute a maximum matching M of G;
2. Assign a new color to each edge in M;
3. Delete the edges of M from the original graph G and for each connected component of the residual graph $G^{'}$ which is not an isolated vertex, assign to it a new color;
4. Output each edge with the color assigned to it.

Fig. 3. Algorithm 2

Because $n(E_2)$ is not easy to estimate, a little trick is employed to divide $E(G)$ in a similar but different way to obtain an upper bound of $OPT(G)$ easily.

Theorem 2: *For any connected graph G, Algorithm 2 achieves an approximation factor of 2.5.*

Proof: Given a graph G, let M be a maximum matching of G and $G^{'} = G - M$. We will take two steps to prove the theorem.

1. Divide the edges in M and the unsaturated vertices into three parts so that we can choose a set of saturated vertices to bound the number of different colors held in the set of edges between unsaturated vertices and saturated vertices;
2. Divide the edge set of G into three subsets to evaluate an upper bound of $OPT(G)$.

Step 1: Obviously, there is no M-augmenting path in G. Based on this important property, we discuss the topology of G in depth. Clearly, there is no edge among the unsaturated vertices of M, and all the unsaturated vertices are adjacent to some of the saturated vertices. For an edge e in M, it is impossible for both of its end points adjacent to different unsaturated vertices (otherwise, there will appear an M-augmenting path in G). This leads to the following three possibilities:

1) neither of the end points of e is adjacent to any unsaturated vertex;
2) only one of the end points of e is adjacent to some unsaturated vertices;
3) both of the end points of e are adjacent to the same unsaturated vertex.

So, we can classify the set of unsaturated vertices into two classes. Class 1: the subset of the unsaturated vertices which are adjacent to both end points of one edge in M at least; Class 2: the left subset of unsaturated vertices which are adjacent to one of the end points of e at most for any given edge e in M. First, we consider the unsaturated vertices in Class 1. There are three kinds of such vertices:

1) those vertices of degree 2;
2) those vertices of degree > 2 and only adjacent to both end points of one edge in M;
3) the left vertices of degree > 2 and adjacent to both end points of at least two edges in M. (See Figure 4).

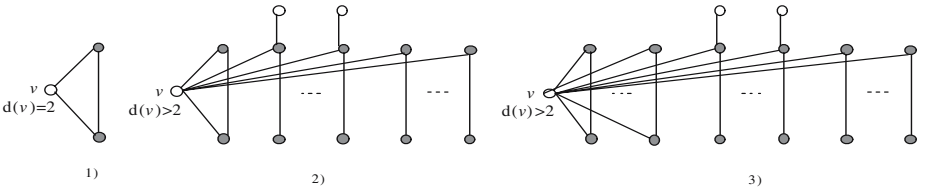


Fig. 4. The three kinds of unsaturated vertices in Class 1. (The filled vertices are saturated ones, and the empty ones correspond to unsaturated vertices.)

We consider three kinds of widgets corresponding to the three kinds of unsaturated vertices mentioned above.

1) widget A: the unsaturated vertex v of degree 2 and the edge e in M whose both end points are adjacent to it;
2) widget B: the unsaturated vertex v of degree > 2 and the only edge e in M whose both end points are adjacent to it;
3) widget C: the unsaturated vertex v of degree > 2 and all the edges (at least two) whose both end points are adjacent to it. (See Figure 5).

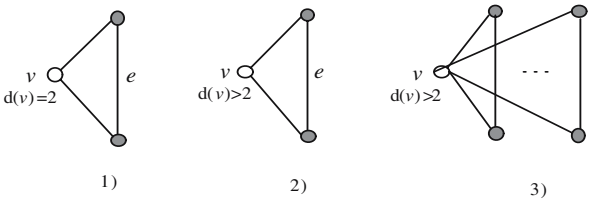


Fig. 5. 1) widget A; 2) widget B; 3) widget C

Widget A and B look very similar, however, they are different. We will show the difference soon. Because of the connectivity of the original graph G, widget A must connect to another edge in M. Clearly v is a two-degree vertex, so it is one of the end points of e in the triangle which connects to another saturated vertex v_1. Obviously, there is an edge e_1 in M incident to v_1. And e_1 satisfies: 1) both of its end points can't be adjacent to an unsaturated vertex at the same time; 2) if one of its end points adjacent to an unsaturated vertex, then the end point must be v_1. For widget B, because the degree of v is more than 2, it must connect to another saturated vertex v_2, and there is an edge e_2 in M incident to v_2. Similarly, e_2 satisfies: the other end point of it, which is not v_2, can't be adjacent to another unsaturated vertex. (See Figure 6).

Now we divide G into three parts as follows: (See Figure 7).

1. Extract all the three kinds of widgets from the original graph G to get three collections of widget A, B and C: $S_A = \{w_{A_1}, ..., w_{A_i}\}$ $(i \geq 0)$, $S_B = \{w_{B_1}, ..., w_{B_j}\}$ $(j \geq 0)$ and $S_C = \{w_{C_1}, ..., w_{C_k}\}$ $(k \geq 0)$.

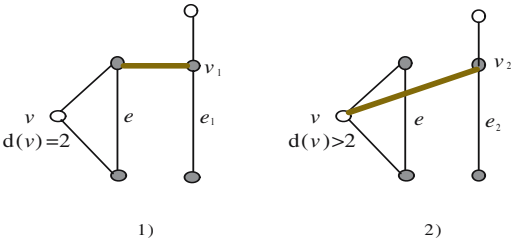


Fig. 6. The bold edge is the connecting edge

2. $Part_1 = S_C$;
   while $(S_B \neq \emptyset)$
   {
   1) Take an element w_B from S_B. Then the unsaturated vertex v in w_B is at least adjacent to one saturated vertex v_2 and v_2 is incident to an edge e_2 in M.
   2) Scan the left elements in S_B. Once the unsaturated vertex of such an element is adjacent to v_2, then take it out of S_B.
   3) Scan the elements in S_A. If one of the two saturated vertices in $w_A \in S_A$ is adjacent to v_2, then take w_A out of S_A.
   4) Put all the widgets taken from S_B and S_A, the connecting edges and e_2 into $Part_1$.
   5) Delete all the elements chosen from S_B and S_A in above steps.
   }
   while $(S_A \neq \emptyset)$
   {
   1) Take an element w_A from S_A. Then one of the saturated vertices in w_A is adjacent to one saturated vertex v_1 and v_1 is incident to an edge e_1 in M.
   2) Scan the elements in S_A. If one of the two saturated vertices in $w_A^{'} \in S_A$ is adjacent to v_1, then take $w_A^{'}$ out of S_A.
   3) Put all the widgets taken from S_A, the connecting edges and e_1 into $Part_1$.
   4) Delete all the elements chosen from S_A in above steps.
   }

3. $Part_2 = \{$The left edges in M having only one of the end points adjacent to unsaturated vertices.$\}$

4. $Part_3 = \{$The left edges in M having no end point adjacent to unsaturated vertices.$\}$

For an edge in M whose both end points are adjacent to the same unsaturated vertex, if one of its end points is incident to a connecting edge, then select the end point as a rectangle vertex, the other as a triangle vertex; otherwise, arbitrarily select one of its end points as a rectangle vertex, the other as a triangle

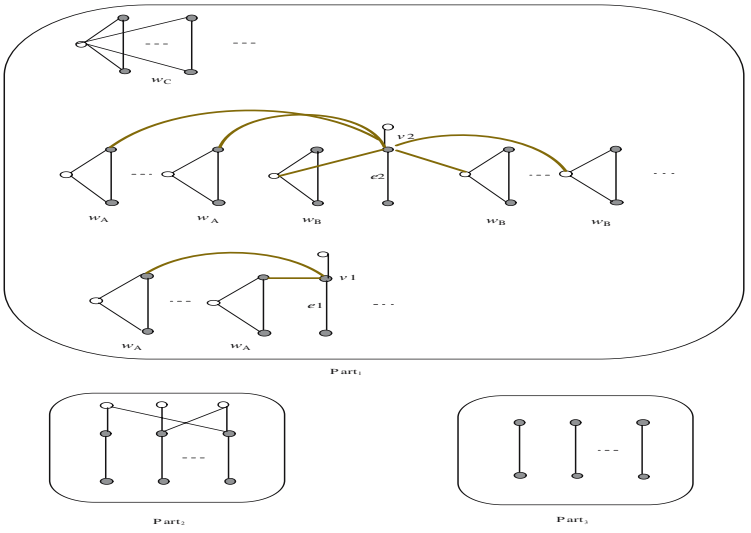


Fig. 7. The topology of graph G (I)

vertex. For an edge in M with only one of its end points adjacent to unsaturated vertices, select the one adjacent to unsaturated vertices as a rectangle vertex, the other as a triangle vertex. For the connecting vertex $v_1(v_2)$, no matter it is adjacent to an unsaturated vertex, take it as a rectangle vertex, and the other end point of $e_1(e_2)$ as a triangle vertex. For an edge in widgets A,B and C in $Part_1$ which connects an unsaturated vertex to one end point of an edge in M, if it is incident to a rectangle vertex, then keep it solid; if it is incident to a triangle vertex, then change it to a dashed line. (See Figure 8).

Step 2: The edge set of G, $E(G)$, can be divided into three non-intersecting parts. E_1: the edges incident to rectangle vertices and the dashed edges; E_2: the edges in M in $Part_3$; E_3: the edges among "the triangle vertices and the vertices in $Part_3$" except the edges in E_2. Obviously, $E(G) = E_1 \cup E_2 \cup E_3, E_1 \cap E_2 = \emptyset, E_1 \cap E_3 = \emptyset, E_2 \cap E_3 = \emptyset$. It is clear that the number of rectangle vertices is equal to the number of edges in M in $Part_1$ and $Part_2$. We use x to denote the number of rectangle vertices. Then the number of edges in M in $Part_3$ is $|M| - x$. Clearly, E_1 can be colored with $2x$ kinds of colors at most, E_2 can be colored with $|M| - x$ kinds of colors at most. Noting that: the triangle vertices and the vertices in $Part_3$ are all saturated vertices. So these vertices have only one degree "free" for new colors no matter how E_1 and E_2 are colored. As a consequence, E_3 can be colored with $\lfloor \frac{[x+2(|M|-x)]}{2} \rfloor$ at most. Thus, we have:

$$OPT(G) \leq 2x + (|M| - x) + \lfloor \frac{[x + 2(|M| - x)]}{2} \rfloor \leq 2|M| + \frac{x}{2} \leq \frac{5|M|}{2} \qquad (3)$$

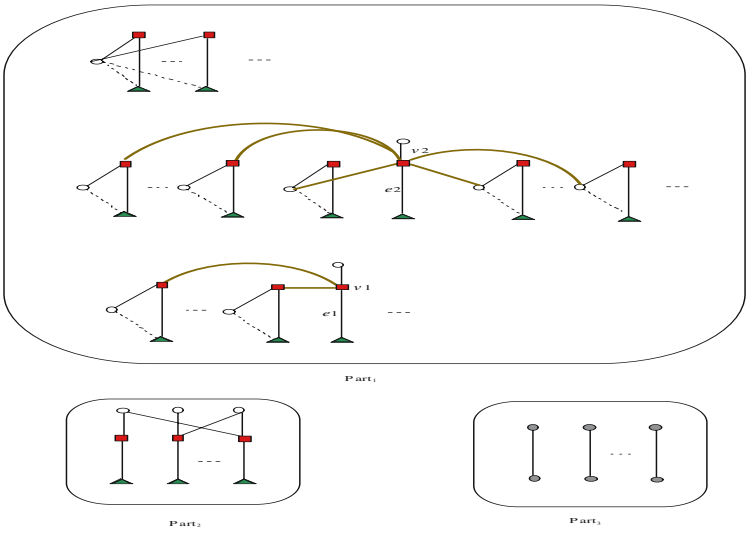


Fig. 8. The topology of graph G (II)

It yields the approximation ratio:

$$\frac{OPT(G)}{ALG(G)} \leq \frac{\frac{5|M|}{2}}{|M|+1} \leq 2.5 \tag{4}$$

4 Approximation Algorithms for Case $q > 2$

Based on the same idea, we can modify Algorithm 1, and get Algorithm 3 shown in Figure 9 for case $q > 2$.

1. Using a greedy method to get a maximal subgraph H with $\Delta(H) \leq q-1$;
2. Assign a new color to each edge of H;
3. Delete the edges of H from the original graph G and for each connected component of the residual graph $G^{'}$ which is not an isolated vertex, assign to it a new color;
4. Output each edge with the color assigned to it.

Fig. 9. Algorithm 3

The first two steps of Algorithm 3 are used to color some edges of G employing a greedy strategy. The greedy procedure of step 1 stops when all the edges of the residual graph $G^{'}$ are at least incident to one vertex v in H with $d_H(v) = q-1$.

Clearly, a maximal subgraph H with $\Delta(H) \leq q-1$ can be found in time $O(|E|)$. The time complexity of Algorithm 3 is $O(|E|)$.

Theorem 3: *For any connected graph G, Algorithm 3 achieves an approximation factor of* $2 + \frac{2}{q-1}$.

Proof: Without loss of generality, we suppose the vertex set of the subgraph H obtained in step 1: $V(H) = V_1 \cup V_2$, where V_1 is the set of vertices with degree $d_H(v) = q-1$, V_2 is the set of vertices with degree $d_H(v) < q-1$. Assume $|V_1| = s$. Then the sum of the degrees of vertices in H is $s(q-1) + \sum_{v \in V_2} d_H(v)$. So H has $\frac{1}{2}[s(q-1) + \sum_{v \in V_2} d_H(v)]$ edges. Thus we get a lower bound of the number of colors used in the solution given by Algorithm 3: $ALG(G) \geq \frac{1}{2}[s(q-1) + \sum_{v \in V_2} d_H(v)] + 1$. (As usual, we assume that the residual graph $G' = G - E(H)$ has at least one edge.)

Now, we begin to estimate an upper bound of the number of colors used by an optimal solution: $OPT(G)$. Clearly, for any edge in G', it is incident to one vertex in V_1 at least. Otherwise, it will contradict the fact that H is a maximal subgraph with $\Delta(H) \leq q-1$. Now we can classify the edges of G into two class: 1) Those incident to a vertex in V_1; 2) Those whose two endpoints are both in V_2. For the first class, they are colored with sq kinds of colors at most. For the second class, they are colored with $\lfloor \frac{1}{2} \sum_{v \in V_2} d_H(v) \rfloor$ kinds of colors at most. So, $OPT(G) \leq sq + \lfloor \frac{1}{2} \sum_{v \in V_2} d_H(v) \rfloor$. The approximation ratio:

$$\frac{OPT(G)}{ALG(G)} \leq \frac{sq + \lfloor \frac{1}{2} \sum_{v \in V_2} d_H(v) \rfloor}{\frac{1}{2}[s(q-1) + \sum_{v \in V_2} d_H(v)] + 1} \leq 2 + \frac{2}{q-1} \tag{5}$$

By a similar idea from the case $q = 2$, we can also find a maximum subgraph whose degree $\leq q-1$ instead of finding a maximal subgraph whose degree $\leq q-1$, which can be used to design a new algorithm with a better approximation ratio. The new algorithm is in Figure 10. To show it is a polynomial time algorithm, the classical maximum b-matching problem should be introduced.

Maximum b-matching problem: Given an undirected graph $G = (V, E)$ and a function b: $V \to \mathbb{Z}^+$ specifying an upper bound for each vertex, the maximum b-matching problem asks for a maximum cardinality set $M \subseteq E$ such that $\forall\ v \in V,\ deg_M(v) \leq b(v)$.

Gabow [12] designed an algorithm of complexity $O(|V||E|log|V|)$ for the maximum b-matching problem in 1983. As a consequence, the time complexity of Algorithm 4 is $O(|V||E|log|V|)$. Now we are ready to give the approximation ratio.

Theorem 4: *For any connected graph G, Algorithm 4 achieves an approximation factor of*

$$1 + \frac{4q-2}{3q^2 - 5q + 2} \tag{6}$$

Proof: (see full paper.)

1. Find a maximum subgraph H_{q-1} with $\Delta(H_{q-1}) \leq q-1$;
2. Assign a new color to each edge of H_{q-1};
3. Delete the edges of H_{q-1} from the original graph G and for each connected component of the residual graph $G^{'}$ which is not an isolated vertex, assign to it a new color;
4. Output each edge with the color assigned to it.

Fig. 10. Algorithm 4

Table 1 compares the approximation ratios of Theorem 3 and Theorem 4. It is clear that Algorithm 4 has made progress compared with Algorithm 3. For instance, if $q = 3$, the approximation ratio of Algorithm 3 is 3, much larger than 1.71, that of Algorithm 4. As q increases, the first ratio approaches 2, while the second one approaches 1. This means Algorithm 4 is almost a precise one in some sense.

Table 1. Comparison of the approximation ratios of Theorem 3, 4

q	3	4	5	6	7	...	∞
$2+\frac{2}{q-1}$	3	$2\frac{2}{3} \approx 2.67$	2.5	2.4	$2\frac{1}{3} \approx 2.33$	...	2
$1+\frac{4q-2}{3q^2-5q+2}$	$1\frac{5}{7} \approx 1.71$	$1\frac{7}{15} \approx 1.47$	$1\frac{9}{26} \approx 1.35$	1.275	$1\frac{13}{57} \approx 1.22$	...	1

5 Conclusion

This paper studies a novel maximum edge coloring problem which arises from the field of wireless mesh networks. We have designed approximation algorithms for cases $q = 2$ and $q > 2$ with approximation ratio 2.5 and $1+\frac{4q-2}{3q^2-5q+2}$ respectively. However, we don't know the complexity of the problem. The corresponding decision problem can be defined as: **Maximum-edge-coloring**$= \{(G, q, k)|(G, q)$ has a k-color solution$\}$. Obviously, it belongs to NP class. We conjecture it is NP-complete.

References

1. Brooks, R.L. On colouring the nodes of a network. *Proc. Cambridge Phil. Soc.* 37, 1941, pp 194-197
2. Karp, R.M. Reducibility among combinatorial problems. In: *Complexity of computer computations* (Eds. R.E.Miller and J.W.Thatcher.) Plenum Press, New York, 1972, pp 85-103
3. Garey, M.R. and Johnson, D.S. The complexity of near optimal graph coloring. *J. ACM* 23, 1976, pp 43-49

4. Turner, J.S. Almost all k-colorable graphs are easy to color. *J. Algor.* 9, 1988, pp 63-82
5. Vizing V.G. On an estimate of the chromatic class of a p-graph. (in Russian) *Diskret. Analiz.* 3, 1964, pp 25-30
6. Holyer, I.J. The NP-completeness of edge-coloring. *SIAM J. Comp.* 10, 1981, pp 718-720
7. Uriel Feige, Eran Ofek and Udi Wieder: Approximating maximum edge coloring in multigraphs. *APPROX* 2002, pp 108-121
8. Ashish Raniwala, Tzi-cker Chiueh: Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. *INFOCOM* 2005, pp 2223-2234
9. Ashish Raniwala, Kartik Gopalan, Tzi-cker Chiueh: Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *Mobile Computing and Communications Review* 8(2), 2004: pp 50-65
10. Vijay V. Vazirani. Approximation algorithms. Springer-Verlag, Berlin, 2001
11. S. Micali, Vijay V. Vazirani. An $O(|V|^{\frac{1}{2}}|E|)$ algorithm for finding maximum matching in general graphs. *Proc.* 21st *IEEE FOCS*, 1980, pp 17–27
12. H.N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. *Proc.* 15th *ACM STOC*, 1983, pp 448-456

Two Improved Range-Efficient Algorithms for F_0 Estimation*

He Sun[1,2] and Chung Keung Poon[1]

[1] Department of Computer Science, City University of Hong Kong
Hong Kong, China
[2] Department of Computer Science and Engineering, Fudan University
Shanghai, China

Abstract. We present two new algorithms for range-efficient F_0 estimating problem and improve the previously best known result, proposed by Pavan and Tirthapura in [15]. Furthermore, these algorithms presented in our paper also improve the previously best known result for Max-Dominance Norm Problem.

1 Introduction

Problem. Let $S = r_1, \cdots, r_n$ be a sequence of intervals where each interval $r_i = [x_i, y_i] \subseteq [1, U]$ is an interval of integers between x_i and y_i. Let $m_j = ||\{i|j \in r_i\}||$ denote the number of intervals in the sequence S that contains j. Then the *kth-frequency moment* of S is defined as $F_k = \sum_{i=1}^{U} m_i^k$. In practice, the *zeroth-frequency moment* of S is the number of distinct elements in $\cup_{i=1}^{n} r_i$.

In this paper, we consider the problem of estimating F_0 in the above data stream model. Let $\varepsilon, \delta > 0$ be two constants. An algorithm $\mathscr{A}$ is said to (ε, δ)-approximate F_0 if the output Z of the algorithm $\mathscr{A}$ satisfies $\Pr[|F_0 - Z| > \varepsilon F_0] < \delta$. The time and space complexity of algorithm $\mathscr{A}$ are functions of the domain size U, the approximation parameter ε and the confidence parameter δ. In practice, the number of intervals n and the size of the universe U are very large. So we seek for algorithms that run quickly using relatively small space. In particular, the time for processing each interval should be sublinear with the length of the interval. We call such algorithms *range-efficient*.

Motivation. The cardinality of a database or data stream is of great importance in itself. In databases, some operations (such as query optimization) require knowledge of the *cardinality*—the number of distinct items—of a specific column in a database. Since commercial databases are usually very large, we can afford to scan each item only once and use limited space to give a desired approximation of F_0. Another application arises from routing of Internet traffic. In this scenario, the router usually has very limited memory and needs to gather

* The work described in this paper was fully supported by a grant from CityU (SRG 7001969).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 659–669, 2007.
© Springer-Verlag Berlin Heidelberg 2007

various statistical properties of the traffic flow. For instance, the number of distinct destination IP addresses in a specific period is a critical property for the router to analyze the behavior of the Internet users. This motivates the *single item case* of the problem, i.e., the estimation of F_0 in a data stream model where each item of the input is a single integer (instead of intervals).

Bar-Yossef et al. [2] formalize the concept of reductions between algorithms for data streams and motivate the concept of list-efficient streaming algorithms which includes range-efficient F_0 estimation as a special case. Through reductions, a range-efficient F_0 algorithm can solve the problem of estimating the number of distinct triangles in graphs. Pavan and Tirthapura [15] also pointed out the relationship between range-efficient F_k estimation algorithm and the Max-Dominance Norm Problem. Though there are other algorithms for the problem that rely on stable distributions and Nisan's pseudorandom generators [6], the solution based on the problem that we focus on is more elegant and has smaller running time.

Related works. In the past twenty years, most research focuses on the single item case. Flajolet and Martin [9] gave the first algorithm for estimating F_0 for this case. The drawback of their algorithm is that they require a perfect hash function to make the input data uniform and independent. In 1999, Alon et al. [1] gave several algorithms for estimating $F_k, k \geq 0$, and used pairwise independent hash functions to get a constant factor approximation F_0 algorithm with space complexity $O(\log U)$. In 2002, Bar-Yossef et al. [2] gave the first algorithm for estimating the number of distinct elements in a data stream that approximates with arbitrarily small relative error. Since then, several approximation schemes have been proposed such as the `Loglog Counting` algorithm [8,11], algorithm using stable distributions [5], and algorithms based on sampling technique [3].

Unfortunately, applying these algorithms on our problem results in an update time proportional to the product between the length of the interval and the running time for updating one item. To overcome this drawback, Bar-Yossef et al. [2] designed two range-efficient approximation algorithms for F_0 and F_2 estimation, which are, to our best knowledge, the first efficient approximation scheme for this kind of problems. In 2005, Pavan and Tirthapura [15] improved the F_0 estimation algorithm of Bar-Yossef et al. and reduced the processing time per item from $O(\frac{1}{\varepsilon^5}\log^5 U \log\frac{1}{\delta})$ to $O(\log\frac{U}{\varepsilon}\log\frac{1}{\delta})$. However the worst case update time per element could be as much as $O(\frac{\log^2 U}{\varepsilon^2}\log\frac{1}{\delta})$.

Results. (1). We give two algorithms with different amortized running time and worst case running time updating each interval for approximating F_0, whose space complexity is the same as the algorithm in [15]. The following table summarizes our results and gives the comparison between our results and the previously best known algorithm proposed in [15]. The $\tilde{O}$ notation suppresses $\log\log U$ factors. (2). We improve the previously best known result for Max-Dominance Norm Problem and reduce the worst case update time from $\tilde{O}(\frac{1}{\varepsilon^2}\log a_{i,j}\log\frac{1}{\delta})$ to $\tilde{O}((\frac{1}{\varepsilon^2}+\log a_{i,j})\log\frac{1}{\delta})$.

2 Preliminaries

Hash functions. A k-universal family of hash functions is a set $\mathscr{H}$ of functions $A \mapsto B$ such that for all distinct $x_1, \cdots, x_k \in A$ and all (not necessarily distinct) $b_1, \cdots, b_k \in B$

$$\Pr_{h \in \mathscr{H}}[h(x_1) = b_1 \wedge \cdots \wedge h(x_k) = b_k] = |B|^{-k}$$

Carter and Wegman's original definition [4] is different from the above, which is what they call *strongly k-universal* hash functions [16].

Here we describe a hash function used in our improved algorithm. First choose a prime number p between U^2 and U^3, and pick a from the set $\{1, \cdots, p-1\}$ and b from $\{0, \cdots, p-1\}$ randomly. Let $h(x) = (a \cdot x + b) \bmod p$. It is well known that $h(x)$ is a pairwise independent hash function. Let $\rho(x)$ be the number of consecutive 0's from the rightmost in x's binary expression. For instance, $\rho(2) = 1$ and $\rho(7) = 0$. In addition, define $\rho(0) = \lceil \log p \rceil$. The following lemma gives the pairwise independence of the hash function $\rho(h(x))$.

Lemma 1. *The random variables* $\{\rho((ax + b) \bmod p) | a \in \{1, \cdots, p-1\}, b \in \{0, \cdots, p-1\}\}$ *are pairwise independent.*

Proof. Since $h(x) = (ax + b) \bmod p$ is a pairwise independent hash function, for any $x \neq y$ and $\alpha, \beta \in \{0, \cdots, p-1\}$, there holds

$$\Pr_{a,b}[h(x) = \alpha \wedge h(y) = \beta] = \Pr_{a,b}[h(x) = \alpha] \cdot \Pr_{a,b}[h(y) = \beta] = \frac{1}{p^2}$$

For all $x \neq y \in \{0, \cdots, p-1\}$ and $i, j \in \{0, \cdots, \lceil \log p \rceil\}$,

$$\begin{aligned}
&\Pr_{a,b}[\rho(h(x)) = i \wedge \rho(h(y)) = j] \\
&= \sum_{\alpha=0}^{p-1} \sum_{\beta=0}^{p-1} \Pr_{a,b}\left[h(x) = \alpha \wedge h(y) = \beta\right] \cdot \Pr\left[\rho(h(x)) = i \wedge \rho(h(y)) = j \,\middle|\, h(x) = \alpha \wedge h(y) = \beta\right] \\
&= \sum_{\alpha=0}^{p-1} \sum_{\beta=0}^{p-1} \Pr_{a,b}\left[h(x) = \alpha \wedge h(y) = \beta\right] \cdot \Pr\left[\rho(\alpha) = i \wedge \rho(\beta) = j\right] \\
&= \frac{1}{p^2} \sum_{\alpha=0}^{p-1} \sum_{\beta=0}^{p-1} \Pr[\rho(\alpha) = i] \cdot \Pr[\rho(\beta) = j] \\
&= \frac{1}{p^2} \frac{p}{2^{i+1}} \frac{p}{2^{j+1}} \\
&= \Pr[\rho(h(x)) = i] \cdot \Pr[\rho(h(y)) = j]
\end{aligned} \tag{1}$$

In summary, the random variables $\{\rho((ax + b) \bmod p) | a \in \{1, \cdots, p-1\}, b \in \{0, \cdots, p-1\}\}$ are pairwise independent. □

Table 1. Comparison of time complexity for range-efficient F_0 estimating algorithm

Algorithm	Worst case update time	Amortized update time
Algorithm in [15]	$O(\frac{1}{\varepsilon^2}\log^2 U \log\frac{1}{\delta})$	$O(\log\frac{U}{\varepsilon}\log\frac{1}{\delta})$
Our algorithm	$\tilde{O}((\frac{1}{\varepsilon^2}+\log U)\log\frac{1}{\delta})$	$\tilde{O}(\log\frac{U}{\varepsilon}\log\frac{1}{\delta})$
Our algorithm[revised]	$O(\log\frac{1}{\varepsilon}\log U\log\frac{1}{\delta})$	$O(\log\frac{1}{\varepsilon}\log U\log\frac{1}{\delta})$

3 Algorithm for Range-Efficient F_0 Estimation

We first give the high level overview of our approach. Like [3,15], our algorithm maintains a current sampling level ℓ. Initially, $\ell = 0$. We use a set S, whose size is $\alpha = \Theta(\frac{1}{\varepsilon^2})$, to store the sampled intervals. When a new interval r comes, the algorithm checks whether or not r intersects with any existing interval r' in S. If there exists such interval r', let $r \leftarrow r' \cup r$. Then we calculate $M(r)$ and $G(r)$, where $M(r) := \max_{x\in r}\rho(h(x))$ and $G(r) := ||\{x \in r|\rho(h(x)) = M(r)\}||$. In other words, $M(r)$ is the highest level achieved by the elements in the interval r and $G(r)$ is the number of elements attaining this level.

If $M(r) \geq \ell$, then we put the interval r into S. When the number of intervals in S exceeds α, the level ℓ increases and the algorithm deletes the intervals whose $M(\cdot)$-value is less than ℓ. Finally, the estimated value of F_0 is

$$Z = \Big(\sum_{i=\ell}^{\lfloor\log p\rfloor} X_i \cdot 2^{i+1}\Big)\cdot 2^{\ell} \tag{2}$$

where

$$X_i = \sum_{r\in S\wedge M(r)=i} G(r) \tag{3}$$

Calculating $M(r)$ and $G(r)$. For the given interval $r = [x, y]$ and hash function $h(x) = (a \cdot x + b) \bmod p$, where p is a prime number, we design an efficient algorithm to calculate $M(r)$ and $G(r)$.

For this problem, a naive solution to get $M(r)$ is to calculate $\rho(h(z))$ for each $z \in [x, y]$, and get the maximum value of them. The time complexity is $O(y - x + 1)$, which could be as much as $\Theta(U)$. In this paper, we reduce the processing time per interval to $O(\log U \log\log U)$.

Consider the following problem: Given the sequence $u, (u+d) \bmod p, \cdots, (u+t\cdot d) \bmod p$, we want to find the maximum integer i, $i \in \{0, \cdots, \lceil\log p\rceil\}$, such that there exists an integer $x \in \{0, \cdots, t\}$ satisfying the following equation

$$(u + x \cdot d) \bmod p \equiv 0 \ (\bmod\ 2^i) \tag{4}$$

It is obvious to see the equivalence of calculating $M(\cdot)$ and the above problem by putting $u = h(x)$, $d = a$ and $t = y - x$.

For any fixed i, Equation (4) is equivalent to $u+x\cdot d \equiv v\cdot 2^i \ (\bmod p)$, for some $v \in \{0, \cdots, p-1\}$. Therefore $d\cdot x \equiv v\cdot 2^i - u \ (\bmod p)$. Since p is a prime number, $(d, p) = 1$ and the solution of the congruence equation $d\cdot x \equiv v\cdot 2^i - u \ (\bmod p)$ is

$$\begin{aligned} x &\equiv d^{\phi(p)-1} \cdot (v \cdot 2^i - u) \bmod p \\ &= d^{-1} \cdot (v \cdot 2^i - u) \bmod p \end{aligned} \tag{5}$$

where $\phi(\cdot)$ is the Euler function.

We can express x as

$$x \equiv (-u \cdot d^{-1} + v \cdot 2^i \cdot d^{-1}) \bmod p \tag{6}$$

Thus we can use the procedure `Hits`, described in [15], to determine the size of the intersection of the set $\{0, \cdots, t\}$ and the sequence

$$u' \bmod p, (u' + d') \bmod p, \cdots, (u' + (p-1) \cdot d') \bmod p$$

where $u' = -u \cdot d^{-1}$ and $d' = 2^i \cdot d^{-1}$.

We now describe our algorithm, `MG`, for computing $M(\cdot)$ and $G(\cdot)$. For the given hash function $h(\cdot)$, integers d, p and interval $r = [x, y]$, set $u' \leftarrow -h(x) \cdot d^{-1}(\bmod p)$ first. Then the algorithm uses the binary search to determine the maximum $i \in \{0, \cdots, \lceil \log p \rceil\}$ such that v:=`Hits`$(p, 2^i \cdot d^{-1}, u', p-1, [0, y-x]) > 0$. Finally, the algorithm outputs i and v as the value of $M(r)$ and $G(r)$.

The formal description of the procedure `Hits`, which calculates the size of the intersection between a given interval and an arithmetic progression over $\mathbb{Z}_p$, can be found in [15].

Theorem 1. *The time complexity of algorithm* `MG` *is* $O(\log U \log\log U)$ *and the space complexity is* $O(\log U)$.

Proof. Since the maximum value of i is $\lceil \log p \rceil$ and we use binary search to determine the required i, we call the procedure `Hits` at most $O(\log\log U)$ times to get the maximum i. By the analysis of [15], the time complexity of `Hits` is $O(\log U)$, and the space complexity is $O(\log U)$. Therefore, the time complexity of the algorithm `MG` is $O(\log U \log\log U)$, and the required space is $O(\log U)$. □

Algorithm and complexity analysis. In the initialization step, the algorithm picks a prime number p between U^2 and U^3, and chooses two numbers a from $\{1, \cdots, p-1\}$ and b from $\{0, \cdots, p-1\}$ at random. Let ℓ be the current level the algorithm stays in and $\ell \leftarrow 0$ initially. In addition, let the sample set S be empty and $\alpha \leftarrow \frac{c}{\varepsilon^2}$ where c is a constant determined by the following analysis. We store an interval r in S as a triple (r, d, w) where $d = M(r)$ and $w = G(r)$.

When a new interval $r_i = [x_i, y_i]$ arrives, the algorithm executes the following operations:

1. If $\exists (r, d, w) \in S$ such that $r_i \cap r \neq \emptyset$:
   (a) While $\exists (r, d, w) \in S$ such that $r \cap r_i \neq \emptyset$
   $S \leftarrow S - \{(r, d, w)\}$, $r_i \leftarrow r \cup r_i$, $X_d \leftarrow X_d - w \cdot 2^{d+1}$,
   $Z \leftarrow Z - w \cdot 2^{d+1}$.
   (b) $d_i \leftarrow M(r_i)$, $w_i \leftarrow G(r_i)$.
   (c) $S \leftarrow S \cup \{(r_i, d_i, w_i)\}$, $X_{d_i} \leftarrow X_{d_i} + w_i \cdot 2^{d_i+1}$, $Z \leftarrow Z + w_i \cdot 2^{d_i+1}$.

2. Else If $M(r_i) \geq \ell$ then
   (a) $d_i \leftarrow M(r_i)$, $w_i \leftarrow G(r_i)$.
   (b) $S \leftarrow S \cup \{(r_i, d_i, w_i)\}$, $X_{d_i} \leftarrow X_{d_i} + w_i \cdot 2^{d_i+1}$, $Z \leftarrow Z + w_i \cdot 2^{d_i+1}$.
   (c) If $||S|| > \alpha$ then
      i. $Z \leftarrow Z - X_\ell$; $S \leftarrow \{(r, d, w) | d > \ell\}$; $\ell \leftarrow \min_{(r,d,w)\in S} d$.
      ii. If $\ell > \lfloor \log p \rfloor$ then return;

When an estimate for F_0 is asked for, the algorithm returns $Z \cdot 2^\ell$.

To boost up the probability of achieving the desired approximation value, we run in parallel $O(\log \frac{1}{\delta})$ copies of the algorithm above and take the median of the resulting approximations as the final estimated value.

Theorem 2. *The space complexity of the algorithm above is* $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$.

Proof. The space required by calculating $M(\cdot)$ and $G(\cdot)$ is $O(\log U)$. For estimation algorithm, the sample S consists of $\alpha = \Theta(\frac{1}{\varepsilon^2})$ elements, each of whom needs $O(\log U)$ space. In addition, the algorithm needs $\min\{\frac{c}{\varepsilon^2}, \lfloor \log p \rfloor\} \cdot \log U$ space to store the value of $X_0, \cdots, X_{\lfloor \log p \rfloor}$. Therefore the total space is $O(\frac{1}{\varepsilon^2} \log U)$. Since we execute the algorithm $O(\log \frac{1}{\delta})$ times in parallel, the space complexity of this algorithm is $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$. □

Theorem 3. *The amortized time to process an interval* $r = [x, y]$ *for the algorithm is* $\tilde{O}(\log \frac{U}{\varepsilon} \log \frac{1}{\delta})$*, and the worst case running time to process an interval* $r = [x, y]$ *is* $\tilde{O}((\frac{1}{\varepsilon^2} + \log U) \log \frac{1}{\delta})$.

Proof. The running time to process an interval consists of three parts: 1. Check whether or not there exists an interval $r' \in S$, such that $r \cap r' \neq \emptyset$; 2. Time for calculating $M(r)$ and $G(r)$; 3. Time for handling an overflow in the sample.

We use a balanced binary search tree T to store the elements in S. So we can use $O(\log \frac{1}{\varepsilon})$ time to check if r intersects with any interval in S in the first part. As Theorem 1 mentioned, we need $O(\log U \log \log U)$ time to calculate $M(r)$ and $G(r)$. Now we analyze the running time of the third part. When the size of S exceeds α, the algorithm uses $O(\frac{1}{\varepsilon^2})$ time to calculate the current level $\ell' \leftarrow \min_{(r,d,w)\in S \wedge d > \ell} d$ and discards the intervals whose $M(\cdot)$'s value is less than ℓ'. This step need scan each element $(r, d, w) \in S$ once, which requires $O(\frac{1}{\varepsilon^2})$ time. Therefore the worst case time complexity of the algorithm is $\tilde{O}((\frac{1}{\varepsilon^2} + \log U) \log \frac{1}{\delta})$.

As for the amortized time, we follow the approach of Pavan and Tirthapura and argue that the total time for handling overflow in the sample (i.e., part 3) over the whole data stream is not more than $\tilde{O}(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$ since the maximum number of level changes is $O(\log U)$. Therefore, the amortized time for inserting an interval for this part is $O(1)$ if the number of input intervals in the data stream is large. Consequently, the amortized time is dominated by the time for part 1 and 2 which is $\tilde{O}(\log \frac{U}{\varepsilon} \log \frac{1}{\delta})$ in total. □

Revised algorithm implementation. The algorithm above uses a balanced binary tree to store the intervals in S. In the streaming algorithms, some researchers (such as [2]) use the maximum number of steps the algorithm spent

on a single item as the measure of time complexity. In order to improve the worst case running time for updating per element, we revise our algorithm proposed above. We use a list of balanced binary trees $T_0, T_1, \cdots, T_u$, $u = \lfloor \log p \rfloor$, to store the intervals in the sample S. The number of trees is not more than $\min\{\lfloor \log p \rfloor, \frac{c}{\varepsilon^2}\}$. When we need to store an interval r in S, the algorithm calculates $M(r)$ and $G(r)$ first of all, and stores r in $T_{M(r)}$ if $M(r)$ is not less than the current level ℓ. At the same time, the algorithm updates the estimator Z, and $X_{M(r)}$, whose value is defined by Equation (2) and (3).

Theorem 4. *The space complexity of the revised algorithm is* $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$.

Proof. The space used by the algorithm is the space required for the procedure `MG` plus the space for storing the list of trees $T_0, \ldots, T_u$. By Theorem 1, the space complexity for calculating $M(r)$ and $G(r)$ is $O(\log U)$. For the list of binary trees, we store at most $O(\frac{1}{\varepsilon^2})$ items, each of which consists of an interval $r_i = [x_i, y_i]$. In addition, we need $O(\min\{\lfloor \log p \rfloor, \frac{c}{\varepsilon^2}\} \cdot \log U)$ space to store X_i for each tree T_i and $O(\log U)$ space to store Z. Therefore the total space is $O(\frac{1}{\varepsilon^2} \log U)$. Since we run $O(\log \frac{1}{\delta})$ copies of the algorithm in parallel, the total space required by the algorithm is $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$. □

Theorem 5. *The amortized time to process an interval* $r = [x, y]$ *for the revised algorithm is* $O(\log \frac{1}{\varepsilon} \log U \log \frac{1}{\delta})$*, and the worst case running time to process an interval* $r = [x, y]$ *is* $O(\log \frac{1}{\varepsilon} \log U \log \frac{1}{\delta})$.

Proof. The running time to process an interval r consists of three parts: 1. Check whether or not there exists an interval $r' \in T_j$, $0 \le j \le \lfloor \log p \rfloor$, such that $r \cap r' \neq \emptyset$; 2. Time for calculating $M(r)$ and $G(r)$; 3. Time for handling an overflow in the sample.

For the first part, let n_i denote the number of intervals in T_i. Since all the intervals in each tree are disjoint, we can use a balanced binary search tree to store the intervals. Therefore for each tree T_i, we can use $O(\log n_i)$ time to check if r intersects with any interval in T_i. The total time for this part is not more than

$$\begin{aligned} \sum_{i=0}^{\lfloor \log p \rfloor} \log n_i &= \log \prod_{i=0}^{\lfloor \log p \rfloor} n_i \\ &\le \log \Big(\frac{\alpha}{\log p} \Big)^{\log p} \\ &= O\big(\log p \big(\log \frac{1}{\varepsilon} - \log \log p \big) \big) \\ &= O\big(\log U \log \frac{1}{\varepsilon} - \log U \log \log U \big) \end{aligned} \tag{7}$$

By Theorem 1, the required time for the second part is $O(\log U \log \log U)$. For part 3, when the size of S exceeds α, the algorithm finds the minimum ℓ', $\ell' > \ell$, such that $T_{\ell'}$ is not an empty tree. The algorithm discards tree T_ℓ, and lets $\ell \leftarrow \ell'$. We can use a linked list to store the root of each (non-empty)

tree and the running time for finding ℓ' is $O(1)$. Since the maximum number of level changes is $O(\log U)$, the total time taken by level changes over the whole data stream is not more than $O(\log U \log \frac{1}{\delta})$, and the amortized time updating per element for this part is $O(1)$ if the number of intervals in the data stream is large.

Combined with the three parts, both the amortized and worst case update time to process each interval are $O(\log \frac{1}{\varepsilon} \log U \log \frac{1}{\delta})$. □

Correctness proof. Let the sample $S = \cup_{i=0}^{\lfloor \log p \rfloor} T_i$, where $T_i = \{r | M(r) = i\}$. Let NT_i be the number of distinct elements in set T_i. In addition, let $W(x, i)$ be the indicator random variable whose value is 1 if and only if $\rho(h(x)) = i$.

Define

$$Z_\ell = \frac{Z}{2^\ell} = \sum_{i=\ell}^{\lfloor \log p \rfloor} X_i \cdot 2^{i+1} \tag{8}$$

Lemma 2. $\mathrm{E}[Z_\ell] = F_0 \cdot \frac{1}{2^\ell}, \mathrm{Var}[Z_\ell] = F_0 \frac{1}{2^\ell}(1 - \frac{1}{2^\ell})$.

Proof. Let $D(I)$ denote the set of distinct elements in $I = \{r_1, \cdots, r_n\}$. We want to estimate $F_0 = ||D(I)||$.

Since $\mathrm{E}[W(x, \ell)] = \frac{1}{2^{\ell+1}}$, we get

$$\mathrm{E}[X_i] = \sum_{r \in T_i} \sum_{x \in r} \mathrm{E}[W(x, i)] = \mathrm{E}[NT_i] \cdot \frac{1}{2^{i+1}}$$

Assume that the current level is ℓ, so we get

$$\mathrm{E}[Z_\ell] = \mathrm{E}\Big[\sum_{i=\ell}^{\lfloor \log p \rfloor} X_i \cdot 2^{i+1} \Big] = \sum_{i=\ell}^{\lfloor \log p \rfloor} 2^{i+1} \mathrm{E}[NT_i] \cdot \frac{1}{2^{i+1}} = \sum_{i=\ell}^{\lfloor \log p \rfloor} \mathrm{E}[NT_i] = F_0 \cdot \frac{1}{2^\ell}$$

By Lemma 1, the random variables $\{W(x, i) | x \in D(I)\}$ are all pairwise independent, thus the variance of Z_ℓ is $F_0 \frac{1}{2^\ell}(1 - \frac{1}{2^\ell})$. □

Theorem 6. $\Pr\{Z \in [(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]\} \geq \frac{2}{3}$.

Proof. Let s be the level in which the algorithm stops, and $t^\star$ is the lowest level such that $\mathrm{E}[Z_{t^\star}] < \frac{\alpha}{C}$, where C is the constant number determined by the following analysis. Let the size of the sample S be $\alpha = \frac{c}{\varepsilon^2}$. Then the probability that the algorithm fails to give a desired estimation is

$$\begin{aligned}
\Pr\{|Z - F_0| > \varepsilon F_0\} &= \Pr\Big\{\Big|\frac{Z}{2^s} - \frac{F_0}{2^s}\Big| > \varepsilon \frac{F_0}{2^s}\Big\} \\
&= \sum_{i=0}^{\lfloor \log p \rfloor} \Pr\Big\{\Big|Z_i - \frac{F_0}{2^i}\Big| > \varepsilon \frac{F_0}{2^i} \Big| s = i\Big\} \cdot \Pr\{s = i\} \\
&= \sum_{i=0}^{\lfloor \log p \rfloor} \Pr\Big\{\big|Z_i - \mathrm{E}[Z_i]\big| > \varepsilon \mathrm{E}[Z_i] \Big| s = i\Big\} \cdot \Pr\{s = i\} \\
&\leq \sum_{i=0}^{t^\star} \Pr\Big\{\big|Z_i - \mathrm{E}[Z_i]\big| > \varepsilon \mathrm{E}[Z_i]\Big\} + \sum_{i=t^\star+1}^{\lfloor \log p \rfloor} \Pr\{s = i\}
\end{aligned} \tag{9}$$

By Chebyshev's inequality, we know that for all $i \in \{0, \cdots, t^\star\}$, there holds

$$\Pr\left\{\left|Z_i - \mathrm{E}[Z_i]\right| > \varepsilon \mathrm{E}[Z_i]\right\} \leq \frac{\mathrm{Var}[Z_i]}{\varepsilon^2 \mathrm{E}^2[Z_i]}$$

On the other hand, if the algorithm stops in the level $\ell' > t^\star$, it implies that there are at least α disjoint intervals in S in level $t^\star$, each of whom contributes at least one to the corresponding $X_j, t^\star \leq j \leq \lfloor \log p \rfloor$. So we get $Z_{t^\star} \geq \alpha$, and

$$\begin{aligned}
\Pr\left\{|Z - F_0| > \varepsilon F_0\right\} &\leq \sum_{i=0}^{t^\star} \frac{\mathrm{Var}[Z_i]}{\varepsilon^2 \mathrm{E}^2[Z_i]} + \Pr\{Z_{t^\star} \geq \alpha\} \\
&< \sum_{i=0}^{t^\star} \frac{2^i}{\varepsilon^2 F_0} + \Pr\left\{Z_{t^\star} - \mathrm{E}[Z_{t^\star}] \geq \alpha - \frac{\alpha}{C}\right\} \\
&< \frac{1}{\varepsilon^2 F_0} \cdot 2^{t^\star + 1} + \frac{1}{C\alpha(1 - 1/C)^2} \\
&< \frac{4}{\varepsilon^2 \mathrm{E}[Z_{t^\star - 1}]} + \frac{1}{C\alpha(1 - 1/C)^2} \qquad (10) \\
&\leq \frac{4C}{\varepsilon^2 \alpha} + \frac{1}{C\alpha(1 - 1/C)^2} \\
&= \frac{4C}{c} + \frac{\varepsilon^2}{Cc(1 - 1/C)^2} \\
&< \frac{4C}{c} + \frac{1}{Cc(1 - 1/C)^2} \\
&< \frac{1}{3}
\end{aligned}$$

by using $C = 3$ and $c = 50$. □

So the probability can be amplified to $1 - \delta$ by running in parallel $O(\log \frac{1}{\delta})$ copies of the algorithm and outputting the median of the returning $O(\log \frac{1}{\delta})$ approximating values.

4 Extension: Max-Dominance Norm Problem

Let the input consist of k streams of m integers, where each integer $1 \leq a_{i,j} \leq U$, $i = 1, \cdots, k, j = 1, \cdots, m$, represents the jth element of the ith stream. The max-dominance norm is defined as $\sum_{j=1}^{m} \max_{1 \leq i \leq k} a_{i,j}$.

Employing stable distributions, Cormode and Muthukrishnan [6] designed an (ε, δ)-approximation algorithm of this problem. Pavan and Tirthapura showed the relationship between this problem and range-efficient F_0 estimation [15]. In the same paper, they gave an approximation algorithm for Max-Dominance Norm Problem, whose space complexity is $O(\frac{1}{\varepsilon^2}(\log m + \log U) \log \frac{1}{\delta})$, with amortized update time $O(\log \frac{a_{i,j}}{\varepsilon} \log \frac{1}{\delta})$ and worst case update time $\tilde{O}(\frac{1}{\varepsilon^2} \log a_{i,j} \log \frac{1}{\delta})$.

Combining with Pavan and Tirthapura's technique and our algorithm presented in this paper, it is not hard to show the following theorem.

Theorem 7. *There exists an* (ε, δ)*-approximation algorithm for Max-Dominance Norm Problem, whose space complexity is* $O(\frac{1}{\varepsilon^2}(\log m + \log U)\log\frac{1}{\delta})$*, with amortized update time* $\tilde{O}(\log\frac{a_{i,j}}{\varepsilon}\log\frac{1}{\delta})$ *and worst case update time* $\tilde{O}((\frac{1}{\varepsilon^2} + \log a_{i,j})\log\frac{1}{\delta})$.

5 Further Work

We consider a more general range-efficient F_0 estimation problem — range-efficient F_0 estimation under the *turnstile model* [13] where there can be both insertions and deletions of intervals. Let the multiset S be empty initially. When the intervals arrive, we can not only insert some intervals into S but also delete the intervals from S. When an estimate is requested, the algorithm need to give a desired approximation value of $||S||$.

Some algorithms, such as [5,10], focus on single item case and are suitable for this turnstile model. However, all these known algorithms cannot be easily generalized to the range-efficient case for the following reasons: (1) It is proven in [5] that stable distributions with small stability parameter can be used to approximate F_0 norm. The difficulty of generalizing this method to range-efficient case is the lack of general range-summable p-stable random variables. Though strong range-summability results are known for F_1 and F_2, for general $0 < p \leq 2$, there is no any known p-stable range-summable random variable construction algorithm, which was also listed in [7]. (2) Ganguly et al. gave another algorithm to estimate the cardinality of the multiset S [10], but this algorithm required the use of $\Theta(\log\frac{1}{\delta})$-wise independent hash function. Let h be such kind of hash functions. The algorithm presented in [10] need to calculate $\rho(h(x))$. Though there exist some k-wise range-summable hash function construction algorithms for general k, it is not clear how to calculate $||\{x \in r | \rho(h(x)) = t\}||$ effectively, for the given interval r and parameter t. We leave this more general range-efficient F_0 estimation problem for further work.

Acknowledgements. We thank Piotr Indyk, Graham Cormode and Omer Reingold for some helpful discussions.

References

1. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137-147, 1999.
2. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 623-632, 2002.
3. Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1-10, 2002.
4. J. L. Carter, M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143-154, 1979.

5. G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (How to zero in). In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 335-345, 2002.
6. G. Cormode, S. Muthukrishnan. Estimating dominance norms of multiple data streams. In *Proceedings of the 11th European Symposium on Algorithms*, pages 148-160, 2003.
7. G. Cormode. Stable distributions for stream computations: it's as easy as 0,1,2. In *Workshop on Management and Processing of Massive Data Streams*, at FCRC, 2003.
8. M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *Proceedings of the European Symposium on Algorithms*, pages 605-617, 2003.
9. P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182-209, 1985.
10. S. Ganguly, M. Garofalakis, and R. Rastogi. Tracking set-expression cardinalities over continuous update streams. *The International Journal on Very Large Data Bases*, 13:354-369, 2004.
11. F. Giroire. Order statistics and estimating cardinalities of massive data sets. *Discrete Mathematics and Theoretical Computer Science*, Vol. AD, pages 157-166, 2005.
12. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 189-197, 2000.
13. S. Muthukrishnan. Data streams: algorithms and applications. Invited talk at 14th ACM-SIAM Symposium on Discrete Algorithms. Available from `http://athos.rutgers.edu/~muthu/stream-1-1.ps`
14. N. Nisan. Pseudorandom generators for space-bounded computation, In *Proceedings of the 22nd Symposium on Theory of Computation*, pages 204-212, 1990.
15. A. Pavan, S. Tirthapura. Range-efficient computation of F_0 over massive data stream. In *Proceedings of the 21st International Conference on Data Engineering*, pages 32-43, 2005.
16. M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Science*, 22:265-279, 1981.
17. R. Weron. On the Chambers-Mallows-Stuck method for simulating skewed stable random variables. Technical report, Hugo Steinhaus Center for Stochastic Methods, Wrocław, 1996.

Approximation to the Minimum Rooted Star Cover Problem

Wenbo Zhao[1,2,⋆] and Peng Zhang[1,2]

[1] State Key Lab. of Computer Science, Institute of Software,
Chinese Academy of Sciences, P.O.Box 8718, Beijing, 100080, China
[2] Graduate University of Chinese Academy of Sciences, Beijing, China
{zwenbo,zhp}@gcl.iscas.ac.cn

Abstract. In this paper, we study the following minimum rooted star cover problem: given a complete graph $G = (V, E)$ with a length function $l : E \to \mathbb{Z}^+$ that satisfies the triangle inequality, a designated root vertex $r \in V$, and a length bound D, the objective is to find a minimum cardinality set of rooted stars, that covers all vertices in V such that the length of each rooted star is at most D, where a rooted star is a subset of E having a common center $s \in V$ and containing the edge (r, s). This problem is **NP**-complete and we present a constant ratio approximation algorithm for this problem.

Keywords: Minimum Rooted Star Cover, Approximation Algorithm.

1 Introduction

We consider an interesting vehicle routing problem, in which we want to find a minimum size collection of subgraphs that cover the graph. Formally, the problem is as follows. Consider a *metric space*, i.e., a complete graph $G = (V, E)$ with a length function $l : E \to \mathbb{Z}^+$ that satisfies the triangle inequality. A root vertex $r \in V$ is designated in this graph. A *star* S in this graph is a subset of E having a common *center* $s \in V$ and an *r-star* S with center s is a star containing the edge (r, s). Also, we refer to a r-star as a *rooted* star. Here, we say a vertex v is *covered* by a star S if v is in the subgraph induced by S and we use the notation $l(S)$ to denote the length of a star S, i.e., $l(S) = \sum_{e \in S} l(e)$. The input to the *minimum rooted star cover problem* consists of a complete graph $G = (V, E)$ with a length function $l : E \to \mathbb{Z}^+$ that satisfies the triangle inequality, one designated root vertex $r \in V$, and a length bound D. The objective is to find a minimum cardinality set of r-stars (rooted stars), that cover all vertices in V. In addition, all these r-stars are required to have length at most D.

This problem has many applications in power network design and routing, resource allocation, and other related areas. Actually, various vehicle routing

⋆ This work is part of the author's master thesis. The author is grateful to his supervisor Prof. Angsheng Li for advice and encouragement. The author is partially supported by NSFC Grant no. 60325206 and no. 60310213.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 670–679, 2007.
© Springer-Verlag Berlin Heidelberg 2007

problems have been extensively studied in the literatures [3,6,8,9]. Most of these papers focus on developing heuristic solutions or solving these problems optimally, there are only a few results on approximation algorithms of these problems. Even et al. [4] considered the *min-max star cover* problem where it is called *unrooted k-stars cover* problem. They presented a bicriteria (4, 4)-approximation algorithm, i.e., a polynomial-time algorithm that outputs a solution which covers all vertices of the graph with no more than $4k$ (unrooted) stars, and the cost of the solution is no more than four times the cost of an optimal solution which uses no more than k (unrooted) stars. Their method is based on LP-rounding. Arkin et al. [2] considered the *minimum star cover* problem. The stars considered in [2] are also unrooted and they are pairwise disjoint, i.e., no vertex is covered by two different (unrooted) stars. In [2], they presented a $(2\alpha + 1)$-approximation algorithm, where α is the approximation ratio for the *minimum metric k-median* [1]. Arkin et al. [2] also used their algorithm for minimum star cover problem to obtain a bicriteria $(3 + \epsilon, 3 + \epsilon)$-approximation for the unrooted k-stars cover problem. Recently, Nagarajan and Ravi study a minimum vehicle routing problem where their goal is to find a minimum cardinality set of paths that cover all vertices of graph [7]. The path considered in [7] is required to start from a specified root vertex and has length no more than a "common deadline", say, D. They also presented a $O(\log D)$-approximation algorithm for the problem.

Inspired by the work of Nagarajan and Ravi in [7], we consider the minimum rooted star problem. We firstly show that it is **NP**-hard. Also, we present an approximation algorithm for this problem by firstly dividing vertices of graph into $O(\log D)$ levels and then connecting the root to each "small" (unrooted) star on each level. Our approximation algorithm is similar to the one in [7], but it achieves a constant performance ratio, which is $7 \cdot (2\alpha + 1)$.

2 Hardness

In this section, we prove that the minimum rooted star cover problem is **NP**-complete even in the special case that the metric space is induced by a weighted star. Here we propose a reduction from the **NP**-complete problem *k-partition problem* [5]: Given a set $T = \{t_1, t_2, \ldots, t_n\}$ of integers, can T be partitioned into k disjoint sets $T_1, T_2, \ldots, T_k$ such that $\sum_{t_i \in T_1} t_i = \sum_{t_i \in T_2} t_i = \ldots = \sum_{t_i \in T_k} t_i$?

Theorem 1. *The minimum rooted star cover problem is* **NP***-complete.*

Proof. Given an instance to the k-partition problem consisting of a set $T = \{t_1, t_2, \ldots, t_n\}$ of integers, we define a complete graph $G = (V, E)$ with $V = \{r, v_1, v_2, \ldots, v_n\}$ and a length function $l : E \to \mathbb{Z}^+$ as follows:

$$l(u, v) = \begin{cases} t_i & (u, v) = (r, v_i) \\ t_i + t_j & (u, v) = (v_i, v_j) \end{cases}$$

We designate the root vertex as $r \in V$ and the length bound $D = \sum_{t_i \in T} t_i / k$.

The key observation is that setting the center of any r-star S on the vertex r will reduce the length of the star. Thus, we can assume that all r-stars in G have their centers on r. Clearly, T can be partitioned into k sets $T_1, T_2, \ldots, T_k$ such that $\sum_{t_i \in T_1} t_i = \sum_{t_i \in T_2} t_i = \ldots = \sum_{t_i \in T_k} t_i$ if and only if the minimum size of the set of r-stars with length bound D is k, where these r-stars cover all vertices of this graph and their centers are all r. This leads to the conclusion of this theorem. □

3 Minimum Rooted Star Cover

In this section, we present an approximation algorithm achieving a constant guarantee for minimum rooted star cover problem. Our algorithm uses an algorithm for minimum star cover problem in [2]. In order to distinguish it from the minimum rooted star cover problem, we also refer to this problem as *minimum (unrooted) star cover* problem, which is described next.

3.1 Minimum Rooted Star Cover

In the minimum (unrooted) star cover problem, we are given a complete graph $G = (V, E)$ with a length function $l : E \to \mathbb{Z}^+$ that satisfies the triangle inequality, and a length bound D. The goal is to cover all vertices of the graph with the minimum number of (unrooted) stars whose lengths are at most D, moreover, these (unrooted) stars are pairwise disjoint, i.e., no vertex is covered by two different (unrooted) stars. As mentioned in the introduction, this problem along with other variants has been studied recently in [2]. The authors in [2] also presented a $(2\alpha+1)$-approximation algorithm, where α is the approximation ratio for the minimum metric k-median problem [1]. Thus, we have the following theorem.

Theorem 2 (Theorem 10 [2]). *There is a $(2\alpha + 1)$-approximation algorithm for the minimum unrooted star cover problem, where α is the approximation ratio for the minimum metric k-median problem.* □

3.2 Minimum Rooted Star Cover

The basic idea of the algorithm for rooted star cover problem is that, if an r-star visits some points with "large" distances from the root, it resembles an unrooted star (with smaller length) covering just those vertices. More concretely, we divide the vertices of the graph into $\log D$ levels, roughly accordingly to their distances from the root, and solve an unrooted star cover problem on each level (with appropriate length constraint). Given a complete graph $G = (V, E)$ with a length function $l : E \to \mathbb{Z}^+$ that satisfies the triangle inequality, one designated root vertex $r \in V$, and a length bound D, algorithm **MinRSC** works as follows.

Algorithm MinRSC

1. Define levels as follows:

$$L_j = \begin{cases} \{v : D-1 < l(r,v) \le D\} & j = 0 \\ \{v : D-2^j < l(r,v) \le D-2^{j-1}\} & 1 \le j \le \lfloor \log D \rfloor \\ \{v : 0 < l(r,v) \le D-2^{\lfloor \log D \rfloor}\} & j = \lfloor \log D \rfloor + 1 \end{cases}.$$

2. For each vertex on L_0, take a single edge from r to this vertex as an r-star.
3. For $j = 1, 2, \ldots, \lfloor \log D \rfloor + 1$ do
   (a) Run the $(2\alpha+1)$-approximation algorithm for the minimum (unrooted) star cover problem, on the vertex set L_j, with length bound 2^{j-1}. Let the (unrooted) stars obtained be $S_1^j, S_2^j, \ldots, S_{t_j}^j$. Their centers are $s_1^j, s_2^j, \ldots, s_{t_j}^j$, respectively.
   (b) For each of these (unrooted) stars, add the edge (r, s_k^j) to the star $S_k^j, 1 \le k \le t_j$, to obtain an r-star, i.e., $S_k^j = S_2^j \cup \{(r, s_k^j)\}$.

For an r-star S with a center s, we use the notation $\delta(S)$ to denote the set of vertices connected by s (excluding root r), i.e., $\delta(S) = \{v \in V : v$ is connected by s and $v \notin \{r, s\}\}$. If S is a (unrooted) star, then $\delta(S) = \{v \in V : v$ is connected by s and $v \neq s\}$. Here, we should mention that the (unrooted) stars obtained by the $(2\alpha+1)$-approximation algorithm in [1] are pairwise disjoint, i.e., no vertex is covered by two different (unrooted) stars. In the following, we firstly present some interesting properties of any instance of r-star.

Lemma 1. *For any r-star S with length bound D, the star center s and the vertices of the set $\delta(S)$ can only locate on at most 3 different levels.*

Proof. For a proof by contradiction, we assume that the star center s and vertices of set $\delta(S)$ locate on 4 different levels. Without loss of generality, we can also assume that the subscripts of these 4 levels are consecutive, i.e., $L_{i+3}, L_{i+2}, L_{i+1}, L_i$. Let the 3 vertices of $\delta(S)$ other than s on the different levels be v_1, v_2, v_3, respectively. Next, we will distinguish 4 cases according to the location of s.

Case 1: $s \in L_{i+3}, v_1 \in L_{i+2}, v_2 \in L_{i+1}$, and $v_3 \in L_i$. It is easy to verify that $l(r,s)+l(s,v_1) > D-2^{i+2}$. Moreover, it is clear that $l(s,v_2) > 2^{i+1}$ and $l(s,v_3) > 2^{i+1}+2^i$. Thus, we have $l(S) \ge l(r,s)+l(s,v_1)+l(s,v_2)+l(s,v_3) > D+2^i > D$. This leads to a contradiction.

Case 2: $v_1 \in L_{i+3}, s \in L_{i+2}, v_2 \in L_{i+1}$, and $v_3 \in L_i$. Again, it is easy to verify that $l(r,s)+l(s,v_2) > D-2^{i+1}$. Moreover, it is clear that $l(s,v_1)+l(s,v_3) > 2^{i+1}+2^i$. Thus, we have $l(S) \ge l(r,s)+l(s,v_1)+l(s,v_2)+l(s,v_3) > D+2^i > D$, This also leads to a contradiction.

Case 3: $v_1 \in L_{i+3}, v_2 \in L_{i+2}, s \in L_{i+1}$, and $v_3 \in L_i$. It is clear that $l(s,v_1)+l(s,v_3) > 2^{i+1}+2^i$. Similarly, we also have $l(r,s)+l(s,v_2) > D-2^{i+1}$. Thus, we have $l(S) \ge l(r,s)+l(s,v_1)+l(s,v_2)+l(s,v_3) > D+2^i > D$, which implies a contradiction.

Case 4: $v_1 \in L_{i+3}, v_2 \in L_{i+2}, v_3 \in L_{i+1}$, and $s \in L_i$. It is clear that $l(s,v_1) > 2^{i+1}+2^i$ and $l(s,v_2) > 2^i$. Moreover, $l(r,s)+l(s,v_3) > D-2^i$. Thus, we have

$l(S) \geq l(r,s)+l(s,v_1)+l(s,v_2)+l(s,v_3) > D+2^{i+1}+2^i > D$, which also implies a contradiction.

With all of this, the lemma thus follows. □

Given an r-star S with length bound D, we may split the r-star S into several r-stars $S_i, i = 1,2,\ldots$, such that for each resultant r-star S_i, the center of S_i and the vertices of $\delta(S_i)$ are all on the same level, and also, the length of each resultant r-star is no more than D. We will prove the following lemma.

Lemma 2. *For any r-star S with star center s and length bound D, there are t r-stars $S_i, 1 \leq i \leq t \leq 5$, such that they satisfy the following conditions:*

1. *for each $S_i, i = 1,2,\ldots,t$, $l(S_i) \leq D$;*
2. *for each $S_i, i = 1,2,\ldots,t$, the center of S_i, s_i, and the vertices of $\delta(S_i)$ are all on the same level;*
3. *$\{s\} \cup \delta(S) = \bigcup_{i=1}^{t}(\{s_i\} \cup \delta(S_i))$.*

Proof. We will prove this lemma by "splitting" S into at most 5 r-stars. By lemma 1, the center s of S and the vertices of $\delta(S)$ can only locate on at most 3 different levels. Assume that the 3 levels are $L_{i_1}, L_{i_2}, L_{i_3}$ and $s \in L_{i_1}$. Divide the vertices of $\delta(S)$ into 3 sets, i.e., $\delta(S^{i_1}) = \delta(S) \cap L_{i_1}, \delta(S^{i_2}) = \delta(S) \cap L_{i_2}, \delta(S^{i_3}) = \delta(S) \cap L_{i_3}$, where $S^{i_k}, 1 \leq k \leq 3$, is a r-star such that the center of S^{i_k} is s, and $\delta(S^{i_k}) = \delta(S) \cap L_{i_k}$. Since $s \in L_{i_1}$, it is clear that r-star S^{i_1} satisfies conditions 1 and 2.

For S^{i_2}, order the vertices of $\delta(S^{i_2})$, $u_1, u_2, \ldots, u_m$, so that $l(s,u_1) \leq l(s,u_2) \leq \ldots \leq l(s,u_m)$, where $m = |\delta(S^{i_2})|$. Since the length of r-star S is no more than D, we have

$$\sum_{i=1}^{m} 2l(s,u_i) \leq 2D - 2l(r,s). \tag{1}$$

Here, we can also assume that left part of the above equation is more than $D - l(r,s)$, otherwise, we could handle this case more easily.

If there exists some $i^*, 1 \leq i^* \leq m$, such that $\sum_{i=1}^{i^*-2} 2l(s,u_i) + l(s,u_{i^*-1}) \leq D - l(r,s)$, but $\sum_{i=1}^{i^*-1} 2l(s,u_i) > D - l(r,s)$, then by inequality (1) we have $\sum_{i=i^*}^{m-1} 2l(s,u_i) + l(s,u_m) \leq D - l(r,s)$. Thus, by triangle inequality, we have

$$l(r,u_1) + \sum_{i=2}^{i^*-1} l(u_1,u_i) \leq l(r,s) + \sum_{i=1}^{i^*-2} 2l(s,u_i) + l(s,u_{i^*-1}) \leq D. \tag{2}$$

Notice that the left hand side of the above inequality is just the length of the r-star with center u_1, and the vertices (excluding r) covered by this r-star are just $u_1, u_2, \ldots, u_{i^*-1}$. Similarly,

$$l(r,u_{i^*}) + \sum_{i=i^*+1}^{m} l(u_{i^*},u_i) \leq l(r,s) + \sum_{i=i^*}^{m-1} 2l(s,u_i) + l(s,u_m) \leq D. \tag{3}$$

Also, the left hand side of the above inequality is just the length of the r-star with center u_i^*, such that the vertices (excluding r) covered by this r-star are exact $u_{i^*}, u_{i^*+1}, \ldots, u_m$.

Otherwise, there exsits some $i^*, 1 \leq i^* \leq m$, $\sum_{i=1}^{i^*-1} 2l(s,u_i) \leq D - l(r,s)$, but $\sum_{i=1}^{i^*-1} 2l(s,u_i) + l(s,u_{i^*}) > D - l(r,s)$, then by inequality (1) we have $\sum_{i=i^*}^{m-1} 2l(s,u_i) + l(s,u_m) \leq l(s,u_{i^*}) + \sum_{i=i^*+1}^{m} 2l(s,u_i) \leq D - l(r,s)$. By the triangle inequality again, we have

$$l(r,u_1) + \sum_{i=2}^{i^*-1} l(u_1,u_i) \leq l(r,s) + \sum_{i=1}^{i^*-1} 2l(s,u_i) \leq D, \tag{4}$$

and

$$l(r,u_{i^*}) + \sum_{i=i^*+1}^{m} l(u_{i^*},u_i) \leq l(r,s) + \sum_{i=i^*}^{m-1} 2l(s,u_i) + l(s,u_m) \leq D, \tag{5}$$

which implies that we can also "split" r-star S^{i_2} into 2 r-stars.

For the r-star S^{i_3}, by the same "splitting" approach as r-star S^{i_2}, we can "split" it into at most 2 r-stars too. It is clear that the vertices covered by these resultant r-stars are just the vertices covered by r-star S. This leads the conclusion of this lemma. □

Now, we will turn our attention on how to divide a r-star S into at most 3 (unrooted) stars, where the center of S, vertices of $\delta(S)$, and the 3 resultant (unrooted) stars are all on the same level.

Lemma 3. *Given any r-star S with center s and length bound D, and s and the vertices of the set $\delta(S)$ are all on the same level L_j, $1 \leq j \leq \lfloor \log D \rfloor + 1$, there are t (unrooted) stars $S_i, 1 \leq i \leq t \leq 3$, such that they satisfy the following conditions:*

1. *for each $S_i, i = 1, 2, \ldots, t$, $l(S_i) \leq 2^{j-1}$;*
2. *for each $S_i, i = 1, 2, \ldots, t$, the center of S_i, s_i, and the vertices of $\delta(S_i)$ are all in the same level L_j;*
3. *$\{s\} \cup \delta(S) = \bigcup_{i=1}^{t} (\{s_i\} \cup \delta(S_i))$.*

Proof. Order the vertices of $\delta(S)$, $u_1, u_2, \ldots, u_m$ such that $l(s,u_1) \geq l(s,u_2) \geq \ldots \geq l(s,u_m)$, where $m = |\delta(S)|$. Since the center of S and the vertices of the set $\delta(S)$ are all in the same level L_j, we have $\sum_{i=1}^{m} l(s,u_i) \leq 2^j$. Since $l(s,u_i) \leq D, 1 \leq i \leq m$, there is some $k, 1 \leq k \leq m$, such that, $\sum_{i=k+1}^{m} l(s,u_i) \leq 2^{j-1}$ but $\sum_{i=k}^{m} l(s,u_i) > 2^{j-1}$. (If $\sum_{i=1}^{m} l(s,u_i) \leq 2^{j-1}$, one (unrooted) star will be sufficient.) Let S_1 be the (unrooted) star having vertex s as center such that $\delta(S_1) = \{u_{k+1}, u_{k+2}, \ldots, u_m\}$. Notice that

$$l(s,u_1) + \sum_{i=2}^{k-1} 2l(s,u_i) + l(s,u_k) \leq \sum_{i=1}^{k-1} 2l(s,u_i) \leq 2 \cdot 2^{j-1}.$$

Here, we can also assume that the left hand side of the above inequality is more than 2^{j-1}, otherwise, another (unrooted) star will be sufficient.

If there exits some i^*, $1 \le i^* \le k$, such that $l(s,u_1)+\sum_{i=2}^{i^*-1} 2l(s,u_i) \le 2^{j-1} < l(s,u_1)+\sum_{i=2}^{i^*-1} 2l(s,u_i)+l(s,u_{i^*})$, then, by triangle inequality we have

$$\sum_{i=1}^{i^*-2} l(u_{i^*-1},u_i) \le l(s,u_1)+\sum_{i=2}^{i^*-1} 2l(s,u_i) \le 2^{j-1}, \tag{6}$$

and

$$\sum_{i=i^*}^{k-1} l(u_k,u_i) \le l(s,u_{i^*})+\sum_{i=i^*+1}^{k-1} 2l(s,u_i)+l(s,u_k) \le 2^{j-1}. \tag{7}$$

Thus, we can let S_2 be the (unrooted) star having vertex u_{i^*-1} as center such that $\delta(S_2)=\{u_1,\ldots,u_{i^*-2}\}$ and let S_3 be the (unrooted) star having vertex u_k as center such that $\delta(S_2)=\{u_{i^*},\ldots,u_{k-1}\}$.

Otherwise, there exits some i^*, $1 \le i^* \le k$, such that $l(s,u_1)+\sum_{i=2}^{i^*-1} 2l(s,u_i)+l(s,u_{i^*}) \le 2^{j-1} < l(s,u_1)+\sum_{i=2}^{i^*} 2l(s,u_i)$, Again, by triangle, inequality, we have

$$\sum_{i=1}^{i^*-1} l(u_{i^*},u_i) \le l(s,u_1)+\sum_{i=2}^{i^*-1} 2l(s,u_i)+l(s,u_{i^*}) \le 2^{j-1}, \tag{8}$$

and

$$\sum_{i=i^*+1}^{k-1} l(u_k,u_i) \le l(s,u_{i^*+1})+\sum_{i=i^*+2}^{k-1} 2l(s,u_i)+l(s,u_k) \le 2^{j-1}. \tag{9}$$

Similarly, we can let S_2 be the (unrooted) star having vertex u_{i^*} as center and $\delta(S_2)=\{u_1,\ldots,u_{i^*-1}\}$ and let S_3 be the (unrooted) star having vertex u_k as center and $\delta(S_2)=\{u_{i^*+1},\ldots,u_{k-1}\}$.

Clearly, these (unrooted) stars satisfy the 3 conditions required by this lemma, and thus the lemma follows. □

With the previous three lemmas, we prove our main theorem as follows.

Theorem 3. *Algorithm* **MinRSC** *is a* $(2\alpha+1)\beta\gamma$*-approximation algorithm for the minimum rooted star cover problem, where* $\beta = 3$, $\gamma = 5$, *and* α *is the approximation ratio for the minimum metric k-median problem.*

Proof. Define n_j to be the size of the optimal solution to the minimum (unrooted) star cover problem on the subgraph induced by vertex set $L_j, 1 \le j \le \lfloor \log D \rfloor + 1$. Let n'_j be the number of the (unrooted) stars returned by the $(2\alpha+1)$-approximation algorithm on the subgraph induced by vertex set $L_j, 1 \le j \le \lfloor \log D \rfloor + 1$ in step 3(a). Let n_0 be the cardinality of L_0 and n'_0 be the number of r-stars returned in the step 2. Since $\frac{n'_i}{n_i} < 2\alpha+1$ and $n_0 = n'_0$, we have

$$\frac{\sum_{j=0}^{\lfloor \log D \rfloor+1} n'_i}{\sum_{j=0}^{\lfloor \log D \rfloor+1} n_i} \le 2\alpha+1.$$

Suppose OPT, a set of r-stars, is one of the optimal solution to the minimum (unrooted) star cover problem. By lemmas 1, 2, and 3, for any r-star $S \in OPT$, we can "split" it into at most $\beta\gamma$ (unrooted) stars such that the vertices of each (unrooted) star are all in the same level respectively, and all these (unrooted) stars cover all the vertices which are covered by S. Moreover, the length of each (unrooted) star in the level L_j is no more than 2^{j-1}. Notice that the set of the resultant (unrooted) stars in the same level also give a feasible solution to the minimum (unrooted) star cover problem in that level. Thus, we have

$$\beta\gamma|OPT| \geq \sum_{j=0}^{\lfloor \log D \rfloor + 1} n_i,$$

which implies that

$$\sum_{j=0}^{\lfloor \log D \rfloor + 1} n'_i \leq (2\alpha + 1)\beta\gamma|OPT|.$$

This completes the proof of theorem 3. □

3.3 Better Approximation

In the previous subsection, we present a general framework to show that our approximation algorithm achieves a constant performance ratio. The basic idea of our framework is to split an r-star into several (unrooted) stars (with appropriate length bound) such that they can cover all vertices of the r-star, and also the vertices of each (unrooted) star are all on the same level. In the following, we will improve the constant guarantee by splitting the r-star into smaller number of (unrooted) stars. For that, we need the following two lemmas.

Lemma 4. *For any r-star S with star center s and the length bound D, if all vertices of the set $\delta(S)$ are on the same level L_j, and $\sum_{v\in\delta(S)} l(s,v) \leq 2^{j-1}, 1 \leq j \leq \lfloor \log D \rfloor + 1$, there are t (unrooted) stars $S_i, 1 \leq i \leq t \leq 2$, such that*

1. *for each $S_i, 1 \leq i \leq t$, the center of S_i, s_i, and the vertices of $\delta(S_i)$ are all on the level L_j,and moreover $l(S_i) \leq 2^{j-1}$;*
2. *$\delta(S) = \bigcup_{i=1}^{t}(\{s_i\} \cup \delta(S_i))$.*

□

The proof of the above lemma is quit similar to that of lemma 2, we omit the details here.

Lemma 5. *For any r-star S with star center s and the length bound D, there are t (unrooted) stars $S_i, 1 \leq i \leq t \leq 7$, such that*

1. *for each $S_i, 1 \leq i \leq t$, the center of S_i, s_i, and the vertices of $\delta(S_i)$ are all in the same level L_{i_j},and moreover $l(S_i) \leq 2^{i_j-1}$;*
2. *$\{s\} \cup \delta(S) = \bigcup_{i=1}^{t}(\{s_i\} \cup \delta(S_i))$.*

Proof. If the center s and the vertices of $\delta(S)$ are all on the same level, by lemma 3, it is easy to cover these vertices by at most 3 (unrooted) stars which satisfy the 2 conditions of the lemma.

If the center s and the vertices of $\delta(S)$ are in 2 different levels. Without loss of generality, we can assume that subscripts of the 2 different levels are consecutive, i.e., L_{i+1}, L_i. There are 2 cases according the location of s.

Case 1: $s \in L_{i+1}$. Clearly, s and the vertices of $\delta(S) \cap L_{i+1}$ can be covered by a single (unrooted) star whose length is at most 2^i. Let S' be the r-star such that the center of S' is s and $\delta(S') = \delta(S) \cap L_i$. By the similar "splitting" approach of lemma 2, we can "split" S' into at most 2 r-stars, S'_1, S'_2. Then by lemma 3, we can "split" the 2 r-stars S'_1 and S'_2 into at most 6 (unrooted) stars. Thus, we can cover the s and the vertices of $\delta(S)$ by at most 7 (unrooted) stars, which satisfy the 2 conditions of this lemma.

Case 2: $s \in L_i$. Clearly, by lemma 3, vertex s and the vertices of $\delta(S) \cap L_i$ can be covered by at most 3 (unrooted) stars such that the length of each (unrooted) star is at most 2^{i-1}. For the vertices of $\delta(S) \cap L_{i+1}$, by lemma 4, they can be covered by at most 2 (unrooted) stars. It is easy to verify that the 5 stars satisfy the conditions of this lemma.

If the center s and the vertices of $\delta(S)$ are in 3 different levels, Without loss of generality, we can assume that their subscripts levels are consecutive, i.e., L_{i+2}, L_{i+1}, L_i. There are 3 cases according to the location of s.

Case 3: $s \in L_{i+2}$. It is easy to verify that there is only one vertex of $\delta(S)$ in level L_i. Thus, we can make it as a (unrooted) star, and we have $\sum_{v\delta(S)\cap L_{i+1}} l(s,v) \leq 2^i$. By lemma 4, the vertices of $\delta(S) \cap L_{i+1}$ can be covered by at most 2 (unrooted) stars. For the vertex s and the vertices of $\delta(S) \cap L_{i+2}$, they can be covered by a (unrooted) star such that its center is s and its length is no more than 2^{i+1}. Clearly, the 4 (unrooted) stars satisfy the conditions of this lemma.

Case 4: $s \in L_{i+1}$. Also, it is easy to verify that there is only one vertex of $\delta(S)$ in level L_i. For the vertex s and vertices of $\delta(S) \cap L_{i+1}$, they can be covered by a (unrooted) star such that its center is s and its length is no more that 2^i. For the vertices of $\delta(S) \cap L_{i+2}$, by lemma 4, they can be covered by 2 (unrooted) stars. Clearly, the 4 (unrooted) stars here satisfy the conditions of this lemma.

Case 5: $s \in L_i$. Assume that one of the vertices of $\delta(S)$ in level L_{i+2} is v_1. Since $l(S) > l(r,s) + l(s,v_1) > D - 2^i + 2^i$, it leads to a contradiction.

All of this, the lemma follows. □

By lemma 5 and we have the following theorem directly.

Theorem 4. *Algorithm* **MinRSC** *is a* $(2\alpha+1)\tau$*-approximation algorithm for the minimum rooted star cover problem, where* α *is the approximation ratio for the minimum metric k-median problem and* $\tau = 7$. □

4 Discussion

In this paper, we show that the minimum rooted star cover problem is **NP**-complete. We present an approximation algorithm for this problem and also we show a general framework to prove that our algorithm is a constant ratio approximation algorithm. Then, by a more careful analysis, we obtain a better performance guarantee. However, the bottleneck for improving approximation ratio is in the case 1 of lemma 5. It is interesting to ask whether we can do something more in this case.

Acknowledgement

We would like to express our sincere thanks to anonymous referees for their comments that improved our paper. Kindly do needful for its publication.

References

1. Arya, V., Garg N., Khandekar, R., Pandit, V., Meyerson, A., Nunagata, K.: Local Search Heuristics for k-median and Facility Location Problems. In the proceedings of STOC 2001, 21–29.
2. Arkin, E. M., Hassin, R., Levin, A.: Approximation for Minimum and Min-Max Vehicle Routing Problems. Journal of Algorithms **59** (2006), 1–18.
3. Desrochers, M., Desrosiers, J., Solomon, M.: A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. Operation Research **40** (1992): 342–354.
4. Even, G., Garg N., Konemann, J., Ravi, R., Sinha, A.: Covering Graph Using Trees and Stars. In the proceedings of APPROX 2003, 24–25.
5. Garey, M. R., Johnson, D. S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco, 1979.
6. Kohen, A., Kan, A. R., Trienekens, H.: Vehicle Routing with Time Windows. Operations Research **36** (1987): 266–273.
7. Nagarajan, V., Ravi, R.: Minimum vehicle routing with a common deadline. In the proceedings of APPROX 2006.
8. Savelsbergh, M.: Local Search for Routing Problems with Time Windows. Annals of Operations Research, **4** (1985): 285–305.
9. Tan, K. C., Lee, L. H., Zhu, K. Q., Ou, K.: Heuristic Methods for Vehicle Routing Problems with Time Windows. Artificial Intelligence in Engineering 2001: 281–295.

Approximability and Parameterized Complexity of Consecutive Ones Submatrix Problems

Michael Dom, Jiong Guo*, and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{dom,guo,niedermr}@minet.uni-jena.de

Abstract. We develop a refinement of a forbidden submatrix characterization of 0/1-matrices fulfilling the Consecutive Ones Property (C1P). This novel characterization finds applications in new polynomial-time approximation algorithms and fixed-parameter tractability results for the problem to find a maximum-size submatrix of a 0/1-matrix such that the submatrix has the C1P. Moreover, we achieve a problem kernelization based on simple data reduction rules and provide several search tree algorithms. Finally, we derive inapproximability results.

1 Introduction

A 0/1-matrix has the *Consecutive Ones Property (C1P)* if there is a *permutation* of its columns, that is, a finite series of column swappings, that places the 1's consecutive in every row[1]. The C1P of matrices has a long history and it plays an important role in applications from computational biology and combinatorial optimization. It is well-known that it can be decided in linear time whether a given 0/1-matrix has the C1P, and, if so, also a corresponding permutation can be found in linear time [1,6,10,11,14]. Moreover, McConnell [13] recently described a certifying algorithm for the C1P.

Often one will find that a given matrix M does not have the C1P. Hence, it is a natural and practically important problem to find a submatrix of M with maximum size that has the C1P [5,8,16]. Unfortunately, even for sparse matrices with few 1-entries this quickly turns into an NP-hard problem [8,16]. In this paper, we further explore the algorithmic complexity of this problem, providing new polynomial-time approximability and inapproximability and parameterized complexity results. To this end, our main technical result is a structural theorem, dealing with the selection of particularly useful forbidden submatrices. Before we describe our results concerning algorithms and complexity in more detail, we introduce some notation.

We call a matrix that results from deleting some rows and columns from a given matrix M a *submatrix* of M. Then the decision version of the problem we study here is defined as follows:

* Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

[1] It can be defined symmetrically for columns; we focus on rows here.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 680–691, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Consecutive Ones Submatrix (COS)
Input: An $m \times n$ 0/1-matrix M and a nonnegative integer $n' \leq n$.
Question: Is there an $m \times n'$ submatrix of M that has the C1P?

We study two optimization versions of the decision problem: The minimization version of COS, denoted by Min-COS, asks for a minimum-size set of columns whose removal transforms M into a matrix having the C1P. The maximization version of COS, denoted by Max-COS, asks for the maximum number n' such that there is an $m \times n'$ submatrix of M having the C1P.

Whereas an $m \times n$ matrix is a matrix having m rows and n columns, the term (x, y)-*matrix* will be used to denote a matrix that has at most x 1's in a column and at most y 1's in a row. (This notation was used in [8,16].) With $x = *$ or $y = *$, we indicate that there is no upper bound on the number of 1's in columns or in rows.

Hajiaghayi [7] observed that in Garey and Johnson's monograph [5] the reference for the NP-hardness proof of COS is not correct. Then, COS has been shown NP-hard for $(2, 4)$-matrices by Hajiaghayi and Ganjali [8]. Tan and Zhang showed that for $(2, 3)$- or $(3, 2)$-matrices this problem remains NP-hard [16]. COS is trivially solvable in $O(m \cdot n)$ time for $(2, 2)$-matrices.

Tan and Zhang [16] provided polynomial-time approximability results for the sparsest NP-hard cases of Max-COS, that is, for $(2, 3)$- and $(3, 2)$-matrices: Restricted to $(3, 2)$-matrices, Max-COS can be approximated within a factor of 0.5. For $(2, *)$-matrices, it is approximable within a factor of 0.5; for $(2, 3)$-matrices, the approximation factor is 0.8.

Let d denote the number of columns we delete from the matrix M to get a submatrix M' having the C1P. Besides briefly indicating the computational hardness (approximation and parameterized) of Max-COS and Min-COS on $(*, 2)$-matrices, we show the following main results.

1. For $(*, \Delta)$-matrices with any constant $\Delta \geq 2$, Min-COS is approximable within a factor of $(\Delta + 2)$, and it is fixed-parameter tractable with respect to d. In particular, this implies a polynomial-time factor-4 approximation algorithm for Min-COS for $(*, 2)$-matrices. Factor-4 seems to be the best factor one can currently hope for, because a factor-δ approximation for Min-COS restricted to $(*, 2)$-matrices will imply a factor-$\delta/2$ approximation for Vertex Cover. It is commonly conjectured that Vertex Cover is not polynomial-time approximable within a factor of $2 - \epsilon$, for any constant $\epsilon > 0$, unless P=NP [12].
2. For $(*, 2)$-matrices, Min-COS admits a data reduction to a problem kernel consisting of $O(d^2)$ columns and rows.
3. For Min-COS on $(2, *)$-matrices, we give a factor-6 polynomial-time approximation algorithm and a fixed-parameter algorithm with a running time of $O(6^d \cdot \min\{m^4 n, mn^6\})$.

Due to the lack of space, several proofs are deferred to the full version of the paper.

2 Preliminaries

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [3,4,15]. One dimension is the input size n (as in classical complexity theory) and the other one the *parameter* d (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(d) \cdot n^{O(1)}$ time, where f is a computable function only depending on d. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction rules*, often yielding a *reduction to a problem kernel (kernelization)*. Here the goal is, given any problem instance x with parameter d, to transform it into a new instance x' with parameter d' such that the size of x' is bounded from above by some function only depending on d, the instance (x, d) has a solution iff (x', d') has a solution, and $d' \leq d$. A mathematical framework to show *fixed-parameter intractability* was developed by Downey and Fellows [3] who introduced the concept of *parameterized reductions*. A parameterized reduction from a parameterized language L to another parameterized language L' is a function that, given an instance (x, d), computes in time $f(d) \cdot n^{O(1)}$ an instance (x', d') (with d' only depending on d) such that $(x, d) \in L \Leftrightarrow (x', d') \in L'$. The basic complexity class for fixed-parameter intractability is W[1] as there is good reason to believe that W[1]-hard problems are not fixed-parameter tractable [3,4,15].

We only consider 0/1-matrices $M = (m_{i,j})$, that is, matrices containing only 0's and 1's. We use the term *line* of a matrix M to denote a row or column of M. A column of M that contains only 0-entries is called a 0-*column*. Two matrices M and M' are called *isomorphic* if M' is a permutation of the rows and columns of M. *Complementing* a line ℓ of a matrix means that all 1-entries of ℓ are replaced by 0's and all 0-entries are replaced by 1's.

Let $M = (m_{i,j})$ be a matrix. Let r_i denote the i-th row and let c_j the j-th column of M, and let M' be the submatrix of M that results from deleting all rows except for $r_{i_1}, \ldots, r_{i_p}$ and all columns except for $c_{j_1}, \ldots, c_{j_q}$ from M. Then M'*contains an entry* $m_{i,j}$ of M, denoted by $m_{i,j} \in M'$, if $i \in \{i_1, \ldots, i_p\}$ and $j \in \{j_1, \ldots, j_q\}$. A row r_i of M *belongs* to M', denoted by $r_i \in M'$, if $i \in \{i_1, \ldots, i_p\}$. Analogously, a column c_j of M belongs to M' if $j \in \{j_1, \ldots, j_q\}$. A matrix M is said to *contain a matrix* M' if M' is isomorphic to a submatrix of M.

3 Hardness Results

As observed by Tan and Zhang [16], Max-COS with each row containing at most two 1's is equivalent to the Maximum Induced Disjoint Paths Subgraph problem (Max-IDPS), where, given an undirected graph $G = (V, E)$, we ask for a maximum-size set $W \subseteq V$ of vertices such that the subgraph of G induced by W, denoted by $G[W]$, is a set of vertex-disjoint paths. Since we may assume w.l.o.g. that the input matrix M has no two identical rows and no row of M contains only one 1, a 0/1-matrix M where each row contains exactly two 1's

can be interpreted as a graph $G_M = (V, E)$ with V corresponding to the set of columns and E corresponding to the set of rows. It is easy to verify that M has the C1P iff G_M is a union of vertex-disjoint paths.

We show the hardness of Max-IDPS by giving an approximation-preserving reduction from the NP-hard INDEPENDENT SET problem to Max-IDPS[2].

Theorem 1. *There exists no polynomial-time factor-$O(|V|^{(1-\epsilon)})$ approximation algorithm, for any $\epsilon > 0$, for* MAXIMUM INDUCED DISJOINT PATHS SUBGRAPH *(Max-IDPS) unless NP-complete problems have randomized polynomial-time algorithms.*

With the equivalence between MAX-COS, restricted to $(*, 2)$-matrices, and Max-IDPS and results from [2,3,9], we get the following corollary.

Corollary 1. *Even in case of $(*, 2)$-matrices,*

1. *there exists no polynomial-time factor-$O(|V|^{(1-\epsilon)})$ approximation algorithm, for any $\epsilon > 0$, for* MAX-COS *unless NP-complete problems have randomized polynomial-time algorithms,*
2. MAX-COS *is W[1]-hard with respect to the number of the columns in the resulting consecutive ones submatrix, and*
3. *assuming P≠NP,* MIN-COS *cannot be approximated within a factor of* 2.7212.

4 $(*, \Delta)$-Matrices

This section will present two positive results for the minimization version of CONSECUTIVE ONES SUBMATRIX (MIN-COS) restricted to $(*, \Delta)$-matrices, one concerning the approximability of the problem, the other one concerning its fixed-parameter tractability. To this end, we develop a refinement of the forbidden submatrix characterization of the C1P by Tucker [18], which may also be of independent interest.

Definitions and Observations. Every 0/1-matrix $M = (m_{i,j})$ can be interpreted as the adjacency matrix of a bipartite graph G_M: For every line of M there is a vertex in G_M, and for every 1-entry $m_{i,j}$ in M there is an edge in G_M connecting the vertices corresponding to the i-th row and the j-th column of M. In the following definitions, we call G_M the *representing graph* of M; all terms are defined in analogy to the corresponding terms in graph theory.

Let M be a matrix and G_M its representing graph. Two lines ℓ, ℓ' of M are *connected in* M if there is a path in G_M connecting the vertices corresponding to ℓ and ℓ'. A submatrix M' of M is called *connected* if each pair of lines belonging to M' is connected in M'. A maximal connected submatrix of M is called a *component* of M. A *shortest path* between two connected submatrices M_1, M_2 of M is the shortest sequence $\ell_1, \ldots, \ell_p$ of lines such that $\ell_1 \in M_1$ and $\ell_p \in M_2$

[2] A different reduction from INDEPENDENT SET to Max-IDPS was independently achieved by Tan and Zhang [16].

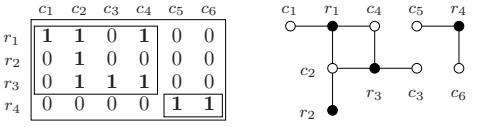


Fig. 1. A matrix with two components and its representing bipartite graph

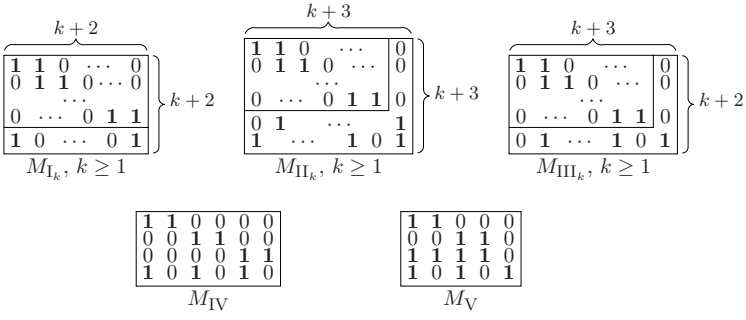


Fig. 2. The forbidden submatrices due to Tucker mentioned in Theorem 2

and the vertices corresponding to $\ell_1, \ldots, \ell_p$ form a path in G_M. If such a shortest path exists, the value $p-1$ is called the *distance* between M_1 and M_2.

Note that each submatrix M' of M corresponds to an induced subgraph of G_M and that each component of M corresponds to a connected component of G_M. An illustration of the components of a matrix is shown in Fig. 1. If the distance between two lines ℓ_1 and ℓ_p is a positive even number, then ℓ_1 and ℓ_p are either both rows or both columns; if the distance is odd, then exactly one of ℓ_1 and ℓ_p is a row and one is a column.

Observation 1. *Let M be a matrix and let ℓ be a line of M. Then ℓ belongs to exactly one component M' of M and M' contains all 1-entries of ℓ.*

The following corollary is a direct consequence of Observation 1.

Corollary 2. *Let M be a matrix and let $M_1, \ldots, M_i$ be the components of M. If the column sets $F_1, \ldots, F_i$ are optimal solutions for* Min-Cos *on $M_1, \ldots, M_i$, respectively, then $F_1 \cup \ldots \cup F_i$ is an optimal solution for* Min-Cos *on M.*

Matrices that have the C1P can be characterized by a set of *forbidden submatrices* as shown by Tucker [18].

Theorem 2 ([18, Theorem 9]). *A matrix M has the C1P iff it contains none of the matrices M_{I_k}, M_{II_k}, M_{III_k} (with $k \geq 1$), M_{IV}, and M_{V} (see Fig. 2).*

The matrix type M_{I} is closely related to the matrix types M_{II} and M_{III}; this fact is expressed, in a more precise form, by the following two lemmas. They are used in the proof of our main structural result presented in Theorem 4.

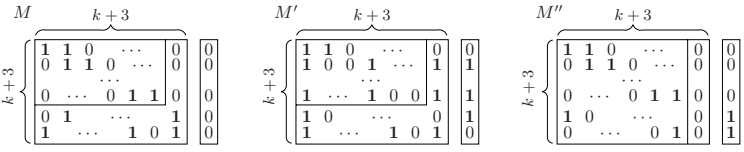


Fig. 3. An illustration of Lemma 1. Matrix M contains an M_{II_k} and a 0-column. Complementing rows r_2, r_{k+1}, and r_{k+2} of M leads to matrix M'. Complementing the rows of M' that have a 1 in column c_{k+3}, namely, r_2, r_{k+1}, and r_{k+3}, transforms M' to matrix M'' which contains an $M_{\mathrm{I}_{k+1}}$ and a 0-column, c_{k+3}.

Lemma 1. *For an integer $k \geq 1$, let M be a $(k+3) \times (k+4)$-matrix containing M_{II_k} and additionally a 0-column, and let M' be the matrix resulting from M by complementing a subset of its rows.*

Then complementing all rows of M' that have a 1 in column c_{k+3} results in a $(k+3) \times (k+4)$-matrix containing $M_{\mathrm{I}_{k+1}}$ and additionally a 0-column.

Proof. Let $R \subseteq \{1, 2, \ldots, k+3\}$ be the set of the indices of the complemented rows, that is, all rows r_i of M with $i \in R$ are complemented.

After complementing the rows $r_i, i \in R$, the column c_{k+3} of M' contains 1's in all rows r_i with $i \in (\{1, \ldots, k+1\} \cap R) \cup (\{k+2, k+3\} \setminus R)$. It is easy to see that complementing these rows of M' results in the described matrix. □

See Fig. 3 for an illustration of Lemma 1. The proof of the following lemma is analogous.

Lemma 2. *For an integer $k \geq 1$, let $M = M_{\mathrm{III}_k}$, and let M' be the matrix resulting from M by complementing a subset of its rows.*

Then complementing all rows of M' that have a 1 in column c_{k+3} results in a $(k+2) \times (k+3)$-matrix containing M_{I_k} and additionally a 0-column.

The Circular Ones Property, which is defined as follows, is closely related to the C1P, but is easier to achieve. It is used as an intermediate concept for dealing with the harder to achieve C1P.

Definition 1. *A matrix has the* Circular Ones Property (Circ1P) *if there exists a permutation of its columns such that in each row of the resulting matrix the 1's appear consecutively or the 0's appear consecutively (or both).*

Intuitively, if a matrix has the Circ1P then there is a column permutation such that the 1's in each row appear consecutively when the matrix is wrapped around a vertical cylinder. We have no theorem similar to Theorem 2 that characterizes matrices having the Circ1P; the following theorem of Tucker is helpful instead.

Theorem 3 ([17, Theorem 1]). *Let M be a matrix. Form the matrix M' from M by complementing all rows with a 1 in the first column of M. Then M has the Circ1P iff M' has the C1P.*

Corollary 3. *Let M be an $m \times n$-matrix and let j be an arbitrary integer with $1 \leq j \leq n$. Form the matrix M' from M by complementing all rows with a 1 in the j-th column of M. Then M has the Circ1P iff M' has the C1P.*

Algorithms. In order to derive a constant-factor polynomial-time approximation algorithm or a fixed-parameter algorithm for Min-COS on $(*, \Delta)$-matrices, we exploit Theorem 2 by iteratively searching and destroying in the given input matrix every submatrix that is isomorphic to one of the forbidden submatrices given in Theorem 2: In the approximation scenario all columns belonging to a forbidden submatrix are deleted, whereas in the fixed-parameter setting a search tree algorithm branches recursively into several subcases—deleting in each case one of the columns of the forbidden submatrix.

Observe that a $(*, \Delta)$-matrix cannot contain submatrices of types M_{II_k} and M_{III_k} with arbitrarily large sizes. Therefore, for both algorithms, the main difficulty is that every problem instance can contain submatrices of type M_{I_k} of unbounded size—the approximation factor or the number of cases to branch into would therefore not be bounded from above by Δ.

To overcome this difficulty, we use the following approach:

1. We first destroy only those forbidden submatrices that belong to a certain finite subset X of the forbidden submatrices given by Theorem 2 (and whose sizes are upper-bounded, therefore).
2. Then, we solve Min-COS for each component of the resulting matrix. As we will show in Lemma 3, this can be done in polynomial time. According to Corollary 2 these solutions can be combined into a solution for the whole input matrix.

The finite set X of forbidden submatrices is specified as follows.

Theorem 4. *Let $X := \{M_{\mathrm{I}_k} \mid 1 \leq k \leq \Delta-1\} \cup \{M_{\mathrm{II}_k} \mid 1 \leq k \leq \Delta-2\} \cup \{M_{\mathrm{III}_k} \mid 1 \leq k \leq \Delta-1\} \cup \{M_{\mathrm{IV}}, M_{\mathrm{V}}\}$. If a $(*, \Delta)$-matrix M contains none of the matrices in X as a submatrix, then each component of M has the Circ1P.*

Proof. Let M be a $(*, \Delta)$-matrix containing at most Δ ones per row, and let X be the set of submatrices mentioned in Theorem 4. We use Corollary 3 to show that if a component of a matrix M does not have the Circ1P, then this component contains a submatrix in X.

Let A be a component of M that does not have the Circ1P. Then, by Corollary 3, there must be a column c of A such that the matrix A' resulting from A by complementing those rows that have a 1 in column c does not have the C1P and, therefore, contains one of the submatrices given in Theorem 2.

In the following, we will make a case distinction based on which of the forbidden submatrices is contained in A' and which rows of A have been complemented, and show that in each case the matrix A contains a forbidden submatrix from X.

We denote the forbidden submatrix contained in A' with B' and the submatrix of A that corresponds to B' with B. Note that the matrix A' must contain a 0-column due to the fact that all 1's in column c have been complemented.

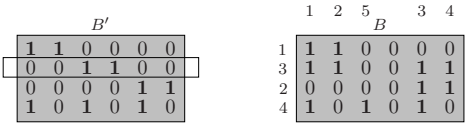


Fig. 4. Illustration for Case 1 in the proof of Theorem 4. Complementing the second row of an M_{IV} generates an M_{V}. (The rows and columns of the M_{V} are labeled with numbers according to the ordering of the rows and columns of the M_{V} in Fig. 2.)

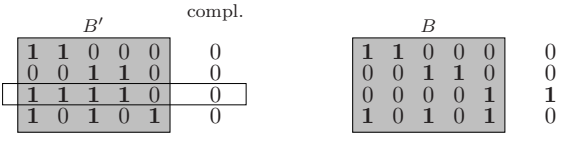


Fig. 5. Illustration for Case 2 in the proof of Theorem 4. Suppose that only the third row of B is complemented. Then B together with the complementing column forms an M_{IV}.

Because no forbidden submatrix given by Theorem 2 contains a 0-column, column c cannot belong to B' and, hence, not to B. We will call this column the *complementing column* of A.

When referencing to row or column indices of B', we will always assume that the rows and columns of B' are ordered as shown in Fig. 2.

Case 1: The submatrix B' is isomorphic to M_{IV}. If no row of B has been complemented, then $B = B'$, and A also contains a submatrix M_{IV}, a contradiction to the fact that M contains no submatrices from X.

If exactly one of the first three rows of B has been complemented, then B contains one 0-column, and B with the 0-column deleted forms an M_{V}, independent of whether the fourth row of B also has been complemented (see Fig. 4). Again, we have a contradiction.

If two or three of the first three rows of B have been complemented, then A contains an M_{I_1} as a submatrix: Assume w.l.o.g. that the first two rows have been complemented. If the fourth row has also been complemented, there is an M_{I_1} consisting of the rows r_1, r_2, r_4 and the columns c_2, c_4, c_5 of B. Otherwise, there is an M_{I_1} consisting of the rows r_1, r_2, r_4 and the columns c_1, c_3, c_6 of B. This is again a contradiction.

Case 2: The submatrix B' is isomorphic to M_{V}. Analogously to Case 1 we can make a case distinction on which rows of A have been complemented, and in every subcase we can find a forbidden submatrix from X in A. In some of the subcases the forbidden submatrix can only be found in A if in addition to B also the complementing column of A is considered. We will present only one example representing all subcases of Case 2. If e.g. only the third row of B has been complemented, then the complementing column of A contains a 0 in all rows that belong to B except for the third. Then B forms an M_{IV} together with the complementing column of A (see Fig. 5).

We omit the details for the other cases. Herein, the case that B' is isomorphic to M_{I_k}, $k \geq \Delta$, is the most complicated one; one has to distinguish two subcases corresponding to the parity of the distance between the complementing column and B. Lemma 1 and Lemma 2 are decisive for the cases that B' is isomorphic to M_{II_k}, $k \geq 1$, and that B' is isomorphic to M_{III_k}, $k \geq 1$, respectively. □

If all submatrices mentioned in Theorem 4 are destroyed, then every component of the resulting matrix has the Circ1P by Theorem 4. The next lemma shows that Min-COS on these components then is polynomial-time solvable.

Lemma 3. Min-COS *can be solved in* $O(n^{\Delta} \cdot m)$ *time, when restricted to* $(*, \Delta)$*-matrices that have the Circ1P. Herein,* n *denotes the number of columns and* m *the number of rows.*

We can now state the algorithmic results of this section.

Theorem 5. Min-COS *on* $(*, \Delta)$*-matrices for constant* Δ *can be approximated in polynomial time within a factor of* $\Delta + 2$ *if* $\Delta = 2$ *or* $\Delta \geq 4$*, and it can be approximated in polynomial time within a factor of* 6 *if* $\Delta = 3$*.*

Proof. Our approximation algorithm searches in every step for a forbidden submatrix of X given by Theorem 4 and then deletes all columns belonging to this submatrix. An optimal solution for the resulting matrix can be found in polynomial time: Due to Corollary 2 an optimal solution for a matrix can be composed of the optimal solutions for its components, and due to Theorem 4 after the deletion of all submatrices from X all components have the Circ1P, which means that an optimal solution for every component can be found in polynomial time as shown in Lemma 3.

Because an optimal solution has to delete at least one column of every forbidden submatrix of X, the approximation factor is the maximum number of columns of a forbidden submatrix from X.

The running time of the algorithm is dominated by searching for submatrix from X with the largest number of rows and columns. Such a submatrix M' with R rows and C columns can be found in $O(\min\{mn^C RC, m^R nRC\})$ time.

If $\Delta \neq 3$, every submatrix of X has at most $\Delta + 1$ rows and $\Delta + 2$ columns. If $\Delta = 3$, the matrix M_{IV} with four rows and six columns is the biggest matrix of X. This yields the claimed approximation factors and running times. □

Theorem 6. *Restricted to* $(*, \Delta)$*-matrices,* Min-COS *with parameter* $d =$ *"number of deleted columns" can be solved in* $O((\Delta + 2)^d \cdot (\min\{m^{\Delta+1}n, mn^{\Delta+2}\} + mn^{\Delta}))$ *time if* $\Delta = 2$ *or* $\Delta \geq 4$ *and in* $O(6^d \cdot (\min\{m^4 n, mn^6\} + mn^3))$ *time if* $\Delta = 3$*.*

Proof. We use a search tree approach, which searches in every step for a forbidden submatrix of X given by Theorem 4 and then branches on which column belonging to this submatrix has to be deleted. The solutions for the resulting matrices without submatrices from X can be found, without branching, in polynomial time due to Lemma 3. The number of branches depends on the maximum number of columns of a forbidden submatrix from X; the running time of the algorithm can therefore be determined analogously to Theorem 5. □

5 $(*, 2)$- and $(2, *)$-Matrices

$(*, 2)$-*Matrices.* By Theorem 6 we know that MIN-COS, restricted to $(*, 2)$-matrices, can be solved in $O(4^d \cdot m^3 n)$ time with d denoting the number of deleted columns. Here, we show that MIN-COS, restricted to $(*, 2)$-matrices, admits a quadratic-size problem kernel. Using the equivalence between MIN-COS restricted to $(*, 2)$-matrices and the minimization dual of MAXIMUM INDUCED DISJOINT PATHS SUBGRAPH (Max-IDPS) (see Sect. 3), we achieve this by showing a problem kernel for the dual problem of Max-IDPS with the parameter d denoting the number of allowed vertex deletions.

We use Min-IDPS to denote the dual problem of Max-IDPS and formulate Min-IDPS as a decision problem: The input consists of an undirected graph $G = (V, E)$ and an integer $d \geq 0$, and the problem is to decide whether there is a vertex subset $V' \subseteq V$ with $|V'| \leq d$ whose removal transforms G into a union of vertex-disjoint paths. W.l.o.g., we assume that G is a connected graph.

Given an instance $(G = (V, E), d)$ of Min-IDPS, we perform the following polynomial-time data reduction:

Rule 1: If a degree-two vertex v has two degree-at-most-two neighbors u, w with $\{u, w\} \notin E$, then remove v from G and connect u, w by an edge.

Rule 2: If a vertex v has more than $d + 2$ neighbors, then remove v from G, add v to V', and decrease d by one.

A graph to which none of the two rules applies is called *reduced*.

Lemma 4. *The data reduction rules are correct and a graph $G = (V, E)$ can be reduced in $O(|V| + |E|)$ time.*

Theorem 7. *Min-IDPS with parameter d denoting the allowed vertex deletions admits a problem kernel with $O(d^2)$ vertices and $O(d^2)$ edges.*

Proof. Suppose that a given Min-IDPS instance (G, d) is reduced w.r.t. Rules 1 and 2 and has a solution, i.e., by deleting a vertex subset V' with $|V'| \leq d$ the resulting graph $H = (V_H, E_H)$ is a union of vertex-disjoint paths. Then H has only degree-one and degree-two vertices, denoted by V_H^1 and V_H^2, respectively. Note that $V_H = V_H^1 \cup V_H^2 = V \setminus V'$.

On the one hand, since Rule 2 has removed all vertices of degree greater than $d+2$ from G, the vertices in V' are adjacent to at most $d^2 + 2d$ V_H-vertices in G. On the other hand, consider a V_H^1-vertex v. If v is not a degree-one vertex in G, then v is adjacent to at least one vertex from V'; otherwise, due to Rule 1, v's neighbor or the neighbor of v's neighbor is adjacent to at least one vertex from V'. Moreover, due to Rule 1 and the fact that H is a union of vertex-disjoint paths, at least one of three consecutive degree-two vertices on a path from H is adjacent in G to at least one vertex from V'. Hence, at least $|V_H|/5$ vertices in V_H are adjacent in G to V'-vertices. Thus, the number of V_H-vertices can be upper-bounded by $5 \cdot (d^2 + 2d)$.

Since H is a union of vertex-disjoint paths, there can be at most $|V_H| - 1 = 5d^2 + 10d - 1$ edges in H. As shown above, each vertex from V' can have at most $d + 2$ incident edges. □

$(2,*)$-*Matrices.* Here, we consider matrices M that have at most two 1's in each column but an unbounded number of 1's in each row. From Theorem 2, if M does not have the C1P, then M can only contain an M_{IV} or an M_{I_k} in Fig. 2.

The following lemma can be shown in a similar way as Theorem 4.

Lemma 5. *Let M be a $(2,*)$-matrix without identical columns. If M does not contain M_{IV} and M_{I_1} and does not have the C1P, then the matrices of type M_{I_k} that are contained in M are pairwise disjoint, that is, they have no common row or column.*

Based on Lemma 5, we can easily derive a search tree algorithm for Min-COS restricted to $(2,*)$-matrices:
1. Merge identical columns of the given matrix M into one column and assign to this column a weight equal to the number of columns identical to it;
2. Branch into at most six subcases, each corresponding to deleting a column from an M_{IV}- or M_{I_1}-submatrix found in M. By deleting a column, the parameter d is decreased by the weight of the column.
3. Finally, if there is no M_{IV} and M_{I_1} contained in M, then, by Lemma 5, the remaining M_{I}-submatrices contained in M are pairwise disjoint. Then, Min-COS is solvable in polynomial time on such a matrix: Delete a column with minimum weight from each remaining matrix of type M_{I_k}.

Clearly, the search tree size is $O(6^d)$, and a matrix of type M_{IV} can be found in $O(\min\{m^4n, mn^6\})$ steps, which gives the following theorem.

Theorem 8. Min-COS*, restricted to $(2,*)$-matrices with m rows and n columns, can be solved in $O(6^d \cdot \min\{m^4n, mn^6\})$ time where d denotes the number of allowed column deletions.*

By removing all columns of every M_{IV}- and M_{I_1}-submatrix found in M, one can show the following.

Corollary 4. Min-COS*, restricted to $(2,*)$-matrices, can be approximated in polynomial time within a factor of 6.*

6 Future Work

Our results mainly focus on Min-COS with no restriction on the number of 1's in the columns; similar studies should be undertaken for the case that we have no restriction for the rows. Moreover, it shall be investigated whether Δ can be eliminated from the exponents in the running times of the algorithms for Min-COS on $(*,\Delta)$-matrices—in the parameterized case the question is if Min-COS on $(*,\Delta)$-matrices is fixed-parameter tractable with d and Δ as a combined parameter, or if Min-COS on matrices without restrictions is fixed-parameter tractable with d as parameter. Finally, we focused only on deleting columns to achieve a maximum-size submatrix with C1P. It remains to consider the NP-hard symmetrical case [5] with respect to deleting rows; our structural characterization theorem should be helpful here as well.

References

1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.
2. I. Dinur and S. Safra. On the hardness of approximating Minimum Vertex Cover. *Annals of Mathematics*, 162(1):439–485, 2005.
3. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
4. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
6. M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1–2):59–84, 2000.
7. M. T. Hajiaghayi. Consecutive ones property, 2000. Manuscript, University Waterloo, Canada.
8. M. T. Hajiaghayi and Y. Ganjali. A note on the consecutive ones submatrix problem. *Information Processing Letters*, 83(3):163–166, 2002.
9. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4): 798–859, 2001.
10. W.-L. Hsu. A simple test for the consecutive ones property. *Journal of Algorithms*, 43:1–16, 2002.
11. W.-L. Hsu and R. M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, 2003.
12. S. Khot and O. Regev. Vertex Cover might be hard to approximate to within $2-\epsilon$. In *Proc. 18th IEEE Annual Conference on Computational Complexity*, pages 379–386. IEEE, 2003.
13. R. M. McConnell. A certifying algorithm for the consecutive-ones property. In *Proc. 15th ACM-SIAM SODA*, pages 768–777. SIAM, 2004.
14. J. Meidanis, O. Porto, and G. P. Telles. On the consecutive ones property. *Discrete Applied Mathmatics*, 88:325–354, 1998.
15. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
16. J. Tan and L. Zhang. The consecutive ones submatrix problem for sparse matrices. To appear in *Algorithmica*. Preliminary version titled "Approximation algorithms for the consecutive ones submatrix problem on sparse matrices" appeared in *Proc. 15th ISAAC*, Springer, volume 3341 of *LNCS*, pages 836–846, 2004.
17. A. C. Tucker. Matrix characterizations of circular-arc graphs. *Pacific Journal of Mathematics*, 2(39):535–545, 1971.
18. A. C. Tucker. A structure theorem for the consecutive 1's property. *Journal of Combinatorial Theory (B)*, 12:153–162, 1972.

Parameterized Algorithms for Weighted Matching and Packing Problems*

Yunlong Liu[1,2], Jianer Chen[1,3], and Jianxin Wang[1]

[1] College of Information Science and Engineering, Central South University, Changsha 410083, P.R. China
[2] School of Continuing Education, Hunan Normal University, Changsha 410012, P.R. China
[3] Department of Computer Science Texas AM University, College Station, TX 77843, USA
hnsdlyl@163.com, chen@cs.tamu.edu, jxwang@mail.csu.edu.cn

Abstract. The weighted m-D MATCHING and m-SET PACKING problems ($m \geq 3$) are usually solved through approximation algorithms. In this paper, we define the parameterized versions of these problems and study their algorithms. For the weighted m-SET PACKING problem, we develop a parameterized algorithm of running time $O(12.8^{mk}n^{O(1)})$, which is based on the recently improved color-coding technology and dynamic programming. By refining the techniques, we also develop a more efficient parameterized algorithm of running time $O(12.8^{(m-1)k}n^{O(1)})$ for the weighted m-D MATCHING problem, which is a restricted version of the weighted m-SET PACKING problem.

1 Introduction

Matching and packing problems are among the most often studied combinatorial problems. In particular, the m-D MATCHING problem and the m-SET PACKING problems are widely applied to many fields such as scheduling, coding optimization, and biology computation. The m-D MATCHING problem can be regarded as a special case of the m-SET PACKING problem. In the following, we first list a number of related definitions for the m-SET PACKING problem. A collection of sets is a *packing* if no two sets in the collection intersect.

Definition 1. m-SET PACKING ([7]): given a collection S of n sets, in which each set contains at most m elements, find a packing in S that contains the maximum number of sets.

A packing is a *k-packing* if it consists of exactly k sets.

* This work is supported by the National Natural Science Foundation of China (60433020) and the Program for New Century Excellent Talents in University (NCET-05-0683).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 692–702, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Definition 2. (parameterized) m-SET PACKING ([5]): given a pair (S, k), where S is a collection of n sets, in which each set contains at most m elements, and k is an integer, either construct a k-packing in S or report that no k-packing exists in S.

The m-PACKING problem can be generalized by allowing each set to have an associated *weight*, which is a non-negative real number. In the weighted case, the *weight* of a packing is defined to be the sum of the weights of the sets in the packing.

Definition 3. (weighted) m-SET PACKING ([2]): given a collection S of n weighted *sets (i.e., each set has an associated weight), each contains at most* m *elements, find a packing of the maximum weight in* S.

The m-SET PACKING problem is NP-complete [7], which means that it has no polynomial time exact algorithm unless P = NP. Nevertheless, researchers have been dealing with it through some effective approaches. In particular, with the development of parameterized computation theory, people have employed efficient parameterized computational techniques to solve the parameterized version of the problem (i.e. the problem defined in Definition 2) and obtained a series of improved algorithms [5,9,10]. Very recently, Liu *et al.* [10] developed an algorithm of running time $O^*(4.61^{3k})$ for the parameterized 3-SET PACKING problem using an improved color-coding construction, which is the best result up to now for the parameterized 3-SET PACKING problem.

On the other hand, to the authors's knowledge, there has not been study for parameterized algorithms for the weighted m-SET PACKING problem. The weighted m-SET PACKING problem is also NP-hard [2] and was studied in the research of approximation algorithms. The best approximation algorithm for the weighted m-SET PACKING problem is due to Chandra and Halldorsson [2], who gave an approximation algorithm of approximation ratio $2(m+1)/3$ for the problem.

The current paper is aimed at using the theory and techniques of parameterized computation theory to develop efficient algorithms for the weighted m-SET PACKING problem and the weighted m-D MATCHING problem. The parameterized weighted m-SET PACKING problem and the parameterized weighted m-D MATCHING problem are defined as follows.

Definition 4. (parameterized weighted) m-SET PACKING: given a pair (S, k), where S is a collection of n weighted sets, in which each set contains at most m elements, and k is an integer, either find a k-packing of the maximum weight in S, or report that no k-packing exists in S.

The m-D MATCHING problem is a restricted version of the m-SET PACKING problem. Let $A_1, \ldots, A_m$ be m pairwise disjoint finite symbol sets. Each (ordered) tuple $(a_1, \ldots, a_m)$, which is an element in $A_1 \times \cdots \times A_m$ (i.e., $a_i \in A_i$ for all i), is called an *m-tuple*. A *matching* is a collection of m-tuples in which no two tuples share a common symbol, and an *k-matching* is a matching containing exactly k

m-tuples. In the weighted case, each m-tuple is associated with a non-negative weight, and the *weight* of a matching is equal to the sum of the weights of the m-tuples in the matching.

Definition 5. (parameterized weighted) m-D MATCHING: given a pair (S, k), where S is a collection of n weighted m-tuples, and k is an integer, either find a k-matching of the maximum weight in S, or report no k-matching exists in S.

We remark that parameterized algorithms for other weighted NP-hard problems have appeared in the literature. For instance, Fernau studied the parameterized complexity for the weighted VERTEX COVER problem and for the weighted EDGE SUBGRAPH problem [6], and Guo and Niedermeier developed parameterized algorithms for the weighted tree-like SET COVER problem [8].

We will concentrate on parameterized algorithms for the problems given in Definition 4 and Definition 5. The paper is organized as follows. In Section 2, we provide some related preliminaries and lemmas. In Section 3, we present in detail a parameterized algorithm for the weighted m-SET PACKING problem. In Section 4, on the basis of the previous section, we give a more efficient parameterized algorithm for the weighted m-D MATCHING problem. Section 5 provides conclusion of the paper.

2 Preliminaries

Before presenting our parameter algorithms, we give some related terminologies and lemmas.

Fix m finite symbol sets A_1, ..., A_m. For an m-tuple $\rho = (a_1, \ldots, a_m)$ in $A_1 \times \cdots \times A_m$, denote by $\mathrm{Val}(\rho)$ the set $\{a_1, \ldots, a_m\}$, and let $\mathrm{Val}^i(\rho) = \{a_i\}$ for all $1 \leq i \leq m$. For a collection S of m-tuples, define $\mathrm{Val}(S) = \bigcup_{\rho \in S} \mathrm{Val}(\rho)$, and $\mathrm{Val}^i(S) = \bigcup_{\rho \in S} \mathrm{Val}^i(\rho)$, for all $1 \leq i \leq m$. The symbols in $\mathrm{Val}^i(S)$ will be called the *i-th dimension symbols* of S.

Definition 6. ([6]) A parameterized maximization problem P can be specified by: (1). a set I_P of *instances*; (2) for each instance $x \in I_P$, a set of *feasible solutions* $S_P(x)$; (3) a *measure function* f_P yielding the value $f_P(x, y)$ for any feasible solution $y \in S_P(x)$ for an instance $x \in I_P$; and (4) a distinguished part of the input called the *parameter* k that is an integer. The objective is for each given instance $x \in I_P$ to find a feasible solution $y \in S_P(x)$ such that $f(x, y)$ is maximized.

To study the complexity of parameterized maximization problems, we introduce the following concept.

Definition 7. A parameterized maximization problem P is *fixed parameter tractable* if there is an algorithm A that, given an instance x in I_P, returns a feasible solution y_0 in $S_P(x)$ such that $f_P(x, y_0) = \max\{f_P(x, y) \mid y \in S_P(x)\}$ if $S_P(x) \neq \emptyset$; or reports that $S_P(x) = \emptyset$. Moreover, the running time of the algorithm A should be bounded by $O(g(k)|x|^{O(1)})$, for a fixed function g.

Recent research has shown that color-coding [1] is a very important technique for solving parameterized problems [3]. It will also be the main technique used in this paper. For this, we give the precise definition and related results as follows.

Definition 8. Let S be a set of elements. A *k-coloring* of S is a function mapping S to the set $\{1, 2, \ldots, k\}$. A subset W of S is colored *properly* by a coloring f if no two elements in W are colored with the same color under f.

Definition 9. A collection $\mathcal{C}$ of k-colorings of a set S is a *k-color coding scheme* if for every subset W of k elements in S, there is a k-coloring in $\mathcal{C}$ that colors W properly. The *size* of the k-color coding scheme $\mathcal{C}$ is the number of k-colorings in $\mathcal{C}$.

The concept of color-coding was introduced by Alon, Yuster, and Zwick [1], who proved that for a set of n elements there exists a k-color coding scheme whose size is bounded by $O(2^{O(k)}n)$. Progress has been made recently. Chen *et al.* [3] presented a construction that gives a k-color coding scheme of size $O(6.4^k n)$ for a set of n elements, and developed an algorithm of running time $O(6.4^k n)$ that generates such a k-color coding scheme.

Lemma 1 ([3]). *For any finite set S of n elements and any integer k, $n \geq k$, there is a k-color coding scheme $\mathcal{C}$ of size $O(6.4^k n)$ for the set S. Moreover, such a k-color coding scheme can be constructed in time $O(6.4^k n)$.*

3 An Algorithm for Weighted m-SET PACKING

We first present an algorithm for the weighted m-SET PACKING problem. Let (S, k) be an instance of the problem, where S is a collection of weighted m-sets and k is the parameter. To simplify our discussion, for an element a in an m-set in a sub-collection S' of S, we simply say that a is an element in S'.

Suppose that we use an (mk)-coloring f to color all the elements in S. We say that a packing P in S is *properly colored* if no two elements in P are colored with the same color under f.

First we suppose that the collection S contains a k-packing. Moreover, we suppose that at least one of the k-packings in the collection S is properly colored under f. We give an algorithm that, under the k-coloring f, finds a properly colored k-packing that has the maximum weight over all properly colored k-packings in the collection S.

We need some notations and terminologies before we present our algorithm. For a packing P in S, we will use the notation $cl(P)$ to denote the set of colors that are used to color the elements in the packing P. In this case, we also say that the packing P *uses* the color set $cl(P)$. Note that the color set $cl(P)$ contains exactly ml colors if the packing P is properly colored and contains exactly l m-sets, and that two packings may use the same color set.

Let $\mathcal{Q}$ be a super-collection[1] of properly colored packings, and let P be a properly colored packing not in $\mathcal{Q}$. By "updating the super-collection $\mathcal{Q}$ with the packing P", we mean we perform the following procedure on $\mathcal{Q}$: (1) if $\mathcal{Q}$ does not contain a properly colored packing that uses the color set $cl(P)$ and if P contains no more than k m-sets, then add P to $\mathcal{Q}$; (2) if $\mathcal{Q}$ contains a properly colored packing P_1 that uses $cl(P)$, but the weight of P_1 is smaller than the weight of P, then replace P_1 in $\mathcal{Q}$ by P; and (3) in all other cases, do nothing.

Our algorithm is given in Figure 1.

Algorithm SLMP(S, k, f)
Input: A collection S of weighted m-sets in which elements are colored by an (mk)-coloring f, and an integer k
Output: a properly colored k-packing whose weight is the maximum over all properly colored k-packings in S;

1. remove all m-sets in S in which any two elements have the same color;
2. let the remaining m-sets in S be ρ_1, ρ_2,$\cdots$, ρ_r;
3. $\mathcal{Q} = \emptyset$;
4. **for** $i = 1$ **to** r **do**
   for each packing P in $\mathcal{Q}$ **do**
   if $cl(P) \cap cl(\rho_i) = \emptyset$
   then $P' = P \cup \{\rho_i\}$; update $\mathcal{Q}$ with P';
5. **if** $\mathcal{Q}$ contains no k-packing
   then return "no properly colored k-packing in S"
   else return the k-packing of the maximum weight in $\mathcal{Q}$.

Fig. 1. An algorithm on a colored collection of m-sets

Theorem 1. *If the collection S contains at least one properly colored k-packing, then the algorithm SLMP(S, k, f) returns a properly colored k-packing whose weight is the maximum over all properly colored k-packings in S.*

Proof. Step 1 is obviously correct: no m-set in which two elements have the same color can be in a properly colored packing.

From step 4 of the algorithm, it can be seen that every packing added to the super-collection $\mathcal{Q}$ is a properly colored packing. Therefore, if the algorithm returns a k-packing in step 5, the packing must be a properly colored k-packing.

At step 3, we assume that the collection S contains the m-sets ρ_1, ρ_2, ..., ρ_r.

For each i, $1 \leq i \leq r$, let $S_i = \{\rho_1, \rho_2, \cdots, \rho_i\}$. We prove by induction on i for the following claim:

Claim. For all $j \leq k$, if S_i contains a properly colored j-packing P_0, then after the i-th execution of the **for**-loop in step 4 of the algorithm,

[1] We use "super-collection" for a collection of packings because each packing itself is a collection of m-sets.

the super-collection $\mathcal{Q}$ contains a properly colored j-packing that uses the color set $cl(P_0)$ and has the maximum weight in S_i over all properly colored j-packings that use the color set $cl(P_0)$.

The initial case $i = 0$ is trivial since $S_i = \emptyset$ and $\mathcal{Q} = \emptyset$.

Now consider a general value $i \geq 1$. Suppose that the collection S_i has a properly colored j-packing P_0 ($j \leq k$). Without loss of generality, let P_0 be a properly colored j-packing that has the maximum weight over all properly colored j-packings in S_i that use the color set $cl(P_0)$. Let ρ_q be the m-set in P_0 with the largest index q. There are two cases.

Case 1. $q < i$. Then P_0 is also a properly colored j-packing in the collection S_q. Obviously, P_0 has the maximum weight over all properly colored j-packings in S_q that use the color set $cl(P_0)$. Therefore, by the inductive hypothesis, after the q-th execution of the **for**-loop in step 4, the super-collection $\mathcal{Q}$ contains a properly colored j-packing P that uses the color set $cl(P_0)$ and has the maximum weight over all properly colored j-packings in S_q that use the color set $cl(P_0)$. This implies that P and P_0 have the same weight and use the same color set. Since the only operations we allow to update the super-collection $\mathcal{Q}$ are to add a packing and to replace a packing of smaller weight by a packing using the same color set but with a larger weight, we conclude that after the i-th execution of the **for**-loop in step 4 (note $q < i$), the super-collection $\mathcal{Q}$ must contain a properly colored j-packing P that uses the color set $cl(P_0)$ and has weight at least as large as that of P_0. By the definition of P_0, P is a properly colored j-packing that uses the color set $cl(P_0)$ and has the maximum weight over all properly colored j-packings in S_i that use the color set $cl(P_0)$. Thus, the induction goes through.

Case 2. $q = i$. Then all m-sets ρ_p in $P_1 = P_0 - \rho_q$ must have their index p bounded by $i-1$. Thus, P_1 is a properly colored $(j-1)$-packing in the collection S_{i-1}. In fact, P_1 must have the maximum weight over all properly colored $(j-1)$-packings in S_{i-1} that use the color set $cl(P_1)$ – otherwise, the packing P_0 would have not been a properly colored j-packing with the maximum weight over all j-packings in S_i that use the color set $cl(P_0)$. By the inductive hypothesis on $i-1$, after the $(i-1)$-th execution of the **for**-loop in step 4, the super-collection $\mathcal{Q}$ contains a properly colored $(j-1)$-packing P_1' that uses the color set $cl(P_1)$ and has the same weight as P_1. Therefore, during the i-th execution of step 4, when this $(j-1)$-packing P_1' is considered, the m-set ρ_q and P_1' will make a properly colored j-packing, which will consequently makes the super-collection $\mathcal{Q}$ to include a properly colored j-packing that uses the color set $cl(P_0)$ and has weight equal to the weight of P_0. Thus, the induction also goes through.

This proves the claim. The theorem now follows directly: let P_0 be a properly colored k-packing whose weight is the maximum over all properly colored k-packings in S. Since $S = S_r$, by the claim, when step 4 is completely done and we reach step 5, the super-collection $\mathcal{Q}$ contains a properly colored k-packing P that uses the color set $cl(P_0)$ and has weight at least as large as that of P_0. In consequence, the k-packing P is properly colored and has the maximum weight over all properly colored k-packings in the collection S. □

The complexity of the algorithm SLMP is given in the following theorem.

Theorem 2. *The time complexity of the algorithm SLMP(S, k, f) is* $O(2^{mk}knm)$*, where* n *is the number of* m*-sets in the input collection* S*.*

Proof. By our rules of updating the super-collection $\mathcal{Q}$, $\mathcal{Q}$ contains only properly colored j-packings, where $j \leq k$. In particular, each packing in $\mathcal{Q}$ uses a color set that contains at most km colors. Moreover, for each color set of at most km colors, the supper-collection $\mathcal{Q}$ contains at most one packing that uses the color set. Since there are no more than 2^{mk} color sets that use at most mk colors, we conclude that the total number of packings in $\mathcal{Q}$ is bounded by 2^{mk}. Now in step 4, for each packing P in $\mathcal{Q}$ (there are at most 2^{km} such P) and for each m-set ρ_i (there are at most n such ρ_i), in time $O(mk)$ we can check if P and ρ_i can make a new properly colored packing and how $\mathcal{Q}$ can be updated (for this we can keep an array of size 2^{km} to record the status of all color sets). In conclusion, the running time of the algorithm is bounded by $O(2^{mk}knm)$. □

Now we are ready for our main result for this section. Consider the algorithm given in Figure 2.

Algorithm SMP(S, k)
Input: A collection S of weighted m-sets, and an integer k
Output: a k-packing of the maximum weight in S;

1. $P_0 = \emptyset$;
2. let μ be the set of all elements appeared in S; let $N = |\mu|$;
3. construct an (mk)-color coding scheme $\mathcal{C}$ of size $O(6.4^{km}N)$ for the set μ;
4. **for** each (mk)-coloring f in $\mathcal{C}$ **do**
4.1 color the elements in S by f;
4.2 call SLMP(S, k, f);
4.3 **if** step 4.2 returns a k-packing P whose weight is larger than that of P_0
4.4 **then** $P_0 = P$;
5. **if** $P_0 = \emptyset$
then return "S contains no k-packing"
else return P_0.

Fig. 2. Algorithm for weighted m-SET PACKING

Theorem 3. *The algorithm SMP solves the parameterized weighted* m-SET PACKING *problem in time* $O(12.8^{mk}(mn)^2k)$*, where* n *is the number of* m*-sets in the collection* S*.*

Proof. The existence of the (mk)-color coding scheme $\mathcal{C}$ and its construction complexity are given by Lemma 1.

Suppose that the collection S contains a k-packing. Let P_1 be a k-packing in S whose weight is the maximum over all k-packings in S. The packing P_1 contains exactly mk elements. Since $\mathcal{C}$ is an (mk)-color coding scheme for the element set μ, there is an (mk)-coloring f_0 in $\mathcal{C}$ such that P_1 is properly colored under f_0. When step 4.2 calls the algorithm SLMP(S, k, f_0), by Theorem 1, a

properly colored k-packing P is returned whose weight is at least as large as that of P_1. By the definition of the packing P_1, the packing P is a k-packing of the maximum weight. Thus, the following step 4.4 will record a k-packing of the maximum weight in P_0, which will be returned in step 5. Thus, in this case the algorithm SMP(S, k) will correctly produce a k-packing of the maximum weight in the collection S.

On the other hand, if the collection S does not contain a k-packing, then since the algorithm SLMP is correct, step 4.2 will never return a k-packing, and the set P_0 will keep empty. Thus, in this case, the algorithm SMP(S, k) will correctly report in step 5 that the collection S contains no k-packing.

Finally, we consider the complexity of the algorithm. Since the (mk)-color coding scheme $\mathcal{C}$ has $O(6.4^{mk}N) = O(6.4^{mk}mn)$ (mk)-colorings, the loop 4.1-4.4 is executed $O(6.4^{mk}mn)$ times. Each execution of the loop, by Theorem 2, takes time $O(2^{mk}knm)$. We conclude that the running time of the algorithm SMP(S, k) is bounded by $O(12.8^{mk}(mn)^2k)$. □

4 An Algorithm for Weighted m-D MATCHING

The weighted m-D MATCHING problem is a restricted version of the weighted m-SET PACKING problem. Therefore, the algorithm presented in the previous section can be directly applied to the weighted m-D MATCHING problem. In this section, we show that for this restricted version, we can develop an even more efficient algorithm.

Recall that an instance of the parameterized weighted m-D MATCHING problem is a pair (S, k), where S is a collection of weighted m-tuples, and each m-tuple is an element in the product set $A_1 \times \cdots \times A_m$ of m pairwise disjoint symbol sets $A_1, \ldots, A_m$. Instead of coloring all symbols in the collection S using an (mk)-coloring, we will only use an $((m-1)k)$-coloring to color the symbols in the 2nd to the m-th dimensions in S. Again let f be an $((m-1)k)$-coloring on the symbols in the 2nd to the m-th dimensions in S. Again we say that a matching is *properly colored* if no two symbols in its 2nd to m-th dimensions are colored with the same color.

Similar to what we did for m-SET PACKING, for a matching M in S, we will use the notation $cl(M)$ to denote the set of colors that are used to color the (2nd to m-th dimension) symbols in the matching M. In this case, we also say that the matching M *uses* the color set $cl(M)$. For a super-collection $\mathcal{Q}$ of properly colored matchings and for a properly colored matching M not in $\mathcal{Q}$, we update the super-collection $\mathcal{Q}$ with the matching M by performing the following procedure on $\mathcal{Q}$: (1) if $\mathcal{Q}$ does not contain a properly colored matching that uses the color set $cl(M)$ and if M contains no more than k m-tuples, then add M to $\mathcal{Q}$; (2) if $\mathcal{Q}$ contains a properly colored matching M_1 that uses the color set $cl(M)$, but the weight of M_1 is smaller than the weight of M, then replace M_1 in $\mathcal{Q}$ by M; and (3) in all other cases, do nothing.

Consider the algorithm given in Figure 3.

Algorithm SLMM(S, k, f)
Input: A collection S of weighted m-tuples in which symbols in the 2nd to the m-th dimensions are colored by an $((m-1)k)$-coloring f, an integer k
Output: a properly colored k-matching whose weight is the maximum over all properly colored k-matchings in S;

1. remove all m-tuples in S in which any two symbols have the same color;
2. let the set of the remaining m-tuples be S';
3. let the symbols in the first dimension of S' be $x_1, \ldots, x_h$;
4. $\mathcal{Q}_{old} = \{\emptyset\}$; $\mathcal{Q}_{new} = \{\emptyset\}$;
5. **for** $i = 1$ **to** h **do**
5.1 let $S'_i = \{\rho_1, \ldots, \rho_r\}$ be the collection of m-tuples in S' whose symbols in the first dimension are in $\{x_1, \ldots, x_i\}$;
5.2 **for** $j = 1$ **to** r **do**
for each matching M in $\mathcal{Q}_{old}$ **do**
if $cl(M) \cap cl(\rho_j) = \emptyset$
then $M' = M \cup \{\rho_j\}$; update $\mathcal{Q}_{new}$ with M';
5.3 $\mathcal{Q}_{old} = \mathcal{Q}_{new}$;
6. **if** $\mathcal{Q}_{old}$ contains no k-matching
then return "no properly colored k-matching in S"
else return the k-matching of the maximum weight in $\mathcal{Q}_{old}$.

Fig. 3. Algorithm for weighted m-D MATCHING

Theorem 4. *If S has at least one properly colored k-matching, then the algorithm SLMM(S, k, f) returns a properly colored k-matching whose weight is the maximum over all properly colored k-matchings in the collection S. The running time of the algorithm SLMM(S, k, f) is bounded by $O(2^{(m-1)k}kn^2m)$, where n is the total number of m-tuples in the input collection S.*

Proof. The proof of the correctness of the algorithm SLMM(S, k, f) is similar to that for the algorithm SLMP, which was given in the proof of Theorem 1. Thus, we only describe the difference here.

As given in the algorithm, let $S'_i = \{\rho_1, \ldots, \rho_r\}$ be the collection of m-tuples in S' whose symbols in the first dimension are in $\{x_1, \ldots, x_i\}$. Define $S'_{ij} = \{\rho_1, \ldots, \rho_j\}$ for $1 \le j \le r$. Then we can prove the following claim by induction.

> **Claim.** For all $q \le k$, if S'_{ij} contains a properly colored q-matching P_0, then during the i-th execution of the **for**-loop in step 5 and after the j-th execution of the **for**-loop in step 5.2, the super-collection $\mathcal{Q}_{new}$ contains a properly colored q-matching in S'_{ij} that uses the color set $cl(P_0)$ and has the maximum weight over all properly colored q-matchings in S'_{ij} that use the color set $cl(P_0)$.

With this claim, as proceeded in Theorem 1, we can easily prove that the algorithm SLMM(S, k, f) produces a properly colored k-matching of the maximum weight over all properly colored k-matchings in the collection S, in case such a k-matching exists.

As for the running time of the algorithm, note that f is an $((m-1)k)$-coloring that uses only $(m-1)k$ colors, and that for each color set, the super-collection $\mathcal{Q}_{old}$ (and the super-collection $\mathcal{Q}_{new}$) records at most one matching. Thus, the total number of matchings in $\mathcal{Q}_{old}$ is bounded by $2^{(m-1)k}$. From this, we can prove, in the way similar to that for the algorithm SLMP, that an execution of the entire step 5.2 takes time bounded by $O(2^{(m-1)k}knm)$. Now the theorem follows because the number h of symbols in the first dimension of S is bounded by the number n of m-tuples in S. □

Now, completely similar to that of Theorem 3, but by constructing an $((m-1)k)$-color coding scheme for the symbols in the 2nd to the m-th dimensions in the collection S, we conclude with the following theorem.

Theorem 5. *The parameterized weighted* m-D MATCHING *problem can be solved in time* $O(12.8^{(m-1)k}m^2n^3k)$*, where* n *is the number of* m*-tuples in the input collection* S*.*

5 Conclusions

In this paper, we study the parameterized weighted m-SET PACKING problem and the parameterized weighted m-D MATCHING problem. By applying the theory and techniques recently developed in parameterized computation theory, we developed fixed-parameter tractable algorithms for the problems. As far as we know, the fixed-parameter tractable algorithms presented in the current paper are the first group of exact algorithms for solving these weighted problems.

Subgraph packing problems have recently drawn much attention [4,11]. The algorithms presented in the current paper can be directly applied to solve the weighted versions of many subgraph packing problems, such as the GRAPH PACKING problem and the GRAPH EDGE-PACKING problem studied in [4,11].

References

1. N. Alon, R. Yuster, and U. Zwick, Color-coding, *Journal of the ACM 42*, (1995), pp. 844–856.
2. B. Chandra and M. Halldorsson, Greedy local improvement and weighted set packing approximation, *Journal of Algorithms 39*, (2001), pp. 223–240.
3. J. Chen, S. Lu, S.-H. Sze, and F. Zhang, Improved algorithms for path, matching, and packing problems, *Proc. 18th Annual ACM-SIAM Symp. on Discrete Algorithms* (SODA 07), (2007), pp. 298–307.
4. M. Fellows, P. Heggernes, F. Rosamond, C. Sloper, and J. Telle, Finding k disjoint triangles in an arbitrary graph, *Lecture Notes in Computer Science 3353* (WG 2004), (2004), pp. 235–244.
5. M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. Rosamond, U. Stege, D. Thilikos, and S. Whitesides, Faster Fixed-parameter tractable algorithms for matching and packing problems, *Lecture Notes in Computer Science 3221* (ESA 2004), (2004), pp. 311–322.

6. H. Fernau, Parameterized maximization, *Technical Report WSI-2001-22*, UniversityTubingen (Germany), Wilhelm-Schickard-Institut fur Informatik, (2001).
7. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman Co., 1979.
8. J. Guo and R. Niedermeier, Exact algorithms and applications for tree-like weighted set cover, *Journal of Discrete Algorithms 4*, (2006), pp. 608–622.
9. W. Jia, C. Zhang, and J. Chen, An efficient parameterized algorithm for m-Set Packing, *Journal of Algorithms 50*, (2004), pp. 106–117.
10. Y. Liu, S. Lu, J. Chen, and S.-H. Sze, Greedy localization and color-coding: improved matching and packing algorithms, *Lecture Notes in Computer Science 4169* (IWPEC 06), (2006), pp. 84–95.
11. L. Mathieson, E. Prieto, and P. Shaw, Packing edge disjoint triangles: a parameterized view, *Lecture Notes in Computer Science 3162* (IWPEC 2004), (2004), pp. 127–137.

Kernelizations for Parameterized Counting Problems

Marc Thurley*

Institut für Informatik, Humboldt-Universität zu Berlin
thurley@informatik.hu-berlin.de

Abstract. Kernelizations are an important tool in designing fixed parameter algorithms for parameterized decision problems. We introduce an analogous notion for counting problems, to wit, *counting kernelizations* which turn out to be equivalent to the fixed parameter tractability of counting problems. Furthermore, we study the application of well-known kernelization techniques to counting problems. Among these are the Buss Kernelization and the Crown Rule Reduction for the vertex cover problem. Furthermore, we show how to adapt kernelizations for the hitting set problem on hypergraphs with hyperedges of bounded cardinality and the unique hitting set problem to their counting analogs.

1 Introduction

In the last decade, *parameterized complexity* matured as a field of refined complexity analyses of algorithms and decision problems. It replaces the classical notion of tractability with *fixed parameter tractability* [12], requiring tractable problems to be solvable by a deterministic algorithm in time $f(k) \cdot n^{O(1)}$ for some (small) parameter k, a computable function $f : \mathbb{N} \to \mathbb{N}$ and n the input size. This notion is tightly connected to the concept of kernelization. Intuitively, a kernelization is a polynomial time computable function mapping problem instances to instances whose size is bounded effectively in terms of the parameter. Ever since its introduction in [11] this concept has found may applications with the result that today most fixed parameter algorithms use techniques that are inspired by this notion [15,8,19,20].

Notably, all of the work mentioned is devoted to decision problems. In contrast, the study of parameterized *counting* problems has not yet matured as far. Some work has been done on carrying over fixed parameter algorithm design to counting problems. For example, the best known algorithms following this approach solve the counting analog of the vertex cover problem in time $O(1.62^k n)$ (due to Fernau [16] - another, yet unpublished, result by Rossmanith [24] establishes an even lower bound of $O(1.47^k n)$). For some other counting problems fixed parameter algorithms were developed [10,17,22] as well. A variety of more

* This research was supported in part by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 703–714, 2007.
© Springer-Verlag Berlin Heidelberg 2007

general results states the fixed parameter tractability of large classes of counting problems [2,7,18].

However, a notion of kernelization that is applicable to counting problems has not yet been developed, although there has been some recent interest in forming a notion of this kind. Nishimura et al. [22] suggested *compactor enumeration*, which performs a reduction to an enumeration problem on an instance with size depending only on the parameter. However, this notion has not yet been thoroughly formalized. In particular, the informal definition of a compactor from [22] is not applicable to counting problems in general.

We will remedy this deficiency by formally describing kernelizations of counting problems. Moreover, we will show that this notion is equivalent to the notion of fixed parameter tractability for counting problems and we give some examples, how this notion can be applied in practice. These examples will include vertex cover and hitting set problems.

There are at least three important kernelizations of the vertex cover decision problem: the Buss kernelization [11], the Crown Rule Reduction [15] and the kernel based on the Nemhauser-Trotter theorem [5]. We will apply the former two to the counting problem (the Nemhauser-Trotter kernel can be applied to counting in a way analogous to that of the Crown Rule [26]). Furthermore, we will adapt the Sunflower kernel [17] for the hitting set problem on hypergraphs with edges of bounded cardinality. Additionally, for the unique hitting set problem [13] we will develop an adaptation of a well-known kernelization of the decision problem to counting.

This paper is organized as follows. Section 1 gives the basic definitions and the concept of counting kernelization. In Section 2 we give kernelizations of vertex cover and hitting set counting problems. Section 3 is devoted to the Crown Rule Reduction. Then, in section 4, we will kernelize the counting unique hitting set problem and the last section will reveal the equivalence of counting kernelizations and fixed parameter tractable counting problems.

2 Definitions and the Concept of Counting Kernelizations

We will briefly sketch the basic notions of fixed parameter tractability. For a more thorough introduction to the field the reader is refered to one of [13,21,17].

Parameterized problems are formed from classical problems by adding a *parameterization of the input alphabet* Σ, that is, a polynomial time computable function $\kappa : \Sigma^* \to \mathbb{N}$. Thus for classical decision problems $Q \subseteq \Sigma^*$ and counting problems $F : \Sigma^* \to \mathbb{N}$ we obtain parameterized analogs (Q, κ) and (F, κ). We denote the class of all fixed parameter tractable decision problems by FPT and that of fixed parameter tractable counting problems by $FFPT$.

For parameterized decision problems, a *kernelization* is a mapping $K : \Sigma^* \to \Sigma^*$ satisfying the following conditions.

(K1) K is polynomial time computable and there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ we have

$$|K(x)| \leq g(\kappa(x))$$

(K2) $x \in Q \Leftrightarrow K(x) \in Q$, for all $x \in \Sigma^*$

Hence, if a problem (Q, κ) has a kernelization it is fixed parameter tractable.

This concept has indeed led to the creation of some of the most efficient fpt-algorithms and it is one of the most frequently applied approaches in parameterized algorithm design. Considering this popularity of kernelization in the field of decision problems one would naturally ask, if this notion can be applied to counting problems as well. We will discuss this question now.

2.1 Counting Kernelization

Note that in this purely theoretical context we regard computations always as computations of Turing machines. We begin with some preliminary definitions.

Definition 1. *Let $\kappa : \Sigma^* \to \mathbb{N}$ be a parameterization of Σ^*.*

(I) A mapping $K : \Sigma^* \to \Sigma^*$ is κ-bounded *if there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$:*

$$|K(x)| \leq g(\kappa(x)).$$

(II) A relation $Y \subseteq \Sigma^* \times \{0,1\}^*$ is K-aware *for a κ-bounded mapping K if there are computable functions $h, f : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$ and every $y \in \{0,1\}^*$:*

(1) Given $K(x)$, it can be decided in time $f(\kappa(x))$ whether $(K(x), y) \in Y$ holds.

(2) if $(K(x), y) \in Y$ then $|y| \leq h(\kappa(x))$.

Note that the notion of *κ-boundedness* is simply derived from the notion of kernelization by omitting (K2), i.e. the part that deals explicitly with decision problems. Furthermore K-aware relations Y will be used to characterize the "solutions" of the reduced instance $K(x)$ that have to be found by search algorithms. Unfortunately, the formal definition of a counting problem does not mention what a solution might be. Therefore we will formalize this by the well-known notion of *witnesses*:

Let $Y \subseteq \Sigma^* \times \{0,1\}^*$ be a relation. For $x \in \Sigma^*$ we define $w_Y(x) := \{y \,|\, (x,y) \in Y\}$ as the set of *witnesses* of x in Y. Note that K-aware relations are compatible with the notions of *P-relations* from classical complexity (see e.g. [23]) in such a way that every P-relation is a K-aware relation.

A notion of counting kernelization needs an additional ingredient. This will become clear by an example. Consider the following problem:

p-#VertexCover

Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$
Parameter: k
Problem: Compute the number of vertex covers of size k in $\mathcal{G}$

Let $\#vc(\mathcal{G}, k)$ denote the number of size k vertex covers in $\mathcal{G}$. Let $K : \Sigma^* \to \Sigma^*$ be κ-bounded, mapping instances of p-#VertexCover to instances of the same problem s.t. for all graphs $\mathcal{H}$ we have $|K(\mathcal{H}, k)| \leq g(k)$, for some computable function $g : \mathbb{N} \to \mathbb{N}$.

Example 1. Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. Suppose that $\#vc(K(\mathcal{G}, k)) = \#vc(\mathcal{G}, k)$. Let $K(\mathcal{G}, k) =: (\mathcal{G}', k')$ for a graph $\mathcal{G}' = (V', E')$ and $k' \in \mathbb{N}$. As $\|\mathcal{G}'\| \leq g(k)$, we know that there can be no more than $\binom{g(k)}{k'}$ vertex covers of cardinality k' in $\mathcal{G}'$.

Let $\mathcal{H} = (W, \emptyset)$ be a graph with $|W| = g(k) + k$. It is easy to see that

$$\#vc(\mathcal{H}, k) = \binom{g(k)+k}{k} > \binom{g(k)}{k} \geq \#vc(K(\mathcal{H}, k)).$$

This example illustrates, that the reduced instance $K(x)$ may have less solutions than x itself. Hence we need a function, say μ, that for each solution y of the reduced instance $K(x)$ computes the number of solutions of an instance x that are "represented" by y. Moreover, the function μ has to depend on the original instance x to be able to create the link between $K(x)$ and x.

Definition 2 (Counting Kernelization). *Let (F, κ) be a parameterized counting problem over Σ. A* counting kernelization of (F, κ) *is a pair (μ, K) of polynomial time computable functions $K : \Sigma^* \to \Sigma^*$ and $\mu : \Sigma^* \times \Sigma^* \times \{0,1\}^* \to \mathbb{N}$ such that for all $x \in \Sigma^*$ the following is satisfied:*

(1) K is κ-bounded
(2) There is a K-aware relation $Y \subseteq \Sigma^* \times \{0,1\}^*$ such that

$$F(x) = \sum_{y \in w_Y(K(x))} \mu(x, K(x), y) \tag{1}$$

We call $|K(x)|$ the size of the kernel.

Note that, on an informal basis, this notion follows the intuition given in [22], to wit, a counting kernelization reduces a counting problem to an enumeration problem on an instance that is bounded in terms of the parameter. This is done in such a way that we can associate solutions of the kernel to (sets of) solutions of the original instance by an efficiently computable function μ.

Nevertheless, this definition still seems somewhat arbitrary. Its motivation will become clear if we know how to apply it to some counting problems. We have to note that in the course of this, we will sloppily define K-aware relations Y that do not directly satisfy $Y \subseteq \Sigma^* \times \{0,1\}^*$, however, it will always be clear that they can be represented straightforwardly in such a way.

3 p-#VertexCover and p-Card-#Hittingset

We start with two tightly connected problems. Recall that a *hitting set* in a hypergraph $\mathcal{H} = (V, E)$ is a set $S \subseteq V$ such that $S \cap e \neq \emptyset$ for all $e \in E$.

Furthermore, if we regard graphs as hypergraphs, vertex covers are simply a special case of hitting sets.

In the analysis of all algorithms considered here we will use the uniform cost measure. This implies that all basic arithmetic computations can be carried out in constant time.

p-card-#HittingSet
Instance: A hypergraph $\mathcal{H} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* $k + d$ with $d := max_{e \in E}\|e\|$ *Problem:* Compute the number of hitting sets of cardinality k in $\mathcal{H}$

We define $\#hs(\mathcal{H}, k)$ to denote the number of hitting sets of size k in the hypergraph $\mathcal{H}$.

The kernelization of the decision problem p-card-HittingSet given in [17] utilizes the well known *Sunflower Lemma*. Given a hypergraph $\mathcal{H} = (V, F)$, a *sunflower* in $\mathcal{H}$ is a family $\mathbf{S} = \{S_1, \ldots, S_k\} \subseteq F$ of hyperedges such that there is a set $C \subseteq V$ satisfying

$$S_i \cap S_j = C \qquad \text{for all } 1 \leq i < j \leq k \tag{2}$$

C is the *core* of the sunflower $\mathbf{S}$ and for all $i \in [k]$ the set $S_i \setminus C$ is called a *petal* if it is nonempty.

Lemma 3 (Sunflower Lemma). *Let $k, d \in \mathbb{N}$ and be $\mathcal{H} = (V, F)$ a d-uniform hypergraph with more than $(k-1)^d \cdot d!$ hyperedges. Then there is a sunflower S of cardinality k in $\mathcal{H}$.*

The Sunflower Lemma can be applied to counting in a straightforward manner:

Lemma 4. *1. There is an algorithm* FindSunflower *that computes a sunflower S according to the Sunflower Lemma in time $O(d\|\mathcal{H}\|)$.*

2. Let S with $|S| = k+1$ be a sunflower in $\mathcal{H}$ with core C. Define $\mathcal{H}' = (V, E')$ with

$$E' := E \setminus S \cup \{C\}$$

then $\#hs(\mathcal{H}, k) = \#hs(\mathcal{H}', k)$.

The kernelization mapping $K_{Sunflower}$ is given in algorithm 1. Observe that the algorithm deletes vertices that are isolated after the graph has been reduced (line 1). The following Lemma shows how to compute the correct hitting set count from the output of $K_{Sunflower}$.

Lemma 5. *Let $(\mathcal{H}, k)$ be an instance of* p-card-#HittingSet *with $\mathcal{H} = (V, E)$ and let $\mathcal{H}' \leftarrow K_{Sunflower}(\mathcal{H}, k)$ with $\mathcal{H}' = (V', E')$. Define $I := V \setminus V'$, then*

$$\#hs(\mathcal{H}, k) = \textstyle\sum_{i=0}^{k} \#hs(\mathcal{H}', i) \cdot \binom{|I|}{k-i}$$

The proof of this Lemma follows from the easily observable fact that any size k hitting set C satisfies $|C \cap V'| = i$ and $|C \cap I| = k - i$ for some suitable $0 \leq i \leq k$. Conversely, for any size i hitting set C' of $\mathcal{H}'$ and for every set $I' \subseteq I$ of cardinality $|I'| = k - i$ the union $C' \cup I'$ is a size k hitting set of $\mathcal{H}$.

```
   K_Sunflower(H, k) // H = (V, E) a hypergraph, k ∈ ℕ
 1 d ← max_{e∈E}|e|;
   // Begin kernelization
 2 for i ← 1 to d do
 3     E_i ← {e ∈ E : |e| = i };
 4     V_i ← ⋃_{e∈E_i} e;
 5     while FindSunflower((V_i, E_i), k + 1) ≠ (∅, ∅) do
 6         Let (S, C) be the sunflower returned by FindSunflower ;
 7         E_i ← (E_i \ S) ∪ {C};
 8         V_i ← (V_i \ ⋃_{X∈S} X) ∪ C;
 9     end
10 end
   // End kernelization
11 V' ← V_1 ∪ V_2 ∪ ... ∪ V_d;
12 E' ← E_1 ∪ E_2 ∪ ... ∪ E_d;
13 return (V', E');
```

Algorithm 1. The Kernelization algorithm based on the Sunflower Lemma

Note that a run of $K_{Sunflower}(\mathcal{H}, k)$ takes time at most $O(d^2 \cdot |E| \cdot \|\mathcal{H}\|)$ and for the hypergraph $\mathcal{H}' = (V', E')$ the Sunflower Lemma implies $|E'| \leq k^d \cdot d! \cdot d$ and $|V'| \leq k^d \cdot d! \cdot d^2$. Thus we have a kernelization of size $\|\mathcal{H}'\| \leq 2 \cdot k^d \cdot d! \cdot d^2$.

To complete the definition of the counting kernelization, we define a relation Y such that for all instances $(\mathcal{H}, k)$ of p-card-#HittingSet and $C \subseteq V$ we have $((\mathcal{H}, k), C) \in Y$ if and only if C is a hitting set of size k in $\mathcal{H}$. Hence, $w_Y((\mathcal{H}', k))$ can be enumerated in time $h(k)$ for some computable function h and we are done by defining

$$\mu((\mathcal{H}, k), (\mathcal{H}', k), C) := \binom{|I|}{k - |C|}.$$

For Vertex Cover we can easily form a slightly better kernel. Observe that, for graphs, the sunflower kernel implies $|E'| \leq 2k^2$ and $|V'| \leq 4k^2$. A simple but powerful kernelization of p-VertexCover is known as *Buss' Kernelization*. Its idea is very similar to that of of the sunflower kernel:

Lemma 6. *Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$ then:*

1. *Any $v \in V$ with $d(v) > k$ is contained in every k-element vertex cover of $\mathcal{G}$.*
2. *If $\Delta(\mathcal{G}) \leq k$ and $\mathcal{G}$ has a k-element vertex cover, then $|E| \leq k^2$.*

From this we derive a kernel of half the size of the sunflower kernel, as for the kernel $\mathcal{G}' = (V', E')$ the Lemma implies $|E'| \leq k^2$ and hence $|V'| \leq 2k^2$. Note that the counting kernelization according to Buss' Lemma can be build completely analogously to that of the sunflower kernel.

Observe furthermore that we can always combine these kernelizations with fixed parameter algorithms, based on the method of bounded search trees. In this

way, for example, for the p-#VertexCover problem, using the algorithm by Fernau [16], we obtain an algorithm which runs in time $O(1.62^k k^2 + poly(n))$ [26].

4 Crown Rule Reduction for p-#VertexCover

One of the most efficient kernelization techniques known in parameterized algorithm design is the application of the so-called *Crown Rule Reduction* [15]. We will see now, how to apply this kernelization to counting vertex covers. This will, somewhat surprisingly, result in a counting kernelization with size exponential in k. Nevertheless it still may guide the way to counting kernelizations of problems for which no kernel is known at all.

Definition 7. *Let $\mathcal{G} = (V, E)$ be a graph. A* crown *in $\mathcal{G}$ is a bipartite subgraph $\mathcal{C} = (I, N(I), F)$ of $\mathcal{G}$ satisfying three conditions:*

(1) I is an independent set in $\mathcal{G}$ and $N(I)$ is the set of all neighbors of vertices from I in $\mathcal{G}$.
(2) F contains all edges from E that connect vertices in $N(I)$ to vertices in I.
(3) $\mathcal{C}$ has a matching of cardinality $|N(I)|$.

Let $vc(\mathcal{G})$ denote the minimum size of a vertex cover in $\mathcal{G}$.

Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. For the time being, let us assume, that $\mathcal{G}$ contains no isolated vertices. The case of isolated vertices will be considered later. We follow the construction of a crown as given in [17], this construction algorithm produces three different results:

(A) it determines $vc(\mathcal{G}) > k$
(B) $|V| \leq 3k$ holds
(C) a crown $\mathcal{C} = (I, N(I), F)$ on at least $|V| - 3k$ vertices is constructed

In the case of p-VertexCover the crown can simply be deleted. In counting we have to do some additional work to obtain an appropriate kernel.

Applying a crown in a counting algorithm. Let $\mathcal{G} \setminus \mathcal{C}$ be the graph obtained from $\mathcal{G}$ by deleting all vertices in $\mathcal{C}$ and all edges incident to vertices in $\mathcal{C}$.

Let $S_\mathcal{C}$ be a vertex cover of $\mathcal{C}$ and let $\mathcal{G}' = (V', E')$ be the graph obtained from $\mathcal{G} \setminus \mathcal{C}$ by deleting all edges covered by vertices in $S_\mathcal{C}$. One can easily see that $\#vc(\mathcal{G}', k - |S_\mathcal{C}|)$ equals the number of size k vertex covers S in $\mathcal{G}$ with $S \supseteq S_\mathcal{C}$. As $|V'| \leq 3k$, $\#vc(\mathcal{G}', k - |S_\mathcal{C}|)$ can be computed in time depending only on k. Therefore, to show how to compute the number of size k vertex covers in $\mathcal{G}$ it remains to show, how to enumerate the vertex covers of $\mathcal{C}$.

Recall that $|N(I)| < k$ and note that, with respect to the edges in $\mathcal{G}$ (and hence in $\mathcal{C}$ as well), all vertices in I have neighbors only in $N(I)$. Therefore, we can define an equivalency relation with at most 2^k equivalency classes by defining for all $v, w \in I$:

$$v \sim w \;:\Leftrightarrow\; N(v) = N(w) \tag{3}$$

For every $v \in I$ define $[v] := \{w \in I \mid v \sim w\}$, i.e. the equivalence class of v.

Claim 1. Given a vertex cover C of $\mathcal{C}$ such that there is a vertex $y \in N(I) \setminus C$. Let $v \in I$ be a vertex with $y \in N(v)$, then $[v] \subseteq C$.

Proof. Assume that there is a $w \in [v]$ with $w \notin C$. Then, by the definition of $[v]$ there is an uncovered edge $\{y, w\}$ in $\mathcal{C}$. Contradiction.

Consider a vertex cover C of $\mathcal{C}$ and let $S = N(I) \cap C$ and $X = I \cap C$. Note that S and X form a partition of C. Let

$$\mathbf{A}(S) := \{[v] \mid v \in I,\ \exists\, y \in N(I) \setminus S \text{ such that } y \in N(v)\}.$$

and define

$$L(S) := \bigcup_{[v] \in \mathbf{A}(S)} [v] \tag{4}$$

Now, claim 1 implies that $L(S) \subseteq X$, that is $\mathbf{A}(S)$ is the (unique) minimal family of equivalency classes that is necessary such that $S \cup L(S)$ is a vertex cover in $\mathcal{C}$. Hence, we finally have to move from minimal (w.r.t. inclusion) vertex covers to arbitrary vertex covers of $\mathcal{C}$. In fact this means to consider the sets $R := X \setminus L(S)$. As $R \subseteq I \setminus L(S)$, for each $l \in \{0, \ldots, k - |S \cup L(S)|\}$ the number of such sets with $|R| = l$ is exactly

$$\binom{|I \setminus L(S)|}{l}$$

Hence, we are able to define our counting kernelization. The relation Y contains a pair $((\mathcal{G}, \mathcal{C}, k), C)$ iff $\mathcal{C} = (W, F)$ is a subgraph of $\mathcal{G}$ and C is a vertex cover of $\mathcal{G}$ with $|C| \leq k$ such that $C \cap W$ is minimal (w.r.t. inclusion) in $\mathcal{C}$.

For the Kernelization mapping K reconsider the exit conditions (A),(B) and (C) of the crown construction. If (A) $vc(\mathcal{G}) > k$, then $K(\mathcal{G}, k) := (\mathcal{G}_0, \mathcal{C}_0, 0)$ with $\mathcal{G}_0 = (\{a, b\}, \{a, b\})$ and $\mathcal{C}_0$ the empty crown, i.e. trivial negative instance. In case (B) $K(\mathcal{G}, k) := (\mathcal{G}, \mathcal{C}_0, k)$ as $\mathcal{G}$ is small. If (C) holds, the construction algorithm returns a crown $\mathcal{C} = (I, N(I), F)$. Then $K(\mathcal{G}, k) := (\mathcal{G}', \mathcal{C}', k)$ where $\mathcal{C}' = (I', N(I'), F')$ is obtained from $\mathcal{C}$ by keeping one vertex from I per equivalence class and deleting the rest of the vertices in I. $\mathcal{G}' = (V', E')$ is obtained from $\mathcal{G}$ in the same manner. Thus, $|I'| \leq 2^k$, $|N(I)| \leq k$ which implies $|V'| \leq 2^k + 4k$, that is, K is a κ-bounded mapping.

Finally, the mapping μ is defined as follows. Let $(\mathcal{G}, k)$ be the input instance and $\mathcal{C} = (I, N(I), F)$ the crown in $\mathcal{G}$. Let the kernel be $K(\mathcal{G}, k) := (\mathcal{G}', \mathcal{C}', k)$ with $\mathcal{C}' = (I', N(I'), F')$ and $\mathcal{G}'$ defined as in the previous paragraph.

Consider a vertex cover C of $\mathcal{G}'$ with $|C| \leq k$ such that $C \cap (I' \cup N(I'))$ is minimal with respect to the vertices in $\mathcal{C}'$. Define $S' := N(I') \cap C$. We know that $C \cap I'$ is one-to-one with $\mathbf{A}(S')$ as I' contains one representative of each equivalency class, hence $|L(S')| = \sum_{v \in C \cap I'} |[v]|$. Thus defining

$$\mu((\mathcal{G}, k), K(\mathcal{G}, k), C) := \binom{|I \setminus L(S')|}{l}$$

with $l := k - |C \cap N(I')| - |L(S')|$ satisfies our needs, as the binomial coefficient is zero if $l < 0$ or $l > |I \setminus L(S')|$. Obviously, μ is polynomial time computable and the correctness of our kernelization follows from the above considerations.

For the case that the input graph $\mathcal{G}$ contains m isolated vertices, it is easy to see that it suffices to delete them from $\mathcal{G}$ and accommodate m in the computation of the binomial coefficient for the μ function.

5 A Kernelization for p-#UniqueHittingSet

The problem p-#UniqueHittingSet provides us with another nice example of applying well known kernelization techniques to counting problems. The corresponding decision problem is described in [13] (see exercise 3.2.5).

p-#UniqueHittingSet

Instance: A hypergraph $\mathcal{H} = (V, E)$
Parameter: $|E|$
Problem: Compute the number of *unique* hitting sets sets in $\mathcal{H}$ that is all sets $X \subseteq V$ satisfying $\forall\, e \in E : |e \cap X| = 1$

As with the previous problems we will outline the well known solution of the decision problem and show how it can be used to design an efficient algorithm for the counting problem.

Let $\mathcal{H} = (V, E)$ be a p-#UniqueHittingSet instance. Let $k := |E|$. For all $x, y \in V$ we define an equivalency relation by

$$x \sim y =_{def} \forall e \in E :\ x \in e \Leftrightarrow y \in e.$$

Clearly, this relation defines l nonempty equivalency classes for an $l \leq 2^k$. Let A_0 be the equivalency class of all isolated vertices and define $\tilde{\mathbf{A}} := \{A_1, \ldots, A_{l-1}\}$ as the family of all nonempty equivalency classes, except for A_0.

Furthermore, we define $\tilde{\mathcal{H}} := (\tilde{\mathbf{A}}, \tilde{E})$. This hypergraph will play the part of the kernel in our algorithm. Its set of hyperedges $\tilde{E}$ represents the hyperedges in E with respect to the equivalency classes in $\tilde{\mathbf{A}}$. To make this precise, we define two functions. Let $h : V \to \tilde{\mathbf{A}}$ be defined by $h(v) := A \in \tilde{\mathbf{A}}$ s.t. $v \in A$, i.e. we map vertices to their corresponding equivalency classes. A second function $f : 2^V \to 2^{\tilde{\mathbf{A}}}$ defined by $f(\{v_1, \ldots, v_b\}) := \{h(v_1), \ldots, h(v_b)\}$ does so analogously for sets of vertices. Note that h (and f) are undefined for isolated vertices (and sets containing them, respectively). The definition of $\tilde{E}$ is easy now:

$$\tilde{E} := \{f(e) \mid e \in E\}.$$

The following Lemma displays the correctness of our approach.

Lemma 8. *Let $\mathcal{H}, \tilde{\mathcal{H}}$ and $\tilde{\mathbf{A}}$ be defined as above.*
Let $\tilde{\mathbf{U}}$ be the set of all unique hitting sets in $\tilde{\mathcal{H}}$, then the number $\#uhs(\mathcal{H})$ of unique hitting sets in $\mathcal{H}$ satisfies:

$$\#uhs(\mathcal{H}) = \sum_{\tilde{S} \in \tilde{\mathbf{U}}} 2^{|A_0|} \prod_{A \in \tilde{S}} |A| \tag{5}$$

By this Lemma, we can develop a counting kernelization (μ, K). We simply define a relation Y s.t. for every hypergraph $\mathcal{H} = (V, E)$ and $S \subseteq V$, we have $(\mathcal{H}, S) \in Y$ iff S is a unique hitting set in $\mathcal{H}$.

We define $K(\mathcal{H}) := \tilde{\mathcal{H}}$ and for any hitting set $\tilde{S}$ of $\tilde{\mathcal{H}}$:

$$\mu(\mathcal{H}, \tilde{\mathcal{H}}, \tilde{S}) := 2^{|A_0|} \prod_{A \in \tilde{S}} |A|.$$

Obviously, by the construction of $\tilde{H}$ we have $|\tilde{E}| = |E| = k$ and $|\tilde{\mathbf{A}}| \leq 2^k$ and $\tilde{\mathcal{H}}$ can be constructed from $\mathcal{H}$ in polynomial time. Furthermore, μ can be computed in polynomial time, as well. Lemma 8 now implies that (μ, K) is a valid counting kernelization.

6 Characterizing FFPT by Counting Kernelizations

Up to now, we have seen that counting kernelizations can be used to describe certain fpt algorithms for counting problems. Theorem 9 diplays that the notion of counting kernelization is even more general in the sense that it provides a complete characterization of fixed parameter tractable counting problems.

Theorem 9. *Let (F, κ) be a parameterized counting problem. The following are equivalent:*

(1) $(F, \kappa) \in FFPT$
(2) (F, κ) has a counting kernelization.

Note that the direction (2) $\Rightarrow$ (1) is straightforward. The other direction of the proof uses as a main tool the following Lemma, which itself shows the relation between fixed parameter tractable counting problems and K-aware relations.

Lemma 10. *Let $(F, \kappa) \in FFPT$ be a parameterized counting problem over Σ. Let $K : \Sigma^* \to \Sigma^*$ be a κ-bounded polynomial time computable mapping.*

There is a K-aware relation $Y \subseteq \Sigma^* \times \{0,1\}^*$ such that for all $x \in \Sigma^*$:

$$F(K(x)) = |w_Y(K(x))|.$$

7 Concluding Remarks

We have introduced kernelizations of fixed parameter tractable counting problems which provide a characterization of fixed parameter tractability. As a line

of future research we would suggest scrutinizing kernelization techniques towards their applicability to counting problems. Although we demonstrated that, in principle the Crown Rule Reduction is applicable to counting problems, for other problems, which were originally kernelized by the Crown Rule, the kernelizations omit some cases that are trivial in decision problems but highly non-trivial for counting. These include, among others, p-SetSplitting [19], p-DisjointTriangles [20] and p-(n-k)-Coloring [6]. In fact it would not be very surprising if their counting analogs turn out to be $\#W[1]$-hard.

References

1. Susanne Albers and Tomasz Radzik, editors. *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*. Springer, 2004.
2. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.
3. Hans L. Bodlaender, editor. *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, volume 2880 of *Lecture Notes in Computer Science*. Springer, 2003.
4. Hajo Broersma, Matthew Johnson, and Stefan Szeider, editors. *Algorithms and Complexity in Durham 2005 - Proceedings of the First ACiD Workshop, 8-10 July 2005, Durham, UK*, volume 4 of *Texts in Algorithmics*. King's College, London, 2005.
5. Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.
6. Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $o(n^2)$ steps. In *WG*, pages 257–269, 2004.
7. Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.
8. Frank K. H. A. Dehne, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. Greedy localization, iterative compression, modeled crown reductions: New fpt techniques, an improved algorithm for set splitting, and a novel 2k kernelization for vertex cover. In Downey et al. [14], pages 271–280.
9. Frank K. H. A. Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, editors. *Algorithms and Data Structures, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005, Proceedings*, volume 3608 of *Lecture Notes in Computer Science*. Springer, 2005.
10. Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Fixed parameter algorithms for counting and deciding bounded restrictive list h-colorings. In Albers and Radzik [1], pages 275–286.
11. R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.
12. Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.
13. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

14. Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors. *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*. Springer, 2004.
15. Michael R. Fellows. Blow-ups, win/win's, and crown rules: Some new directions in fpt. In Bodlaender [3], pages 1–12.
16. Henning Fernau. Parameterized algorithmics: A graph-theoretic approach. Habilitationsschrift, Universität Tübingen, 2005.
17. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.
18. M. Frick. *Easy Instances for Model Checking*. PhD thesis, Universität Freiburg, 2001.
19. Daniel Lokshtanov and Christian Sloper. Fixed parameter set splitting, linear kernel and improved running time. In Broersma et al. [4], pages 105–113.
20. Luke Mathieson, Elena Prieto, and Peter Shaw. Packing edge disjoint triangles: A parameterized view. In Downey et al. [14], pages 127–137.
21. Rolf Niedermeier. Invitation to fixed-parameter algorithms. Habilitationsschrift, Universität Tübingen, 2002.
22. Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Parameterized counting algorithms for general graph covering problems. In Dehne et al. [9], pages 99–109.
23. Christos H. Papadimitriou. *Computational Complexity*. Pearson, 1994.
24. Peter Rossmanith. (unpublished).
25. Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.
26. M. Thurley. Tractability and intractability of parameterized counting problems. diploma thesis. humboldt universität zu berlin, 2006.
27. Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.
28. Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

Revisiting the Impossibility for Boosting Service Resilience

Xingwu Liu[1], Zhiwei Xu[1], and Juhua Pu[2]

[1] Research Center for Grid and Service Computing, Institute of Computing Technology, Chinese Academy of Sciences
[2] School of Computer Science, BeiHang University
xlw@software.ict.ac.cn, zxu@ict.ac.cn, pujh@buaa.edu.cn

Abstract. An asynchronous distributed system consisting of a collection of processes interacting via accessing shared services or variables. Failure-tolerant computability for such systems is an important issue, but too little attention has been paid to the case where the services themselves can fail. Recently, it's proved that consensus problem can't be $(f+1)$-resiliently solved using a finite number of reliable registers and f-resilient services (failure-aware services must be fully connected). We generalize the result in two dimensions. Firstly, it's shown that the impossibility holds even if infinitely many registers and services are allowed. Secondly, we prove that replacing the reliable registers with reliable shared variables still leave the impossibility to hold, if only failure-oblivious services are allowed.

1 Introduction

1.1 Background

Asynchronous distributed systems consist of a finite collection of processes interacting via some shared mechanisms. The examples of such mechanisms include shared variables, communication channels, atomic objects, and services [1, 2]. A service is itself an arbitrary distributed system with more complicated interface than atomic objects. The major difference between atomic objects and services lies in that a response of an atomic object at a port must exclusively correspond to a previous invocation at the same port, while an invocation to a service at a port may lead to an finite number of responses at any ports, and the service itself may generate responses even without invocations.

Fault-tolerant computability problem is fundamental for asynchronous distributed systems, which explores whether a task can be solved by asynchronous distributed systems, when the components may be subject to failures?

In the related massive literature, consensus tasks [3] are most frequently studied, because they are fundamental in the sense that every atomic object can be implemented from consensus objects and shared read/write variables [4]. [2] also deals with the computability of consensus tasks, proving that there is no $(f+1)$-resilient implementation of consensus objects from canonical f-resilient

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 715–727, 2007.
© Springer-Verlag Berlin Heidelberg 2007

failure-aware services, canonical f-resilient failure-oblivious services, and canonical reliable registers, if each failure-aware service is required to be connected with each process (the requirement is called full connection). [2] is noticeable because it's the first paper and presently the only one studying the computability using fault-prone services, while most of the related work only considers reliable atomic objects.

Note that the results in [2] are based on two assumptions. **Finiteness assumption**: only a finite number of services and registers are included in a system. **Distribution assumption**: canonical registers, which are atomic objects of *read/write* type, are used instead of shared variables. However, systems not satisfying these assumptions are also of significance in the theory of distributed computing. For example, in the well-known Herlihy's hierarchy [4], the universal implementation of an n-process atomic object needs infinitely many n-consensus objects if the n-consensus objects are not augmented with a *reset* operation, and more seriously, the hierarchy can't be maintained if read/write shared memory is not available [5].

1.2 Problem Addressed

The following question naturally arises: can we implement a $(f+1)$-resilient consensus object from infinitely many reliable read/write variables and f-resilient services?

This question can't be answered as a trivial corollary of the results in [2], since no one has proven that a system with shared variables and fault-prone services can be simulated by one with reliable atomic objects and fault-prone services. [6] shows that any shared variable system can be simulated by a shared atomic object system, but it relies on (1) that a process and its user are exclusively enabled, which fails to hold for our case, and (2) all processes in the simulated system interact only via shared variables, while services are also allowed in our systems.

Furthermore, two obstacles exist if we straightforwardly borrow the proof in [2]. Firstly, the proof of [Lemma 5, 2] doesn't carry over, as shown in Subsection 3.2. Secondly, Claim 3 in [Lemma 8, 2] fails if shared reliable read/write variables are used instead of canonical registers, due to the reason in Subsection 4.1.

1.3 Related Work

There is a long line of research on the computability of distributed decision problems in shared object systems. However, much of it focuses on fault-prone processes and assumes that the shared variables and objects are reliable. Well-known examples include the impossibility of set-consensus/renaming using shared read/write variables [7,8], the solvability of set consensus using arbitrary objects under certain conditions [9], and the Herlihy's hierarchy.

Afek et al. [10] first studied the solvability of consensus with faulty shared memory. It's shown that faults do not qualitatively decrease the power of such

primitives as test-and-set and read-modify-write, in that they retain their positions in Herlihy's hierarchy. The failures of a variable are spontaneous modifications of its value, modeled by arbitrary *writes* from the environment of the whole system.

Though Afek et al. have focused on a few types of shared memory, Jayanti et al. [11] studied implementing arbitrary objects with fault-prone base objects. They classify object failures into responsive and non-responsive, and shows that any type of object can be implemented so as to keep reliable when some base objects are responsively faulty, and that any non-responsive fault can't be tolerated in this sense. The faults considerd by Afek et al belong to the responsive class.

Attie et al [2] go a step further by considering implementing objects using general services rather than only shared variables and atomic objects. Another critical difference of [2] from [10, 11] lies in the failure model. In [10,11], a base object fails spontaneously, i.e. independently of the processes; however, [2] models a failure of a service with an input action *fail* from the system environment. Because any client process of the service shares the *fail* action with the service, the failures of a service and its processes are not independent. The main result of Attie et al is that it's impossible to implement a $(f+1)$-resilient consensus object from canonical reliable registers, f-resilient canonical atomic objects, and f-resilient services. When failure-aware services are allowed, full connectivity should be satisfied in order to maintain the impossibility. The result is also the starting point of the present paper.

1.4 Our Contribution

We generalize [2] in two dimensions. We first discard the finiteness assumption, proving that all the results in [2] hold even if an infinite number of services and canonical registers can be in one system. This is achieved by a novel scheduling approach such that each of an infinite collection of tasks has infinitely many opportunities to be scheduled. Then we discard the distribution assumption, showing that using the seemingly *more synchronized* shared read/write variables instead of canonical registers still maintains the impossibility results if failure-aware services are not allowed. For this end, a generalized version of [Theorem 13.7, 6] is proposed, indicating that any distributed system where processes communicate via reliable shared variables and services can be simulated by another one with the shared variables replaced by reliable atomic objects of the corresponding types. This step is itself very interesting, since it frees us from many constraints in constructing or optimizing an asynchronous distributed system.

1.5 Organization

The rest of this paper is organized as follows. Section 2 presents some preliminaries. Section 3 and Section 4 are devoted to the two dimensions of our generalization, respectively. And Section 5 concludes this paper.

2 Preliminaries

We assume the terminology of [2] and [6]. Some concepts are mentioned below. For more detail, please refer to the references.

2.1 Sequential Types, Atomic Objects, and Services

We remind the notion of a "sequential type", in order to describe allowable sequential behavior of atomic objects. It's borrowed from [2]. A sequential type $\Gamma = \langle\ V,\ V_0$, invs, resps, $\delta\ \rangle$ consists of:

- V, a nonempty set of values,
- $V_0 \subseteq V$, a nonempty set of initial values,
- invs, a set of invocations,
- resps, a set of responses, and
- δ, a relation from invs $\times V$ to resps $\times V$ that is total, in the sense that, for every $(a,v)\in$invs$\times$V, there is at least one $(b,\ v')\in$resps$\times V$ such that $((a,v),\ (b,\ v'))\in\delta$.

We sometimes use dot notation, writing $\Gamma.V$, $\Gamma.V_0$, Γ.invs, . . . for the components of Γ.

Example. *Read/write* sequential type: Here, V is a set of "values", $\Gamma.V_0=\Gamma.v_0$, where v_0 is a distinguished element of V , $invs = \{read\}\cup\{write(v) : v\ \ V\}$, $resps=V\cup\{ack\}$, and $\delta= \{((read,\ v),\ (v,\ v)) : v\in V\ \}\cup\{((write(v),\ v'),\ (ack,\ v)): v,\ v'\in V\}$.

Example. Binary consensus sequential type: Here, $V = \{\{0\},\ \{1\},\ \emptyset\}$, $V_0=\{\emptyset\}$, $invs = \{init(v)) : v\in\{0,\ 1\}\}$, $resps=\{decide(v)\colon v\in\{0,\ 1\}\}$, and $\delta=\{((init(v),\emptyset),\ (decide(v),\ \{v\})) \colon v\in V\}\cup\{((init(v),\ \{v'\}),\ (decide(v'),\ v')) \colon v,\ v'\in V\}$.

An atomic object of sequential type Γ is an automaton whose behavior, if serialized in some way, satisfies Γ. An atomic object of read/write type is called a register, and that of binary consensus type is called a consensus object. An atomic object is f-resilient if when no more than f ports fail, all non-failed ports can provide correct behavior. A canonical f-resilient atomic object of type Γ is a special atomic object which describes the allowable concurrent behavior of all f-resilient atomic objects of type Γ. For more about atomic objects, see [2, 6].

A service is a generalized notion of atomic object. Services can be classified into failure-oblivious and failure-aware ones, depending on whether they provide a port with the failure information of other ports. The behavior of a service is also specified by a service type. For more about services, see [2].

In the present paper and [2], both atomic objects and services permit concurrent operations at the same or different endpoints, in the sense that multiple invocations can be issued, without waiting for responses to the previous ones.

2.2 Shared Variable Systems with Services

In [2], all the systems considered consist of processes, reliable registers, and failure-prone services, and have only a finite number of components. In Section 4

of the present paper, we will consider a similar system model which is different in two aspects. Firstly, we use shared reliable variables instead of reliable register. Secondly, an infinite number of services and variables may be included in a system. The system in fact is composed of a shared variable subsystem and a collection of services, with the actions used to communicate among the components hidden.

The shared variable subsystem is modeled as a single I/O automaton. The interface of a process P_i includes invocation and response actions to interact with the external world, input and output actions to invoke and receive response from the services, and an input $fail_i$ to model an unexpected failure. We assume that the $fail_i$ input action affects P_i in such a way that, from that point onward, no output actions or shared variable actions are enabled. It's supposed that in any state, a process P_i always has an action enabled.

A process can access the shared variables only by internal actions, and each action can access at most one shared variable. So, we further distinguish between two different kinds of internal actions: those that involve the shared variables (called shared variable actions) and those don't. If a variable x is of sequential type $\Gamma x = \langle\ V,\ V_0$, invs, resps, $\delta\ \rangle$, each shared variable action accessing it by P_i must be of the form:

Precondition: $p(state_i)$ //p is a predicate of P_i's current state
Effect: $(b,\ x) \leftarrow \delta(a,\ x)$ // a$\in \Gamma x.invs$
$state_i) \leftarrow$ any s such that $(state_i)$, b, $s)$ belongs to g //g is a certain relation

In the complete system, processes interact only via services and variables, services don't communicate directly with one another, and can't access the shared variables. The interface of the complete system consists of all the invocation and response actions of the processes, plus $fail_i$ for every process P_i.

An issue has to be clarified. Composing I/O automata is usually under the following condition: each action is shared by only a finite number of component automata. Otherwise some properties may fail to hold, for example, [Theorems 8.3 and 8.5, 6]. However, in our specification, two services S_1 and S_2 share the input action $fail_i$ if i is an endpoint for both S_1 and S_2, so it's possible that infinitely many services share an input action. Fortunately, whether this phenomenon occurs is irrelevant to the proof of our first main result, and our second main result only uses services such that necessary properties still hold even if infinitely many services share some $fail_i$.

2.3 The Implementation of Consensus Objects

We also consider boosting resilience in implementing f-resilient consensus objects. By the definition of binary consensus type, an n-process f-resilient consensus objects satisfies:

- Agreement. No two processes decide on different values,
- Validity. Any value decided on is the initial value of some process,

– Termination. In every fair execution in which at most f processes fail, any non-faulty process receives an input eventually decides.

3 Impossibility for Infinite Systems

To begin our argument, the idea of [2] has to be briefly reviewed. In this section, a *finite* (*respectively*, *infinite*) *system* will stand for a system with a finite (respectively, infinite) collection of reliable registers and fault-prone services.

3.1 A Brief Review

All the impossibility results of Theorem 1, 10, and 11 in [2] can be restated collectively as the following proposition.

Proposition 1. (f+1)-resilient consensus objects can't be implemented from finitely many canonical reliable registers and canonical f-resilient services if each failure-aware service must be connected to all the processes.

The proof is by contradiction and includes seven lemmas. Assume that there is a finite system C to solve such a problem. In [Lemma 2, 2], it's shown that once a task of C is enabled in the final state of a finite execution, it's enabled from then on until it's applied. This commutability is used in proving [Lemma 5, 2]. [Lemma 3, 2] is trivial, claiming that every finite failure-free input-first execution of C is either bivalent or univalent, based on which [Lemma 4, 2] guarantees the existence of a finite bivalent execution. Then [Lemma 5, 2] shows that from the finite bivalent execution one can construct a hook-like subgraph of the graph $G(C)$ of C. [Lemmas 6 and 7, 2] prove that two univalent finite executions, if similar in some sense, must have the same valence, which leads to a contradiction that $G(C)$ contains no hooks, as stated in [Lemma 8, 2]. So, such a finite system C doesn't exist.

For convenience and without loss of generality, [2] assumes that the processes P_i, $i \in I$, are deterministic automata in the sense that in each state s, there is at most one transition (s, a, s') such that a is non-input action. The services are also assumed to be deterministic in the sense that its type has a single initial value and the transition relations are mappings. It's also assumed that each process has a single task, and always has an action enabled. In this section, we adopt these assumptions, so any failure-free execution of C can be defined by applying a sequence of tasks, one after the other, to the initial state of C.

The above mentioned $G(C)$ is defined as follows.

(1) The vertices of $G(C)$ are the finite failure-free input-first extensions of the finite bivalent execution α_b).

(2) $G(C)$ contains an edge labeled with task e from α to α' provided that $\alpha' = e(\alpha)$, the extension of α with the task e.

By the determinism assumption, for any vertex α of $G(C)$ and any task e, there is at most one edge labeled with e outgoing from α.

The above mentioned hook is a subgraph of $G(C)$ of the form in Fig. 1, where s_1 and s_2 are univalent but have different valence.

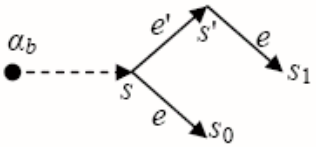


Fig. 1. A hook. (from [2]).

3.2 A Generalization to the Case of Infinite Systems

We prove that the above Proposition 1 also holds even if infinite systems are allowed. Our main result in this section is Theorem 2.

Theorem 2. $(f+1)$-resilient consensus objects can't be implemented from infinitely many canonical reliable registers and canonical f-resilient services if each failure-aware service must be connected to all the processes.

As in [2], we also proceed by contradiction and perform an analysis on the hook structure. So, assume there is an infinite system C to implement $(f+1)$-resilient consensus objects. [Lemmas 2-8, 2], except [Lemma 5, 2], all carry over since they don't care whether the number of canonical reliable registers and canonical f-resilient services is finite or infinite. Thus, if only [Lemma 5, 2] keeps correct, our Theorem 2 holds. However, the proof of [Lemma 5, 2] doesn't carry over, because it extend an execution with an infinite number of tasks in a round-robin fashion. When infinitely many services are used, there are infinitely many tasks, so the round-robin extension must be modified in order to guarantee that the resulting execution, if infinite, is fair. This new version of [Lemma 5, 2] is presented as the following Lemma 3.

In an infinite system, there are infinitely many tasks, so the idea of our modification is to extend the execution in infinite rounds, and to consider only a finite collection of tasks (called candidate task set for this round) for each round. If the candidate task sets are chosen properly, every task eventually has infinite opportunities to be considered.

Before the formal proof, some definitions from [2,6] have to be reminded.

A finite failure-free input-first execution α is defined to be 0-valent if (1) some failure-free extension of α contains a $decide(0)_i$ action, for some process P_i, and (2) no failure-free extension of α contains a $decide(1)_i$ action. The definition of a 1-valent execution is symmetric. A finite failure-free input-first execution α is univalent if it is either 0-valent or 1-valent, and is bivalent if it's not univalent.

Lemma 3. $G(C)$ contains a hook.

Proof. Arbitrarily arrange all the tasks of C into an infinite sequence $\sigma = \sigma_1\sigma_2...\sigma_n...$ such that the process tasks all appear in prefix $\sigma_1\sigma_2...\sigma_n$. Let $\Sigma_i = \sigma_1\sigma_2...\sigma_{n+i}$, $i \geq 1$.

Now starting with a bivalent failure-free α_b), we construct a path π in $G(C)$ round after round.

In round i, for each $i \geq 1$, we consider the tasks in the segment Σ_i from left to right. Suppose that we have reached a bivalent execution α so far, and that task e is the next one in Σ_i that is applicable to α.

[Lemma 2, 2] implies that, for any finite failure-free extension $\alpha' = \alpha \cdot \gamma$ where e is not executed along the suffix γ, e is applicable to α', and hence $e(\alpha')$ is defined. We look for a vertex α' of $G(C)$, reachable from α without following any edge labeled with e, such that $e(\alpha')$ is bivalent. If no such vertex α' exists, the path construction terminates. Otherwise, we proceed to $e(\alpha')$ and then there are two possibilities. (1) If there a task e' after e in Σ_i which is applicable to $e(\alpha')$, consider e' in the next step. (2) If either e is the end of Σ_i or no task after e in Σ_i is applicable to $e(\alpha')$, then go to round $i+1$ and consider the left-most task e' in Σ_{i+1} that is applicable to $e(\alpha')$. In the second case, such e' in Σ_{i+1} always exists since Σ_{i+1} contains all process tasks which are always enabled by assumption.

For the detail of this construction, please refer to [Appendix I, 12].

We claim that π must be finite. Suppose for contradiction that it's infinite. Then infinitely many rounds occur in the construction. Given a task σ_i, it gets a turn to be executed in every round j such that $j \geq \max\{1, i\text{-}n\}$, so it gets infinitely many turns to be executed in π. As a result, π is a failure-free input-first fair execution of C. By the termination condition of consensus object, every process decides in π, which contradicts the facts that every finite prefix of π is bivalent.

The rest of the proof tries to find a hook structure following the finite execution . Since the counterpart of the proof of [Lemma 5, 2] doesn't care whether the collection of tasks is finite or infinite, it carries over. □

4 Impossibility for Systems with Shared Variables

In this section we further generalize Theorem 2 by replacing the canonical reliable registers with shared reliable variables. We show that Theorem 2 still holds if failure-aware services are not allowed. Based on the system model in Subsection 2.2, the following Theorem 3 is obtained.

4.1 Impossibility and the Idea for Its Proof

Theorem 4. $(f+1)$-resilient consensus objects can't be implemented from infinitely many shared reliable *variables* and canonical failure-oblivious f-resilient services.

Suppose for contradiction that there exists such an implementation. Lemmas 2-7, plus Claims 1-5 in the proof of [Lemma 8, 2], except Claim 3, all carry over. However, the proof of Claim 3 relies on the key fact that an invocation to a canonical register from a process P_i only affects the local state of the process and the ith invocation buffer of the register, leaving the register's value unchanged. On the contrary, an access to a shared variable instantaneously modifies its value. As a result, the proof of that Claim 3 can't carry over.

To circumvent proving that claim, we simulate systems with shared variables and canonical services by those with reliable registers and canonical services. Then apply Theorem 2, rather than follow its proof.

4.2 The Construction of a Simulating System

In fact, the simulation presented here is quite generic, because of the following characteristics. Firstly, the services are not necessarily canonical ones. Secondly, there is no constraint on the resilience of the services. Thirdly, the reliable atomic objects are required to be canonical. Fourthly and lastly, unlike [Theorem 13.7, 6], the existence of turn functions are not required.

However, some constraints on services are still needed. Arbitrarily chose a segment $\gamma = \alpha \cdot \beta$ from a (fair) trace of a service S. (1) If α is a response and β is an invocation, then replacing γ with $\beta \cdot \alpha$ still results in a (fair) trace of S. (2) If α and β are responses at different endpoints, then replacing γ with $\beta \cdot \alpha$ still results in a (fair) trace of S. (3) If α=$fail_i$ for some endpoint i of S, and β is an arbitrary external action of S (not an invocation from endpoint i), then replacing γ with $\beta \cdot \alpha$ still results in a (fair) trace of S. (4) If α=$fail_i$, there is an (fair) execution e of S such that τ=trace(e) and either α is the first action or immediately follows an external action in e.

These constraints aren't so restrictive, for example, any canonical failure-oblivious service satisfies all of them.

Now consider a shared variable system C with services, as specified in Subsection 2.2. The services are supposed to satisfy the above constraints. There are two technical assumptions on the processes. (1) In every state, each process has an action enabled. (2) There is a single task for each process, containing all non-input actions of the process. The two assumptions don't reduce the generality of our simulation (as in the sense of Theorem 5), since there must be such a system that simulates C.

Let V be the set of shared variables of C, and associate each v in V with a compatible reliable atomic object o_v. Atomic object o and variable v are said to be compatible if both of the following conditions hold. (1) o and v are of the same sequential type Γ_v. (2) A process P of C can directly access v if and only if P is in the endpoint set of o.

Let O=$\{o_v | v \in V\}$. We construct a system $T(C)$ which intuitively, is derived from C by replacing each $v \in V$ with o_v. The aim of $T(C)$ is to simulate C in some sense. The construction of $T(C)$ is specified as follows.

A shared variable system with services can be described as $\langle \mathfrak{P}, \mathfrak{I}, V \rangle$, where $\mathfrak{P}, \mathfrak{I}, and\, V$ are its set of processes, services, and shared variables, respectively. If C= $\langle \{P_1, P_2, ...P_n\}, \mathfrak{I}, V \rangle$, then $T(C)$ =$\langle \{Q_1, Q_2, ...Q_n\}, \mathfrak{I} \cup O \rangle$. The processes Q_i is almost the same as P_i, but has the following difference in states, signature, and transition relation.

Q_i includes all the state components of P_i, plus seven more. (1) A binary **semaphore**, whether it's 1 indicates whether process Q_i is waiting for the response from an atomic object in O. (2) An 1-length queue **pending-invo**, storing a pending invocation to an object in O. (3) An 1-length queue **vari-resp**, storing the response just received from an object in O. (4) A variable **local-tran**, recording how to transit the local state of Q_i once the response from an object in O is received. (5) An infinite first-in-first-out queue **resp-buffer**, holding the responses sent by services in $\mathfrak{I}$ but not yet *processed* by Q_i. (6) A binary **flag**,

whether it's 1 implies whether it's Q_i's turn to process a response in the buffer. (7) A Boolean **failed**, whether it's True implies whether $fail_i$ has occurred. Initially, semaphore=0, pending-invo is empty, local-tran is arbitrary, vari-resp is empty, resp-buffer is empty, and flag=1. The introduction of the flag is a little technical, in order to preclude that Q_i keeps busy with only receiving and processing service responses in a fair execution.

Let Φ_i denote the set of internal actions of P_i that access a shared variables. Given an arbitrary $c \in \Phi_i$, introduce an action, denoted by $l(c)$, which is to start up c. Introduce another action $stub_i$ to perform the local part of all c. For each input action b of P_i, introduce an action $l(b)$, which is intended to *process* b, i.e. to change the local state of Q_i as b does that of P_i; the original b is preserved, while it means that P_i only receives the input. Introduce an internal action $live_i$, to keep Q_i live even it fails just after some $l(c)$. Let the input signature, output signature, and internal signature of P_i be In, Out, and Int respectively, and those of Q_i be In', Out', and Int' respectively. Then $In'=In \cup \{b|$ b is an output action at endpoint i of an atomic object in $O\}$, $Out'=Out \cup a | a \neq fail_i \wedge$ a is an input action at endpoint i of an atomic object in $O\}$, and $Int'=(\text{Int-}\Phi_i) \cup \{l(c)| c \in \Phi_i\} \cup \{l(b)|b \in In \wedge b \neq fail_i\} \cup \{stub_i\}$.

To give an intuition on the transition relation of Q_i, we sketch here the idea of using $T(C)$ to simulate C.

Firstly, all the output actions and internal actions except those in Φ_i are simulated directly, with only their preconditions changed to involve the new state components.

Secondly, an action $c \in \Phi_i$ that accesses a shared variable is simulated by four steps of Q_i. The first step $l(c)$ starts up the simulation of c, enqueuing *pending-invo* with the invocation to v involved in c, and storing into *local-tran* the relation about how to change local state as in c. The second step invokes o_v as is hinted by *pending-invo*, and then dequeues *pending-invo*. The third step enqueues *vari-resp* with the response from o_v. The forth step $stub_i$ changes the local state according to *vari-resp* and *local-tran*, and dequeues *vari-resp*. The four steps are collectively called a complete simulation of c, $l(c)$ is celled the initialization, and $stub_i$ the finalization.

The four steps must be executed one after another without interruption, so other actions, even the inputs from services and external world, must be temporarily disabled. As a result, on the one hand, $l(c)$ sets semaphore to 1, $stub_i$ resets it to 0, the other actions keep it unchanged and most of them are enabled only if it's 0. On the other hand, each input action b$\neq fail_i$ of P_i is simulated in two steps, one (also named b in Q_i) to buffer the response, and the other (i.e. $l(b)$) to update the local state using the response. b and $l(b)$ don't have to be executed consecutively, and in fact, $l(b)$ is always done between some $stub_i$ and the next $l(c)$.

There are two technical maneuvers. On the one hand, in order for the simulation to preserve fairness of traces, it must be precluded that Q_i indefinitely performs only b and $l(b)$ for inputs b and ignores real workload. So, each $l(b)$ sets flag to 0 and is enabled only if flag=1, and flag can be reset to 1 only by

other input actions. On the other hand, to keep Q_i live, when it fails during the simulation of a shared variable action, semaphore must be reset to 0.

Please refer to [Appendix II, 12] for the formal definition of $T(C)$.

4.3 A Property of $T(C)$

If a system S is obtained by composing a collection of component automata and then hiding a set of actions, denote by $R(S)$ the system where the set of actions are not hidden.

Lemma 5. For $R(T(C))$ and $R(C)$, Theorems 8.2, 8.3, 8.5, and 8.6 in [6] hold, though each $fail_i$ action may be shared by infinitely many components. □

This guarantees that under some conditions, the (fair) traces/executions of the component automata can be pasted into (fair) traces/executions of the complete system.

Lemma 6. $T(C)$ simulates C in the following sense.

(1) They have the same interface,

(2) Any (fair) trace of $T(C)$ is a also a (fair) trace of C, up to a permutation which preserves both the order of invocations and that of responses at each port, and preserves the input-covering property. A trace is input-covering if at each port, an invocation precedes all responses.

Remark of the proof: The basic idea is similar to the proof of [Theorem 13.7, 6], but there are two key differences. Firstly, the existence of services, plus the non-existence of turn functions, makes it possible that a process Q_i) receives inputs (including responses) during its simulation of a variable access of P_i. Secondly, inputs sent to Q_i) are buffered and not necessarily processed immediately, so it's possible that Q_i) receives inputs or performs locally controlled actions during its simulation of an input to P_i. As a result, that proof can't carry over. Our proof of Lemma 6 is elaborated in [Appendix III, 12]. □

Suppose the above C is an implementation of n-process consensus object from shared variables and canonical f-resilient failure-oblivious services. A trace of C is said to be input-first if each P_i begins with an init() action, and has no other init() actions.

Corollary 7. There is a fair input-first trace of C where no more than $f+1$ processes fail, and which doesn't satisfy the three conditions of consensus.

Proof: Assume for contradiction that this is not the case. In the construction of $T(C)$, if each o_v is a canonical reliable register, then $T(C)$ is a system with canonical reliable registers and f-resilient failure-oblivious services. [2] in fact proves that there is a fair input-first trace α of C where no more than $f+1$ processes fail, and which doesn't satisfy the three conditions of consensus. By Theorem 5, there is a fair trace β of C which is the same as α up to a certain permutation. The property of the permutation guarantees that is also input-first, and that the projection of β to each port of C is the same as that of α

to each port of $T(C)$. As a result, no more than $f+1$ processes fail in β but β doesn't satisfy the three conditions of consensus. A contradiction. □

Corollary 6 immediately leads to Theorem 4.

5 Conclusion

We have generalized the results of [2] in two dimensions. First, we show that all the impossibility results still hold even if infinitely many reliable registers and fault-prone services are allowed. Second, we show that even if the system is strengthened by replacing canonical reliable registers with shared variables, the impossibility still holds in the case where canonical failure-oblivious services are used. To prove the second result, we in some degree generalize [Theorem 13.7, 6] by discarding the requirement of turn functions, and by extending shared variable systems to systems having both shared variables and failure-oblivious services. Our work under way is to extend our second result to the case of failure-aware services.

Acknowledgment

The work is supported by China's Natural Science Foundation (60603004, 60403023).

References

1. S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 33(2):51-59, June 2002.
2. P. Attie, et al. The Impossibility of Boosting Distributed Service Resilience (Extended abstract). In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS2005), 39-48. The full version is available at http://theory.lcs.mit.edu/tds/papers/Attie/boosting-tr.ps
3. M. Fischer, et al. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(3):374-382, April 1985.
4. M. Herlihy. Wait-free synchronization. ACM Transactions on Programming Languages and Systems, 13(1):124-149, January 1991.
5. P. Jayanti. On the robustness of Herlihy's hierarchy. In Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC1993), 145-157.
6. N. Lynch. Distributed Algorithms. Morgan Kaufmann Publishers, 1996.
7. M. Herlihy and N. Shavit. The topological structure of asynchronous computability. Journal of the ACM, 46(6):858-923, November 1999.
8. M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM Journal on Computing, 29:1449-1483, March 2000.
9. M. Herlihy and S. Rajsbaum. Set Consensus Using Arbitrary Objects. In Proceedings of 13th Annual ACM Symposium on Principles of Distributed Computing (PODC 1994), pp. 324-333.

10. Y. Afek, et al. Computing with Faulty Shared Memory. In Proceedings of 13th Annual ACM Symposium on Principles of Distributed Computing (PODC 1992), pp. 47-58.
11. P. Jayanti, et al. Fault-Tolerant Wait-Free Shared Objects. Journal of the ACM, 45(3): 451-500, May 1998.
12. X. Liu, et al. Revisiting the Impossibility for Boosting Service Resilience. Technical report, January 1998. Available at http://blog.software.ict.ac.cn/xliu/files/2007/03/RevisitingtheImpossibilityfor BoostingServiceResilience-full.pdf

An Approximation Algorithm to the k-Steiner Forest Problem

Peng Zhang*

State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, P.O.Box 8718, Beijing, 100080, China
Graduate University of Chinese Academy of Sciences, Beijing, China
zhangpeng04@iscas.cn

Abstract. Given a graph G, an integer k, and demands set $D = \{(s_1, t_1), \ldots, (s_l, t_l)\}$, the k-Steiner Forest problem finds a forest in graph G to connect at least k demands in D such that the cost of the forest is minimized. This problem is proposed by Hajiaghayi and Jain in SODA'06. Thereafter, using Lagrangian relaxation technique, Segev et al. give the first approximation algorithm to this problem in ESA'06, with performance ratio $O(\min\{n^{2/3}, \sqrt{l}\} \log l)$. We give a new approximation algorithm to this problem with performance ratio $O(\min\{n^{2/3}, \sqrt{l}\} \log k)$ via greedy approach, improving the previously best known ratio in the literature.

1 Introduction

Given a graph $G = (V, E)$ with costs on edges, an integer $k > 0$, and a demand set $D = \{(s_1, t_1), \ldots, (s_l, t_l)\}$, where each demand (a.k.a. source-sink pair) is distinct but vertices in different demands are not required to be distinct, the k-Steiner Forest problem asks to find a forest in graph G to connect at least k demands in D, such that the cost of the forest is minimized. The k-Steiner Forest prolem is proposed by Hajiaghayi and Jain [8] when they studied the prize-collecting generalized Steiner tree problem.

If we remove the given parameter k in k-Steiner Forest, then we get the classical Steiner Forest problem. The currently best approximation ratio for Steiner Forest is $2(1-1/l)$, due to Agrawal, Klein and Ravi [1] and Goemans and Williamson [5]. On the other hand, k-Steiner Forest also captures the classical rooted k-MST and rooted k-Steiner Tree problems. If in k-Steiner Forest let $D = \{(r, v) \colon \forall v \in V - \{r\}\}$ where r is the given vertex used as root, then k-Steiner Forest reduces to the rooted k-MST problem. Similarly, if in k-Steiner Forest let $D = \{(r, v) \colon \forall v \in R - \{r\}\}$ where R is the set of vertices required to be connected and r is the given vertex used as root, then k-Steiner Forest reduces to the rooted k-Steiner Tree problem. Garg [4] showed that in fact the unrooted k-MST problem is equivalent to the rooted

* Supported by NSFC grants No. 60325206 and No. 60310213. This work is part of the author's Ph.D. thesis prepared at Institute of Software, Chinese Academy of Sciences under the supervision of Prof. Angsheng Li.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 728–737, 2007.
© Springer-Verlag Berlin Heidelberg 2007

k-MST problem, and gave a 2-approximation algorithm to k-MST, which is also the currently best known approximation to k-MST. Both k-MST and k-Steiner Tree have been studied using the Lagrangian relaxation technique by Chudak, Roughgarden and Williamson [2].

Although the basic Steiner Forest problem [1,5] and the prize-collecting Steiner Forest problem [8] (which is called the prize-collecting generalized Steiner Tree problem therein) can be well approximated, it is difficult to approximate k-Steiner Forest. Hajiaghayi and Jain [8] proved that if k-Steiner Forest can be approximated within α, then the Densest k-Subgraph problem can be approximated within α^2. On the other hand, the best known performance ratio for Densest k-Subgraph is $O(n^{1/3-\epsilon})$ for some small $\epsilon > 0$ [3]. It seems that obtaining performance ratio better than $O(n^{1/6-\epsilon})$ for k-Steiner Forest requires new ideas.

Segev et al. [10] gave the first non-trivial approximation algorithm to k-Steiner Forest,with performance ratio $O(\min\{n^{2/3}, \sqrt{l}\}\log l)$. Their approach is the Lagrangian relaxation technique. In fact, since the seminar work of Jain and Vazirani [7] for the Facility Location and the k-Median problems, in which the Lagrangian relaxation technique was introduced to the design and analysis of approximation algorithm for the first time, the study on approximating the prize-collecting version and k-version of many optimization problems has attracted more attention (see, for instance, [2,7,9]).

1.1 Our Results and Techniques

We give a simple greedy approximation algorithm to this problem with performance ratio $O(\min\{n^{2/3}, \sqrt{l}\}\log k)$, improving the previously best known ratio $O(\min\{n^{2/3}, \sqrt{l}\}\log l)$ [10]. The optimal solution to k-Steiner Forest must consist of several disjoint trees, and thus is a forest. Intuitively, a "good" tree in a solution to k-Steiner Forest connects many demands by as small as possible cost. In other words, to find a cost-effective tree connecting some demands is a core problem to k-Steiner Forest. The cost-effectiveness of a tree in [10] (which is referred to as the density of a tree therein) is defined as the ratio of the cost of the tree and the number of demands connected by this tree. Segev et al. gave a constructive method to find a tree with cost-effectiveness not much more than that of the most cost-effective tree, meanwhile with cost not exceeded Δ much more, where Δ is a given budget and is guessed by means of exhaustive searching such that $OPT \leq \Delta \leq 2OPT$. Then they gave a greedy prize-collecting algorithm for the prize-collecting Steiner Forest problem. Since the prize-collecting algorithm possesses the so-called Lagrangian Multiplier Preserving [6] property, k-Steiner Forest can be solved by reducing to prize-collecting Steiner Forest in the framework of Lagrangian relaxation.

We consider k-Steiner Forest in the framework of Set k-Cover, which is the k-version of the Set Cover problem. Set k-Cover finds a sub-family of the given set family $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ to cover at least k elements in the given grounding set $U = \{e_1, e_2, \ldots, e_n\}$. Slavík [11] gave a greedy approximation algorithm for Set k-Cover with performance ratio H_k, where $H_k = 1 + \frac{1}{2} + \frac{1}{3} +$

$\cdots + \frac{1}{k}$ is the k-th harmonic number. The greedy algorithm for Set k-Cover finds the most cost-effective set in each iteration to cover some elements still remaining in U, until at least k elements are covered. In [11] the cost-effectiveness of a set S is carefully defined as the ratio of the cost of S and the number of elements validly covered by S. Given k' being the number of elements that need to be covered currently, the number of elements validly covered by S is k' if S covers more than k' new elements, and is the number of new elements covered by S otherwise.

We first prove a coupling lemma which shows that the greedy algorithm coupled with an α-approximation algorithm for finding a cost-effective covering set gives an αH_k-approximation to the Set k-Cover-like problem. This is very useful for the problems that can be casted in the framework of Set k-Cover but it is not known how to find the most cost-effective covering set in polynomial time. Obviously k-Steiner Forest can be casted in the framework of Set k-Cover. We define the cost-effectiveness of a covering tree T as the ratio of the cost of T and the numbers validly covered by T, the same as that in [11]. Our newly defined cost-effectiveness of covering tree is different from that in [10]. Then, by implanting the newly defined cost-effectiveness into the constructive method proposed in [10], we give a polynomial time α-approximation algorithm to find a cost-effective covering tree. This eventually leads to an αH_k-approximation algorithm to k-Steiner Forest, where $\alpha = O(\min\{n^{2/3}, \sqrt{l}\})$. Our global approach is greedy, which is completely different from that (the Lagrangian relaxation technique) of [10]. Although our algorithm relies on the constructive method proposed therein, we would like to point out that the constructive method armed with our cost-effectiveness of covering tree and the coupling lemma are key to the success of our algorithm. Our algorithm is of two nested loops and is simpler and faster, while the algorithm in [10] has three nested loops. More importantly, our method gives better performance ratio to the problem k-Steiner Forest. We believe that our method is of independent interest and may have more applications.

2 The Coupling Greedy Algorithm for Set k-Cover

Slavík [11] proposed the greedy algorithm for the problem Set k-Cover, which is denoted by $\mathcal{A}$ in our setting. Given the grounding set $U = \{e_1, e_2, \ldots, e_n\}$ and set family $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ with each set $S \in \mathcal{S}$ having cost $c(S)$, the algorithm $\mathcal{A}$ repeatedly finds the most cost-effective set S_j in each iteration j, until the found sets cover at least k elements in U. Denote by C_j the set of covered elements at the beginning of iteration j. For a set $S \in \mathcal{S}$, define $\text{new}(S) = |S \backslash C_j|$, that is, $\text{new}(S)$ is the number of elements newly covered by set S. Suppose that at the beginning of iteration j the number of elements needs to be covered is k'. Then the cost-effectiveness of S is defined by $\text{cost-ef}_v(S) = \frac{c(S)}{\text{val}(S)}$, where $\text{val}(S) = \min\{k', \text{new}(S)\}$ is the number of elements validly covered by S. Note that in the traditional greedy algorithm for Set Cover, the cost-effectiveness of a set S is defined by $\text{cost-ef}_n(S) = \frac{c(S)}{\text{new}(S)}$. We remind the readers that the

function new$(\cdot)$ and val$(\cdot)$ are defined with respect to the set of elements currently covered by $\mathcal{A}$.

If the most cost-effective set S can be found in polynomial time, then algorithm $\mathcal{A}$ gives an H_k-approximation. There are many problems which can be casted in the framework of Set k-Cover, but for these problems the most cost-effective set may not be found (or do not know how to find) in polynomial time, usually due to that the family of sets is of exponential size. This motivates us to couple the greedy algorithm $\mathcal{A}$ with an approximation algorithm to find the most cost-effective covering set.

Lemma 1 (The coupling lemma). *If the greedy algorithm $\mathcal{A}$ for* Set k-Cover *is coupled with an algorithm $\mathcal{H}$, where $\mathcal{H}$ in polynomial time finds an α-approximation to the most cost-effective covering set S_j^* in each iteration j of $\mathcal{A}$, then $\mathcal{A}$ outputs an αH_k-approximation to the problem* Set k-Cover *in polynomial time.*

Proof. Without loss of generality, suppose that by reordering the elements in U, the elements covered by algorithm $\mathcal{A}$ in the executing process of $\mathcal{A}$ are $e_1, e_2, \ldots, e_{\hat{k}}$, breaking ties arbitrarily. Recall that $\hat{k} \geq k$. Before giving the detailed analysis, we first introduce some notations. Denote by OPT the optimum of Set k-Cover, and let $O = \{O_1, \ldots, O_r\}$ be an optimal solution. Denote by A_j the set found by algorithm $\mathcal{H}$ in the j-th iteration of $\mathcal{A}$. For each element e_i covered by A_j, define $\text{price}(i) = \frac{c(A_j)}{\text{val}(A_j)}$. Then we know that $A(I) = \sum_j c(A_j) = \sum_{i=1}^{k} \text{price}(i)$, where $A(I)$ is the value of the approximate solution found by $\mathcal{A}$.

We argue that for every $1 \leq i \leq k$, $\text{price}(i) \leq \frac{\alpha}{k-i+1} OPT$. This will give the performance ratio αH_k for the coupled greedy algorithm $\mathcal{A}$. Recall that k' is the number of elements that need to be covered at the beginning of iteration j. For clarity, let $a_j = \text{val}(A_j)$. Since the prices are the same as that for all elements $e_{k-k'+1}, \ldots, e_{k-k'+a_j}$, and $\frac{1}{k'} \leq \frac{1}{k-i+1}$ for each $k-k'+1 \leq i \leq k-k'+a_j$, we only need to show that $\text{price}(i) \leq \frac{\alpha}{k'} OPT$, where $i = k - k' + 1$ is the "first" element covered by A_j. We already have that $\text{price}(i) = \frac{c(A_j)}{\text{val}(A_j)} \leq \alpha \cdot \frac{c(S_j^*)}{\text{val}(S_j^*)}$ because A_j is an α-approximation to S_j^*. By the optimality of S_j^*, $\frac{c(S_j^*)}{\text{val}(S_j^*)} \leq \frac{c(O_u)}{\text{val}(O_u)}$ for each set $O_u \in O$ satisfying that $\text{val}(O_u) \neq 0$. So we get

$$\frac{c(S_j^*)}{\text{val}(S_j^*)} \leq \frac{\sum_{O_u \in O:\ \text{val}(O_u) \neq 0} c(O_u)}{\sum_{O_u \in O:\ \text{val}(O_u) \neq 0} \text{val}(O_u)} \leq \frac{OPT}{k'},$$

where the last inequality holds since $\sum_{O_u \in O:\ c(O_u) \neq 0} \text{val}(O_u) \leq OPT$, and at the beginning of iteration j, there are at least k' elements in $(\bigcup_{O_u \in O} O_u) \setminus C_j$, resulting in $\sum_{O_u \in O:\ \text{val}(O_u) \neq 0} \text{val}(O_u) \geq |(\bigcup_{O_u \in O} O_u) \setminus C_j| \geq k'$. The proof is completed. □

We give two remarks on the coupling lemma. Firstly, the function val$(\cdot)$ plays an important role in the proof of the coupling lemma, since we define price(i) =

$\frac{c(A_j)}{\text{val}(A_j)}$ for every element i covered by A_j. Secondly, for the problems that can be casted into the framework of SET k-COVER, the coupling lemma gives a general approach to obtaining an αH_k-approximation to the problems.

3 Approximating k-Steiner Forest

3.1 Find Cost-Effective Covering Tree

Then we turn back to the problem k-STEINER FOREST. k-STEINER FOREST can be viewed as a SET k-COVER-like problem in which the grounding set $U = \{d_1, d_2, \ldots, d_l\}$ where each $d_i = (s_i, t_i)$ denotes a demand, and the set family $\mathcal{S}$ is the collection of all edges set E_j such that the edges in E_j form a tree in G. Obviously $\mathcal{S}$ has exponential size.

In the framework of SET k-COVER for k-STEINER FOREST, one of the key steps is to find the most cost-effective covering tree. Since there are exponential trees in G, it is obvious that one can not hope to find the most cost-effective covering tree by enumerating them one by one. Denote by F_j the collection of trees selected by the algorithm at time j (actually, at the beginning of iteration j), and by R_j the set of demands not yet connected at time j. Let $C_j = D \setminus R_j$ be the set of demands already connected by the algorithm at this point. For succinctness, when the time j is clear in the context, F_j, C_j, R_j are also written as F, C, R, respectively. For a tree T in G, denote by $\text{dem}(T)$ the number of demands connected by T. Similarly to that in the setting of SET k-COVER, we define $\text{new}(T)$ to be the number of demands in R connected by T. Since the goal is to connect at least k demands in D, given the already connected demands set C and the selected tree collection F, adding T to F may connect more than k demands. The quantity exceeding k is useless to the goal. Under this consideration, the function $\text{val}(\cdot)$ is defined by

$$\text{val}(T) = \min\{k', \text{new}(T)\},$$

where $k' = k - |C|$. That is, $\text{val}(T)$ is the number of demands validly connected by T. Then the cost-effectiveness of tree T is defined by

$$\text{cost-ef}_v(T) = \frac{c(T)}{\text{val}(T)}.$$

The problem of finding the most cost-effective covering tree is to find a tree T such that $\text{cost-ef}_v(T)$ is minimized over all possible trees in G. Similarly, we also define the cost-effectiveness of a tree T with respect to the function $\text{dem}(\cdot)$ as

$$\text{cost-ef}_d(T) = \frac{c(T)}{\text{dem}(T)}.$$

Given the unconnected demand set R, suppose that T^* is the most cost-effective covering tree. For clarity, T^* is also called the optimal covering tree.

Segev et al. [10] gave a constructive method to find a covering tree, which is approximately optimal with respect to the function cost-ef$_d(\cdot)$ (which is referred to as the density function therein). We modify their method by using the new idea to make it fit our settings, that is, to find a covering tree which is approximately optimal with respect to the function cost-ef$_v(\cdot)$. Before giving the algorithm, we first prove Lemma 2. We state the rooted quota-MST problem here, which is used in the proof of Lemma 2. Given a graph G with costs on edges and each vertex $v \in V(G)$ having profit$(v) \in \mathbb{Z}^+$, a vertex $r \in V(G)$, and a nonnegative quota Q, the rooted quota-MST problem asks to find a spanning tree $\widetilde{T}$ with minimized cost satisfying $\sum_{v \in V(\widetilde{T})} \text{profit}(v) \geq Q$. If each profit$(v)$ is polynomial bounded, this problem can be reduced to the k-MST problem and thus can be approximated within factor 2 by the result of [4]. In Lemma 2, T^* is an optimal covering tree and is not known in advance.

Lemma 2. *Given $q = \text{val}(T^*)$, an arbitrary vertex $r \in T^*$, and an arbitrary tree T that contains r, then an augmented tree $T^+ \supseteq T$ can be constructed in polynomial time such that at least one of the two properties (1) $|V(T^+)| \geq |V(T)| + \sqrt{q}/2$ and (2) $\text{val}(T^+) \geq 3q/8$ holds.*

Proof. Suppose that $\text{new}(T) \geq q/2$. If $\text{new}(T) \leq k'$, then we have $\text{val}(T) = \min\{k', \text{new}(T)\} = \text{new}(T) \geq q/2$. If $\text{new}(T) > k'$, since $q = \text{val}(T^*) = \min\{k', \text{new}(T^*)\} \leq k'$, we know that $\text{val}(T) = \min\{k', \text{new}(T)\} = k' \geq q$. So T itself is a tree satisfying property (2) in the lemma and we are done.

So in the following we suppose that $\text{new}(T) < q/2$.

If $|V(T^*) \setminus V(T)| \geq \sqrt{q}/2$ then one can construct a quota spanning tree $\widetilde{T}$ rooted at r by approximating the rooted quota-MST instance on G, in which profit$(v) = 1$ for all $v \in V(G) \setminus V(T)$, profit$(v) = 0$ for all vertices in $V(T)$, and the quota is $\sqrt{q}/2$. Now $T^+ = T \cup \widetilde{T}$ is a feasible tree satisfying the property (1) in the lemma. Such a tree $\widetilde{T}$ must exist, since T^* itself is a feasible quota spanning tree rooted at r.

Otherwise we have $|V(T^*) \setminus V(T)| < \sqrt{q}/2$. Once again, a quota spanning tree $\widetilde{T}$ rooted at r can be constructed by approximating the rooted quota-MST instance on G, in which profit(v) for all $v \in V(G) \setminus V(T)$ is equal to the number of demands in R consisting of v and an additional vertex from T, all vertices in $V(T)$ have zero profit, and the quota is $3q/8$.

We argue that such a spanning tree $\widetilde{T}$ must exist. Denote by $D(T)$ the set of demands covered by a tree T. For $0 \leq j \leq 2$, define set $N_j = \{(s_i, t_i) \in R \cap D(T^*) \colon |\{s_i, t_i\} \cap V(T)| = j\}$, that is, N_j is just the set of demands newly covered by T^* with exactly j endpoints in $V(T)$. Since even if all the vertices in $V(T^*) \setminus V(T)$ form demands newly covered by T^*, there are at most $\binom{|V(T^*) \setminus V(T)|}{2} \leq \binom{\lfloor \sqrt{q}/2 \rfloor}{2} \leq q/8$ new demands entirely in $V(T^*) \setminus V(T)$, eventually we have that $|N_0| \leq q/8$. Notice that $\text{new}(T^*) \geq \min\{k', \text{new}(T^*)\} = \text{val}(T^*) = q$ and $|N_2| \leq \text{new}(T) < q/2$. So we have

$$\begin{aligned} |N_1| &= \text{new}(T^*) - |N_0| - |N_2| \\ &\geq q - |N_0| - |N_2| \\ &\geq q - \frac{q}{8} - \frac{q}{2} \\ &= \frac{3q}{8}. \end{aligned}$$

Since $\sum_{v \in V(T^*) \setminus V(T)} \text{profit}(v) \geq |N_1| \geq 3q/8$, T^* itself is a feasible tree to the rooted quota-MST instance.

Now we get a tree $T^+ = T \cup \widetilde{T}$ with $\text{new}(T^+) \geq 3q/8$. Again, if $\text{new}(T^+) \leq k'$ then we have $\text{val}(T^+) = \min\{k', \text{new}(T^+)\} = \text{new}(T^+) \geq 3q/8$. If $\text{new}(T^+) > k'$ then we have $\text{val}(T^+) = \min\{k', \text{new}(T^+)\} = k' \geq q$ since $q = \text{val}(T^*) = \min\{k', \text{new}(T^*)\} \leq k'$. So, we always have that $\text{val}(T^+) \geq 3q/8$ and hence T^+ satisfies the property (2) in the lemma.

The lemma follows since we can solve the two instances separately and preferably pick the second tree as output when both are feasible, although T^* is unknown to us. □

We shall point out that although in the proof of Lemma 2 we adapt the constructive method in [10], our result $\text{val}(T^+) \geq 3q/8$ in Lemma 2 is stronger than that in [10] because of the newly defined sets N_j's, which is crucial to the success of our algorithm.

Algorithm $\mathcal{B}$

input: Graph G, integer $k' > 0$, and the uncovered demands set R.
output: A tree T covering some demands in R.

(denote by T^* an optimal covering tree)
1. guess $q = \text{val}(T^*)$ from $\{1, \ldots, k'\}$ and an arbitrary vertex $r \in T^*$ from $V(G)$
2. **if** $q < n^{2/3}$ **then return** the shortest path connecting any demand in R and **stop**
   (else we have that $q \geq n^{2/3}$)
3. **let** $T \leftarrow \{r\}$
4. repeatedly extend T by Lemma 2 until $\text{val}(T) \geq 3q/8$
5. **return** T

For algorithm $\mathcal{B}$ we have Lemma 3.

Lemma 3. *In polynomial time, algorithm $\mathcal{B}$ finds a covering tree T satisfying that cost-ef$_v(T) \leq \alpha \cdot$ cost-ef$_v(T^*)$, where $\alpha = O(n^{2/3})$ and T^* is an optimal covering tree with respect to the function cost-ef$_v(\cdot)$.*

Proof. Suppose that algorithm $\mathcal{B}$ guesses the right $q = \text{val}(T^*)$ and $r \in T^*$. If $q < n^{2/3}$, the algorithm returns the shortest path connecting a new demand in R as the covering tree T. Obviously in this case $\text{val}(T) = 1$. So we get

$$\text{cost-ef}_v(T) = \frac{c(T)}{\text{val}(T)} = c(T) \leq c(T^*) \leq n^{2/3} \cdot \frac{c(T^*)}{q} = \alpha \cdot \text{cost-ef}_v(T^*),$$

where the first inequality holds since T^* connects at least one new demand.

Next we consider the case $q \geq n^{2/3}$. By the constructive proof of Lemma 2, we know that T is extended by approximating the problem rooted quota-MST, which can be approximated with factor 2 by the work of [4]. Before new$(T) \geq 3q/8$, at least $\sqrt{q}/2$ new vertices are added to T in each iteration of step 4. So the number of iterations is at most $\frac{n}{\sqrt{q}/2} + 1 \leq 2n^{2/3} + 1$, and T is extended at most $2n^{2/3} + 1$ times from the trivial tree $\{r\}$, whose cost is zero. Thus we have $c(T) \leq (2n^{2/3} + 1) \cdot 2c(T^*)$. By step 4 of the algorithm, we have

$$\text{cost-ef}_v(T) = \frac{c(T)}{\text{val}(T)} \leq \frac{c(T)}{3q/8} \leq \frac{8}{3}(2n^{2/3} + 1)\frac{2c(T^*)}{q} = \alpha \cdot \text{cost-ef}_v(T^*)$$

provided $q \geq n^{2/3}$.

Finally, trying all the possible $1 \leq q \leq k'$ and $r \in V(G)$ and picking the tree with minimal cost-ef$_v(\cdot)$ completes the proof. □

Segev et al. proposed a separate construction method in [10] similar to the one in Lemma 2. For the details of the construction the readers may refer to [10]. Using the method in [10], we are able to construct an approximate covering tree T such that cost-ef$_v(T) \leq O(\sqrt{l}) \cdot$ cost-ef$_v(T^*)$. The proof of the following lemma is similar to that of Lemma 2 and Lemma 3 and hence is omitted.

Lemma 4. *A covering tree T can be found in polynomial time, such that cost-ef$_v(T) \leq \alpha \cdot$ cost-ef$_v(T^*)$, where $\alpha = O(\sqrt{l})$ and T^* is an optimal covering tree with respect to the function cost-ef$_v(\cdot)$.* □

3.2 The Greedy Algorithm for k-Steiner Forest

Now we are ready to deduce the greedy algorithm for k-Steiner Forest. The algorithm is given as algorithm $\mathcal{C}$. In each iteration of algorithm $\mathcal{C}$, the algorithm finds an approximate covering tree T and adds T to F, until at least k demands have been connected. At last, the algorithm outputs F as the solution. As before, the notation k' denotes the number of demands that need to be covered at the beginning of iteration j.

Algorithm $\mathcal{C}$

input: Graph G, demand set $D = \{(s_1, t_1), \ldots, (s_l, t_l)\}$, integer $k > 0$.
output: A collection of trees F that connects at least k demands in D.

1. **let** $F \leftarrow \emptyset$, $C \leftarrow \emptyset$, $R \leftarrow D$
2. **while** $k' = k - |C| > 0$ **do**
3. find a covering tree T by algorithm $\mathcal{B}$
4. **let** $F \leftarrow F \cup \{T\}$, $C \leftarrow C \cup D(T)$, $R \leftarrow R \setminus D(T)$
5. **end**
6. **return** F

In step 4 of algorithm $\mathcal{C}$, the notation $D(T)$ denotes the set of demands connected by T. For algorithm $\mathcal{C}$, we have the main theorem of this paper.

Theorem 1 (The main theorem). *k-Steiner Forest can be approximated within factor $O(\min\{n^{2/3}, \sqrt{l}\}\log k)$ in polynomial time.*

Proof. By Lemma 3, in each iteration of algorithm $\mathcal{C}$ an approximate covering tree T is found in polynomial time, such that cost-ef$_v(T) \leq \alpha_1 \cdot$ cost-ef$_v(T^*)$ where $\alpha_1 = O(n^{2/3})$. By the coupling lemma, algorithm $\mathcal{C}$ yields an $\alpha_1 H_k$-approximation to k-Steiner Forest. Also, algorithm $\mathcal{C}$ is of polynomial time, since there are at most k iterations in total. Similarly, by Lemma 4 and the coupling lemma, algorithm $\mathcal{C}$ yields an $\alpha_2 H_k$-approximation in polynomial time, where $\alpha_2 = O(\sqrt{l})$. The theorem follows. □

We give some technical comments on algorithm $\mathcal{C}$. Firstly, although an optimal solution to k-Steiner Forest must be a forest, in general the trees found by algorithm $\mathcal{C}$ are not disjoint. So, the output of algorithm $\mathcal{C}$ is really a collection of trees. Secondly, consider step 4 of the algorithm. When a covering tree T is added to F, there may be some new demands $\{d' = (s', t')\}$ covered by F and T together, since in general T is not disjoint from the already found trees in F. In algorithm $\mathcal{C}$, which obeys the framework of Set k-Cover, these demands in $\{d'\}$ are not added to C and hence still remain in R. In other words, the algorithm only considers the demands covered by a separated tree in F are "really" covered. On the other hand, if we also add the demand in $\{d'\}$ to C, and remove them from R, the same performance ratio will be proved by slightly modifying the proof of the coupling lemma (that is, by taking only as many demands in $\{d'\}$ to form a feasible k-cover and defining the price of them to be zero). The advantage of such action is only that it will accelerate the algorithm.

4 Discussion

We obtain an improved performance ratio $O(\min\{n^{2/3}, \sqrt{l}\}\log k)$ for the k-Steiner Forest problem via greedy approach, while the previous best known ratio in the literature is $O(\min\{n^{2/3}, \sqrt{l}\}\log l)$, obtained by Segev et al. [10] through the Lagrangian relaxation technique. Our global framework is completely different from that in [10]. Another difference is that our algorithm is simpler and faster. We believe that the coupling lemma for Set k-Cover has its own independent interest. To improve the performance ratio further for k-Steiner Forest in the framework of Set k-Cover, it seems that entirely new constructive method for building a covering tree is needed. Furthermore, the constructive method used in this paper does not take advantage of the special structure of the underlying graph, implying that approximating the problem k-Steiner Forest on trees, which is also proposed by Hajiaghayi and Jain [8], still remains open.

References

1. Agrawal, A., Klein, P., Ravi, R.: When trees collide: an approximation algorithm for the generalized Steiner problem on networks. SIAM Journal of Computing, **24**(3):440–456, 1995.
2. Chudak, F., Roughgarden, T., Williamson, D.: Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. Mathematical Proramming, **100**(2):411–421, 2004.
3. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. Algorithmica, **29**:410–421, 2001.
4. Garg, N.: Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC'05), 302–309, 2005.
5. Goemans, M., Willamson, D.: A general approximation technique for constrained forest problems. SIAM Journal on Computing, **24**(2):296–317, 1995.
6. Jain, K., Mahdian, M., Markaksi, E., Saberi, A., Vazirani, V.: Greedy facility location algorithms analyzied using dual fitting with factor-revealing LP. Journal of the ACM, **50**(6):795–824, 2003.
7. Jain, K., Vazirani, V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. Journal of the ACM, **48**(2):274–296, 2001.
8. Hajiaghayi, M., Jain, K.: The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06), 631–640, 2006.
9. Levin, A., Segev, D.: Partial multicuts in trees. In Proceedings of the 3rd International Workshop on Approximation and Online Algorithms (WAOA'05), 320–333, 2005.
10. Segev, D., Segev, G.: Approximate k-Steiner Forests via the Lagrangian relaxation technique with internal preprocessing. In Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06), 600–611, 2006.
11. Slavík, P.: Improved performance of the greedy algorithm for partial cover. Information Processing Letters, **64**:251–254, 1997.

A Distributed Algorithm of Fault Recovery for Stateful Failover

Indranil Saha[1] and Debapriyay Mukhopadhyay[2,⋆]

[1] Honeywell Technology Solutions Lab (HTSL),
151/1, Doraisanipalya, Bannerghatta Road, Bangalore 560 076, India
indranil.saha@honeywell.com
[2] Ixia Technologies,
Infinity Tower II, 8th Floor, Block GP, Sector V, Salt Lake, Kolkata 700091, India
dmukhopadhyay@ixiacom.com

Abstract. In [8], a high availability framework based on Harary graph as network topology has been proposed for stateful failover. Framework proposed therein exhibits an interesting property that an uniform load can be given to each non-faulty node while maintaining fault tolerance. A challenging problem in this context, which has not been addressed in [8] is to be able to come up with a distributed algorithm of automated fault recovery which can exploit the properties exhibited by the framework. In this work, we propose a distributed algorithm with low message and round complexity for automated fault recovery in case of stateful failover. We then prove the correctness of the algorithm using techniques from formal verification. The safety, liveness and the timeliness properties of the algorithm have been verified by the model checker SPIN.

Keywords: distributed algorithm, stateful failover, verification of programs, SPIN model checker.

1 Introduction

Critical business processes and mission critical systems should provide a high degree of availability and reliability to the end users and redundancy techniques are mostly used to achieve this. For distributed systems, redundancy can be achieved by using extra copies of its components which include hardware, software and network components. These systems are often called *highly available systems* or *fault tolerant systems* and they are capable of tolerating against certain kind of failures which can be either software or hardware component failures.

From the availability perspective, systems can be broadly classified into two heads depending on their intended application. In one hand, we have applications in which availability is one of the requirements, but occasional loss of application state information or data is tolerable. On the other hand, there are

⋆ When this work was carried out, the author was a member of Honeywell Technology Solution Lab, Bangalore, India.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 738–749, 2007.
© Springer-Verlag Berlin Heidelberg 2007

mission critical systems with stringent fault tolerance requirements, such as aircraft flight control systems, where restoration of the state or data pertaining to the application is required for highly accurate recovery. Thus, we have two types of failovers - stateless and stateful. In stateless failover of an application, the system is simply restarted without any state or data restoration.

A number of fault tolerant designs for specific multi-processor architectures have been proposed in the past [2,3,4,5,6,7]. Graph theoretic models have been extensively used to represent processor-to-processor interconnection structure is represented as a graph. As for example, minimum k-Hamilton graphs are widely used to meet reliability considerations for loop type communication networks [5,6,7]. Fault tolerant networks based on de Bruijn graph are proposed in [4] and can tolerate up to $k-2$ node faults, where the graph is regular of degree k and have k^n number of vertices for some n. But, none of these works discriminates between stateless and stateful failover. In case of stateless failover, any live node in the network is a promising candidate to take over the processes of any failed node, which in case of stateful failover is not. So the most pertinent questions that need to be addressed for stateful failover are - i) how to distribute the state information of a node across the network, i.e., how to decide who are the nodes to receive the state information of which nodes; ii) how to checkpoint the state information and how often it is required to be done.

A framework of high service availability for stateful failover, which is tolerant up to a maximum of k-processor faults in a network is proposed in [8] based on Harary graph. In this work the answer of the first question has been provided with sufficient theoretical justifications, but the second problem has not been dealt with. It has been assumed that the network is consisted of a set of multiprocessors and they are connected via bi-directional links. In addition, this work also relies on the assumption that the processors are equipped with enough computing power and thus can execute more than one processes.

High availability framework proposed there [8] not only enables the network to tolerate a maximum of k processor faults, but also guarantees that in the event of k processor failures, the load can be uniformly distributed across the rest of the network. Design always ensures connectedness among non-faulty nodes in the network, exploiting which it is possible to develop a distributed solution of automated recovery. But, the problem of finding a distributed solution of automated fault recovery has not been considered in that work. In this work, we aim to solve this problem and have come up with a distributed algorithm of automated fault recovery for stateful failover based on the framework proposed in [8] utilizing all the properties of the framework. Analysis shows that message and the round complexity of the algorithm are considerably low. Distributed algorithms are very hard to design and more so is to prove that they are correct. So, formal verification can be applied to prove the correctness of the distributed algorithms. Model checking is one such technique where a formal model of the system is constructed against which the properties of the system are verified. To prove the correctness of our algorithm we have used the model checker SPIN,

which is extensively used to formally verify distributed algorithms. Relevant properties of our algorithm, we have verified here using SPIN.

We organize our work as follows. Section 2 outlines the network and fault model and also reviews the framework for k-fault tolerance proposed in [8]. Distributed algorithm ensuring automated fault recovery is described in section 3 along with its analysis. In section 4, we describe the formal verification experiment on the proposed distributed algorithm. Section 5, finally, concludes our paper.

2 High Availability Framework for k-Fault Tolerance

2.1 Network and Fault Model

In this work we consider the same network and the fault model under which the framework proposed in [8] has been shown to function properly. The network is consisted of the set of nodes N with $|N| = n$ and each node is labeled with a unique id from 0 to $n-1$. We assume that each node is handling one process initially, and is capable of executing at most m processes simultaneously. We denote by p_i, the process which node $i \in N$ starts executing initially when the network becomes functional. We consider in this work failstop kind of failures, i.e., the nodes in the network can stop operating at any point of time due to a crash and k node faults are allowed in the network. With a processor failed, all the links incident on that node also becomes non-functional.

Given the set of nodes N, each node $i \in N, (0 \leq i \leq n-1)$, in the network is connected to the set of nodes $P_i \subseteq N$, such that $|P_i| = l = k + x$, where $k + x (\leq n-1)$ is even, and

$$P_i = \{j \in N : j = (i+p)(mod\ n), where - l/2 \leq p \leq l/2, p \neq 0\} \qquad (1)$$

Thus, the underlying undirected graph modeling the network can be written as (N, E) where $E = \cup_{i=0}^{n-1}\{(i,j) : j \in P_i\}$. Since, we are talking about undirected graphs, so (i,j) and (j,i) represents the same edge (link) connecting node i with j. Its easy to see here that the graph (N, E) represents a regular network, for, the degree of each node is l.

Thus, for any n and k, the graph (N, E) corresponds to the Harary Graph [1] $H_{l,n}$, where

$$l = k + x \geq \begin{cases} k+2, & \text{for } k \text{ even,} \\ k+1, & \text{for } k \text{ odd,} \end{cases}$$

and hence is l-connected with $\chi(G) \geq l (> k)$, $\chi(G)$ denotes the connectivity of G. This implies that there exists no node cutset of size k.

2.2 Framework

Given the value of k, the amount of fault tolerance that we require, the state information of processor i, $i \in N$, is periodically forwarded to all the nodes in

the set $F_i \subseteq N$ such that $|F_i| = k$ and

$$F_i = \{j \in S : j = (i+p)(mod\ n), where - \lfloor k/2 \rfloor \leq p \leq \lceil k/2 \rceil, p \neq 0\}. \quad (2)$$

Following Theorems have been theoretically proved in [8] for the framework. Here we will simply state the theorems for the sake of completeness as the distributed algorithm that we propose here utilizes those properties.

Theorem 1. A. *Forwarding state information of each process to k other nodes in the network following (2) ensures k-fault tolerance.*
B. *A sufficient condition to ensure k-fault tolerance is to forward the state information by each node to at least k other nodes in the network.*

Theorem 2. *As long as $k \leq \lfloor \frac{m-1}{m}.n \rfloor$, no live node has to execute more than m processes including one of its own and an algorithm to attain the same under the proposed framework can also be found.*

Theorem 3. *Minimum number of nodes with which any node in a network with $n > 2k$ (or $n = 2k$) is required to be connected directly is $2k$ (or $2k-1$) to ensure that all the eligible nodes corresponding to a process can be updated about its state information all the time in one hop.*

Note that, $F_i \subseteq P_i$ and only the nodes in F_i are updated about the state information of process p_i by node i in the event of no failure. Lets now consider the case that a set T of nodes has failed with $|T| \leq k$. Then, for each node $i \in T$, one of the node, say $j \in F_i - T$, starts executing p_i apart from running its own process p_j. For each $i \in T$, existence of such a $j \in F_i - T$ is guaranteed by Theorem 1. Node j then starts forwarding the state information of process p_i to the set of nodes in $F_i - T - \{j\}$ (Theorem 3 assures that node j is directly connected to all the nodes in $F_i - T - \{j\}$) and also continues sending the state information of its own process p_j to the set of nodes in $F_j - T$.

Given a set of k faulty nodes, Theorem 2 can be applied to find out a fault tolerant solution which will map each faulty node to a live node eligible to take up the process of the faulty node. This only applies when all the k nodes have either failed at the same time or they have failed at different times but fault tolerance is sought only after k nodes have failed. A more realistic consideration is to be able to apply fault tolerant solution when faults occur, i.e., being able to take into account the order in which faults have happened. To be specific, when node i has failed, the node that will fail next is unknown - it could even be the node who has taken up the process of node i. Designing a fault tolerant solution considering this realistic scenario is a challenging problem and in Section 3, we have provided a distributed solution for that.

3 Automated Fault Recovery

3.1 Distributed Algorithm

In this section, we present a distributed algorithm of automated fault recovery up to a maximum of k faults, the design of which has been done following a round

based model. In the first round when the network starts up, each and every node i sends an $INFO$ message to all the nodes in the set F_i. It also receives $INFO$ messages from all the nodes j in the network such that $i \in F_j$. $INFO$ message is consisted of the tuple (j, F_j) and thus helps the recipient (node i) of the message to know about the set of nodes eligible to take up the process p_j when node j fails. Every node i stores this (j, F_j) pair on its local memory on receipt of an $INFO$ message.

Node i then evaluates $(i - j)$ *mod* n and this value is assigned to the variable Id_diff_j. Node i also finds out its position in the set F_j and assigns it to the variable $pref_j^i$. Nodes that are ahead of i in their position in the set F_j will be considered before i when the need of executing process p_j of j arises. But, still they may not be in a position to take over process p_j because of the fact that either they are overloaded (i.e., executing m processes) or they have failed, in which case responsibility goes to node i. Node i assigns values to the variables Id_diff_j and $pref_j^i$ for each node j from which it receives an $INFO$ message. $pref_j^i$ is assigned either of the values $\frac{k}{2} - Id\_diff_j + 1$ or $\frac{k}{2} + (n - Id\_diff_j)$ depending on whether Id_diff_j is less than equal to or greater than $\frac{n}{2}$. The rational behind this assignment is to ensure that eligible nodes in the right of j are given higher preferences than nodes in the left of j while it comes to take over p_j. If $i \in F_j$ is the rightmost node from j then $pref_j^i = 1$, meaning that node i will be given the chance first to take over the process p_j of j.

Preference Finding Algorithm

[Code for node i, $0 \leq i \leq n-1$]

$node\_status = NOT\_OVERLOADED$;
$no\_of\_processes = 1$;
$T = \phi$;
$Other\_Processes = \phi$;

forall *nodes* j *such that* $i \in F_j$ **do**
  $fault\_flag_j = FALSE$;
end

Send $INFO$ message to all the nodes in F_i;

Receive $INFO$ messages from all the nodes j such that $i \in F_j$;

forall *such node* j **do**
  Save the tuple (j, F_j) in local memory;
  $Id\_diff_j = (i - j)$ *mod* n;
  if $Id\_diff_j \leq \frac{n}{2}$ **then**
    $pref_j^i = \frac{k}{2} - Id\_diff_j + 1$;
  else
    $pref_j^i = \frac{k}{2} + (n - Id\_diff_j)$;
  end
end

Algorithm 3.1. Preference Finding Algorithm for Node i

Starting from the second round in each successive rounds, every live node i sends $STATUS$ message for every process p_j (including its own process p_i) that is running on it to all the live nodes in the set F_j. $STATUS$ message for a process p_j is consisted of the tuple (p_j, S_{p_j}), where S_{p_j} denotes the state information of the corresponding process, which is required for any node in the set F_j to initiate the process. If in a round, node i does not receive the $STATUS$ message from a live node j in the network such that $i \in F_j$, then it assumes that node j has failed. If a node fails, then for each process running on it there can be at most k nodes to take up the load of this process. To decide which node will take up the load of which process being executed by this failed node, we adopt the following scheme.

When node i detects that node j such that $i \in F_j$ has failed, then it first checks whether it is already overloaded (i.e., executing m processes) or not. if it is not already overloaded, then node i waits up to $pref_j^i$ rounds to start the process p_j of the faulty node j by using the state information obtained from the last $STATUS$ message from j. After initiating process p_j of j, node i sends $RESOLVED_j$ message to all other nodes in the set F_j. But, if node i receives the $RESOLVED_j$ message within $pref_j^i$ rounds, then that means some other node in F_j has already taken up the process p_j and in which case node i does not require to start p_j. In case node i is already overloaded, it then expects to receive $RESOLVED_j$ message within $k + pref_j^i$ rounds. But if it does not receive the $RESOLVED_j$ message, then in the $k + pref_j^i$-th round it constructs a set A consisting of those processes $p_l, (l \neq i)$ such that $|F_j - T| \leq |F_l - T|$. If A is not an empty set (According to Theorem 1, the set A can never be empty, unless the number of faulty nodes is more than k), then node i finds out a process $p_l \in A$ for which $|F_l - T|$ is maximum and then stops this process and starts executing p_j. The process p_k that gets stopped by node i will be then taken up by some other node in F_k and this is possible by following the same algorithm since nodes in F_k without having received the $STATUS$ message corresponding to p_k detects that process k has failed.

We have two implicit requirements in order for this distributed algorithm to work. They are as follows. All the nodes in the network are required to be time synchronized; and the time required for a message to reach from one node to another should be less than the span of each round. We will now illustrate our algorithm through an example. Lets consider that network is consisted of 10 nodes and the value of k is equal to 4 and as such it is only required on behalf of each node to execute at most 2 processes (i.e., $m = 2$). Following the framework described in Section 2, the underlying graph modeling the network will be a regular graph with degree of each node equals to 8 and also $H_{4,10}$ comes as a subgraph of this network. Preferences of each node i, can then be found by following the *Preference Finding Algorithm* and they are as shown in Table 1. If we now consider that nodes $9, 2, 8$, and 0 will fail in this order, then Table 2 illustrates how the distributed *Fault Recovery Algorithm* works.

Fault Recovery Algorithm

[Code for node i, $0 \leq i \leq n-1$]

forall *process* p_k *executing in node* i **do**
  Send $STATUS$ message to all the nodes in F_k;
end

Receive $STATUS$ message from all the processes p_j such that $i \in F_j$;

if $STATUS$ *message is not received from* p_j *such that* $i \in F_j$ **then**
  $fault\_flag_j = TRUE$;
  $round_j = 0$;
  $T = T \cup \{j\}$;
end

forall *all the processes* p_j *such that* $i \in F_j$ **do**
  if $node\_status == NOT\_OVERLOADED$ **then**
    if $fault\_flag_j$ **then**
      $round_j = round_j + 1$;
      if $RESOLVED_j$ *is not yet received* **then**
        if $round_j == pref_j^i$ **then**
          Start process p_j;
          Send $RESOLVED_j$ message to all the other nodes in F_j;
          $Other\_Processes = Other\_Processes \cup \{p_j\}$;
          $fault\_flag_j = FALSE$;
          $no\_of\_processes = no\_of\_processes + 1$;
          if $no\_of\_processes == m$ **then**
            $node\_status = OVERLOADED$;
          end
        end
      else
        $fault\_flag_j = FALSE$;
      end
    end
  else
    if $fault\_flag_j$ **then**
      $round_j = round_j + 1$;
      if $RESOLVED_j$ *is not yet received* **then**
        if $round_j == k + pref_j^i$ **then**
          Find $A = \{p_l \in Other\_Processes : |F_j - T| \leq |F_l - T|\}$;
          if $A \neq \phi$ **then**
            Stop process $p_l \in A$ for which $|F_l - T|$ is maximum;
            Start process p_j;
            Send $RESOLVED_j$ message to all the other nodes in F_j;
            $Other\_Processes = (Other\_Processes - \{p_l\}) \cup \{p_j\}$;
            $fault\_flag_j = FALSE$;
          end
        end
      else
        $fault\_flag_j = FALSE$;
      end
    end
  end
end

Algorithm 3.2. Fault Recovery Algorithm for Node i

Table 1. Preference Table

i	$pref^i_{i-2}$	$pref^i_{i-1}$	$pref^i_{i+1}$	$pref^i_{i+1}$
0	$pref^0_8 = 1$	$pref^0_9 = 2$	$pref^0_1 = 3$	$pref^0_2 = 4$
1	$pref^1_9 = 1$	$pref^1_0 = 2$	$pref^1_2 = 3$	$pref^1_3 = 4$
2	$pref^2_0 = 1$	$pref^2_1 = 2$	$pref^2_3 = 3$	$pref^2_4 = 4$
3	$pref^3_1 = 1$	$pref^3_2 = 2$	$pref^3_4 = 3$	$pref^3_5 = 4$
4	$pref^4_2 = 1$	$pref^4_3 = 2$	$pref^4_5 = 3$	$pref^4_6 = 4$
5	$pref^5_3 = 1$	$pref^5_4 = 2$	$pref^5_6 = 3$	$pref^5_7 = 4$
6	$pref^6_4 = 1$	$pref^6_5 = 2$	$pref^6_7 = 3$	$pref^6_8 = 4$
7	$pref^7_5 = 1$	$pref^7_6 = 2$	$pref^7_8 = 3$	$pref^7_9 = 4$
8	$pref^8_6 = 1$	$pref^8_7 = 2$	$pref^8_9 = 3$	$pref^8_0 = 4$
9	$pref^9_7 = 1$	$pref^9_8 = 2$	$pref^9_0 = 3$	$pref^9_1 = 4$

Table 2. Illustration of the Algorithm

Failed Node Id	*Assigned To*	*No. of Rounds*	*Explanation*
9	1	1	$pref^1_9 = 1$
2	4	1	$pref^4_2 = 1$
8	0	1	$pref^0_8 = 1$
0	1	6	$pref^1_0 = 2$ (Stops p_9)
8	7	3	$pref^7_8 = 3$
9	7	8	$pref^7_9 = 4$ (Stops p_8)
8	6	4	$pref^6_8 = 4$

3.2 Analysis of the Algorithm

Theorem 4. *At most $2k$ rounds are required to resolve a single fault.*

Proof. For any process p_i corresponding to the node i, we have $|F_i| = k$. Now, when it is detected that node i has failed, the *Fault Recovery Algorithm* in each round examines a particular node in the set F_i to check whether it is in the $NOT\_OVERLOADED$ condition. That requires k rounds. Even in these k rounds it may not be possible for any of the nodes in F_i to start process p_i because some of them may have failed and some of them are overloaded. But, according to Theorem 1, it is not possible that all the nodes in F_i are faulty and as such there must be one live node eligible to take up the process p_i. If it happens to be the case that the only live node eligible to take up process p_i is having the highest $pref$ value, then k more rounds are still required for this node to start process p_i by forcefully stopping another process that it was executing previously. This is the worst case that we can think of fault to get resolved as is also evident from the example that we have explained earlier. Hence, at most $2k$ rounds are required to resolve a single fault. □

Theorem 5. *To resolve a single fault, the maximum number of $RESOLVED$ messages that is required to be sent across the network is $(k-2)m+1$, where m is the maximum number of processes that a node is capable of executing.*

Proof. Let us first consider that node i is executing m processes, viz, p_i (of its own), and $m-1$ other processes $p_{j_1}, p_{j_2}, \ldots, p_{j_{m-1}}$. This essentially means that nodes $j_1, j_2, \ldots, j_{m-1}$ have already failed. Then all these m processes are taken up by appropriate eligible processors. For each of these $m-1$ other processes there can be at most $k-2$ live nodes and it is required to send the $RESOLVED$ message to each of these live nodes. For p_i, the number of live nodes can be at most $k-1$ and it is also required to send the $RESOLVED$ message of p_i to each of them. Hence, the upper bound of the number of $RESOLVED$ messages that are required to be sent across the network is $(k-2)(m-1)+k-1 = (k-2)m+1$. □

4 Formal Verification of the Distributed Algorithm

To prove the correctness of the proposed algorithm, we use model checker SPIN [9] to formally verify the desired properties of the algorithm. SPIN is a tool for model checking distributed systems automatically and verifies properties of distributed algorithms modeled in the Promela language, by exploring their state space. Promela is a non-deterministic guarded-command language for modeling systems of concurrent processes that can interact via shared variables and message channels. Given a concurrent system modeled by a Promela program, SPIN can check for violations of user specified assertions, and temporal properties expressed by LTL formulas. When a violation of a property is detected, SPIN reports a scenario i.e. an execution sequence where the property is not valid.

We have modeled the processor as a parameterized process *processor*. The number of nodes, maximum fault-tolerance, maximum number of loads that can be taken by a processor are defined at the start of the model, so that they can be changed very easily to create different instances. The nodes have been defined as a structure which is as given below.

```
typedef Node {
        byte id;
        byte status;    /* 0 - Faulty, 1- Not Overloaded, 2 - Overloaded */
        byte noofprocesses;
        Neighbour neighbours[L];    /* L = 2k */
        Process processes[P];       /* P = m + 1 */
}
```

The neighbors and the processes follow the following data types

```
typedef Neighbour {
    byte id;
    byte status;        /* 0 - Faulty, 1- Not Overloaded, 2 - Overloaded */
}
typedef Process {
    byte procid;
    byte nodeid;
    byte faultflag;     /* 0 - Not Faulty,  1- Faulty,  2 - Unknown */
    byte round;
}
```

The *neighbors* of a node are those nodes to whom the $STATUS$ message of the node is required to be sent. The *processes* for a node represents the processes that the node may have to take at any point of time in future. The *procid* of a process is the ID of the node where the process was running for the first time. The *nodeid* field of a process indicates on which node the process is currently running. The *faultflag* variable is used to keep track of the status of the node where the process was running last. When a neighbor of a node detects that the node has become faulty it makes the flag corresponding to that process 1. Whenever it takes up that process or receives a $RESOLVED$ message corresponding to that process, it makes the faultflag 0. This is done to check the liveness property that will be discussed later. The variable *round* indicates that how many rounds have been passed after the node detected that the node where the process was running is faulty.

To introduce faults in any order, a separate process *endround* has been introduced. After the end of a round, zero, one or more number of processors are randomly made faulty by this process until the total number of faulty nodes are less than equal to k.

In order to prove the correctness, we have considered for verification safety property, liveness property, and timeliness property. Liveness property has been expressed as a LTL formula, and other properties are inserted as simple assertions at proper places in the Promela model. The properties are described below.

Safety 1. Whenever a node becomes faulty, at least one of its neighboring nodes is non-faulty. This property has been checked as an assertion in the *endround* process after making a node faulty.
assert(node[node[j].neighbours[0].id].status != 0 ||
node[node[j].neighbours[1].id].status != 0);

Safety 2. No node has to take more than M processes at any point of time.
In the process corresponding to a processor, when a node take up a new process we increase its *noofprocesses* field by 1 and this is followed by the following assertion
$assert(node[nodeid].noofprocesses <= M)$,
where the meaning of M has been discussed earlier.

Liveness. Whenever a node becomes faulty, its process is eventually taken up by some other live nodes.
The property has been expressed by the following LTL formula
$[](s -><> t)$,
$s \cong$ node[0].processes[1].faultflag==1, $t \cong$ node[0].processes[1].faultflag==0

Timeliness. Every fault is recovered in no more than 2K rounds.
When a processor detects that the node where one of the processes that it is supposed to take was running is faulty, it starts counting the round. When this round is equal to its preference value for that process, then if that process has not yet been taken up by some other process, it takes up that process. So, in any case, the value of the round should not be more than 2K according to Theorem 3. We check this by the following assertion
$assert(node[nodeid].processes[j].round <= 2 * K)$

We have been able to verify our model for N=8, K=3 and M=2 and all lower instances. Due to the state-space explosion problem inherent in model checker SPIN, we could not verity our algorithm for more than 8 processors.

5 Conclusion

In this paper we have presented a distributed algorithm of automated fault recovery for stateful failover in a network. In whatever way the fault may arise the algorithm can handle that fault and in at most $2k$ rounds the processes of the faulty processor are taken up by a(some) eligible live node(nodes) in the network. The message complexity of our algorithm is linear with the number of nodes. The correctness of the algorithm has been proved by modeling the algorithm in SPIN and verifying its desired properties.

References

1. F. Harary, "The Maximum Connectivity of a Graph", *Proc. Nat. Acad. Sci.*, U.S.A., 48, pages 1142-1146, 1962.
2. J. G. Kuhl and S. M. Reddy, "Distributed Fault Tolerance for Large Multiprocessor Systems", *Computer Architecture News*, 8, page 23-30, *7th Intl. Symposium on Computer Architectures*, 1980.
3. C.L. Yang and G. M. Massona, "Distributed Algorithm for Fault Diagnosis in Systems with Soft Failures", *IEEE Transaction on Computers*, Vol. 37, No. 11, November 1988.
4. M. A. Sridhar, C. S. Raghavendra, "Fault-Tolerant Network Based on De Bruijn Graph", *IEEE Transaction On Computers*, Vol. 40, No. 10, October 1991.
5. K.Mukhopadhyay, B.P. Sinha, "Hamiltonian Graphs with Minimum Number of Edges For Fault-Tolerant Topologies", *Information Processing Letters*, 44, pages 95-99, November, 1992.
6. T. Sung, T. HO, C. Chang AND L. Hsu, "Optimal k-Fault-Tolerant Networks for Token Rings", *Journal of Information Science and Engineering*, 16, pages 381-390, 2000.
7. C. Hung, L. Hsu, T. Sung, "On the construction of combined k-fault-tolerant Hamiltonian graphs", *Networks*, 37(3), pages 165-170, 2001.
8. I. Saha, D. Mukhopadhyay, S. Banerjee, "Designing Reliable Architecture for Stateful Fault Tolerance, In Proceedings of *Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)*, pages 545-551, 2006.
9. G. J. Holtzman. The SPIN Model Checker, Primer and Reference Manual, Addison-Wesley, 2003.

6 Appendix

6.1 Harary Graph

An undirected graph G can be represented as (N, E), where N denotes a set of nodes and E denotes a set of edges which can be defined as an unordered

pair of distinct nodes. A *node cut* of G is a subset N' of N such that $G - N'$ is disconnected, where $G - N'$ is obtained by removing all the nodes in N' and also by removing all the edges incident on the nodes in N'. A *k-node cut* is a node cut of cardinality k. The minimum cardinality of a node cut is called *connectivity* $\chi(G)$ of G and the graph G is called *k-connected* if $\chi(G) \geq k$, i.e., there exists no node cut of size $k-1$. Its a known result that the least number of edges that a k-connected graph on $n(n > k)$ vertices can have is greater or equal to $\frac{nk}{2}$. Harary graph [1] $H_{k,n}$, which we will be describing next, is a k-connected graph on n vertices with exactly $\frac{nk}{2}$ edges.

Structure of a Harary graph $H_{k,n}$ is defined for three different cases.

Case 1: k **even.** Then $H_{k,n} = (N, E)$ is defined as follows. The nodes in N are labeled as $0, 1, 2, \ldots, n-1$ and

$$E = \cup_{i=0}^{n-1}\{(i,j) : j = (i+p)(mod\ n), where - k/2 \leq p \leq k/2 \text{ and } p \neq 0\}. \quad (3)$$

(i, j) and (j, i) both denotes the same edge, since we are talking about undirected graphs.

Case 2: k **odd,** n **even.** $H_{k,n}$ is then constructed by first drawing $H_{k-1,n}$ following case 1 and then by adding edges joining node i to node $(i + \frac{n}{2})(mod\ n)$ for $1 \leq i \leq \frac{n}{2}$.

Case 3: k **odd,** n **odd.** $H_{k,n}$ is then constructed by first drawing $H_{k-1,n}$ and then joining node 0 to nodes $\frac{n-1}{2}$ and $\frac{n+1}{2}$ and node i to node $(i + \frac{n+1}{2})(mod\ n)$ for $1 \leq i \leq \frac{n-1}{2}$.

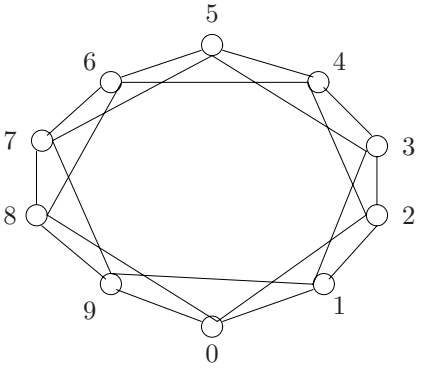


Fig. 1. Harary Graph $H_{4,10}$

The fact that the graph $H_{k,n}$ is k-connected with minimum number of edges is known from Harary, 1962.

Path Embedding on Folded Hypercubes

Sun-Yuan Hsieh

Department of Computer Science and Information Engineering,
National Cheng Kung University,
No. 1, University Road, Tainan 70101, Taiwan
hsiehsy@mail.ncku.edu.tw

Abstract. We analyze some edge-fault-tolerant properties of the folded hypercube, which is a variant of the hypercube obtained by adding an edge to every pair of nodes with complementary address. We show that an n-dimensional folded hypercube is $(n-2)$-edge-fault-tolerant Hamiltonian-connected when $n(\geq 2)$ is even, $(n-1)$-edge-fault-tolerant strongly Hamiltonian-laceable when $n(\geq 1)$ is odd, and $(n-2)$-edge-fault-tolerant hyper Hamiltonian-laceable when $n(\geq 3)$ is odd.

1 Introduction

Because of the hypercube's importance, many variants of it have been proposed (for example, see [3,6,7,16]). One variant that has been the focus of a great deal of research is the *folded hypercube*, an extension of the hypercube constructed by adding an edge to every pair of nodes that are the farthest apart, i.e., two nodes with complementary addresses. It has been shown that, compared to a regular hypercube, the folded hypercube can improve the system's performance in many measurements, such as diameter, mean inter-node distance, and traffic density [3,20].

A *graph* $G = (V, E)$ is a pair of two sets composed of a *node set* V and an *edge set* E, where V is a finite set and E is a subset of $\{(u, v)|\ (u, v)$ is an unordered pair of $V\}$. We also use $V(G)$ and $E(G)$ to denote the node set and edge set of G, respectively. A *path*, $P[v_0, v_k] = \langle v_0, v_1, \ldots, v_k\rangle$, is a sequence of distinct nodes in which any two consecutive nodes are adjacent. We call v_0 and v_k the *end-nodes* of the path. A path with end-nodes u and v is said to be a *u-v path*. A path may contain a *subpath*, denoted as $\langle v_0, v_1, \ldots, v_i, P[v_i, v_j], v_j, v_{j+1}, \ldots, v_k\rangle$, where $P[v_i, v_j] = \langle v_i, v_{i+1}, \ldots, v_{j-1}, v_j\rangle$. A *cycle* is a path with $v_0 = v_k$ and $k \geq 3$. When the Hamiltonicity of a graph G is being investigated, it is necessary to determine whether G is Hamiltonian or Hamiltonian-connected. A cycle (respectively, path) in G is called a *Hamiltonian cycle* (respectively, *Hamiltonian path*) if it contains every node of G exactly once. G is said to be *Hamiltonian* if it contains a Hamiltonian cycle, and *Hamiltonian-connected* if there exists a Hamiltonian path between every two nodes of G.

A graph $G = (V_0 \bigcup V_1, E)$ is *bipartite* if $V_0 \cap V_1 = \emptyset$ and $E \subseteq \{(x, y)|\ x \in V_0$ and $y \in V_1\}$. We say V_0 and V_1 are *partite sets* of G, and $V_0 \bigcup V_1$ a *bipartition*. Two well-known interconnection networks, hypercubes [6,14] and star

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 750–759, 2007.
© Springer-Verlag Berlin Heidelberg 2007

graphs [1,12], are both bipartite. However, because a bipartite graph is not Hamiltonian-connected, except for K_1 or K_2, Simmons [17] introduced the concept of Hamiltonian-laceability for such graphs. A Hamiltonian bipartite graph $G = (V_0 \bigcup V_1, E)$ is *Hamiltonian-laceable* if there is a Hamiltonian path between any two nodes x and y, where $x \in V_0$ and $y \in V_1$. Hsieh *et al.* [11] extended this concept and proposed the concept of *strong Hamiltonian-laceability*. A graph $G = (V_0 \bigcup V_1, E)$ is *strongly Hamiltonian-laceable* if there is a simple path of length $|V_0| + |V_1| - 2$ between any two nodes of the same partite set. Lewinter *et al.* [15] introduced another concept, called hyper Hamiltonian-laceability. A bipartite graph $G = (V_0 \bigcup V_1, E)$ is *hyper Hamiltonian-laceable* if for any node $f \in V_i$, $i \in \{0, 1\}$, there is a Hamiltonian path of $G - f$ between any two nodes of V_{1-i}.[1]

The edge-fault-tolerant Hamiltonicity proposed by Hsieh *et al.* [10] measures Hamiltonicity in interconnection networks with faulty edges. A Hamiltonian graph G is *k-edge-fault-tolerant Hamiltonian* if $G - F$ remains Hamiltonian for every $F \subset E(G)$ with $|F| \leq k$. A Hamiltonian-laceable graph G is *k-edge-fault-tolerant Hamiltonian-laceable* if $G - F$ remains Hamiltonian-laceable for every $F \subset E(G)$ with $|F| \leq k$. A strongly Hamiltonian-laceable graph G is *k-edge-fault-tolerant strongly Hamiltonian-laceable* if $G - F$ remains strongly Hamiltonian-laceable for every $F \subset E(G)$ with $|F| \leq k$. A hyper Hamiltonian-laceable graph G is *k-edge-fault-tolerant hyper Hamiltonian-laceable* if $G - F$ remains hyper Hamiltonian-laceable for every $F \subset E(G)$ with $|F| \leq k$.

Latifi *et al.* [13] showed that an n-dimensional hypercube is $(n-2)$-edge-fault-tolerant Hamiltonian. Tsai *et al.* [18] further showed that an n-dimensional hypercube is $(n-2)$-edge-fault-tolerant strongly Hamiltonian-laceable, and $(n-3)$-edge-fault-tolerant hyper Hamiltonian-laceable. Wang [20] showed that the n-dimensional folded hypercube is $(n-1)$-edge-fault-tolerant Hamiltonian. It is known that the n-dimensional folded hypercube is bipartite (non-bipartite) when n is odd (even) [23]. In this paper, we show that an n-dimensional folded hypercube is $(n-2)$-edge-fault-tolerant Hamiltonian-connected when $n(\geq 2)$ is even, $(n-1)$-edge-fault-tolerant strongly Hamiltonian-laceable when $n(\geq 1)$ is odd, and $(n-2)$-edge-fault-tolerant hyper Hamiltonian-laceable when $n(\geq 3)$ is odd.

2 Preliminaries

When using undirected graphs to model interconnection networks, our fundamental graph terminologies follow those in [21]. A *subgraph* of $G = (V, E)$ is a graph (V', E') such that $V' \subseteq V$ and $E' \subseteq E$. Given a set $V' \subseteq V$, the subgraph of $G = (V, E)$ *induced* by V' is the graph $G' = (V', E')$, where $E' = \{(u, v) \in E |\ u, v \in V'\}$. In a graph G, the *neighbors* of a node v are the nodes adjacent to it in G, and the *degree* of v is the number of edges incident

[1] Let S be a set of edges and/or nodes of a graph G. Throughout this paper, the notation $G - S$ represents the resulting graph obtained by deleting those elements in S from G.

to it. A graph is said to be *regular* if all nodes have a common degree. Two graphs G_1 and G_2 are *isomorphic* if there is a one-to-one function π from $V(G_1)$ onto $V(G_2)$ such that $(u,v) \in E(G_1)$ iff $(\phi(u), \phi(v)) \in E(G_2)$.

An n-dimensional hypercube (n-cube for short) can be represented as an undirected graph $Q_n = (V, E)$, where V consists of 2^n nodes that are labelled as binary numbers of length n from $\underbrace{00\ldots0}_{n}$ to $\underbrace{11\ldots1}_{n}$, and E is the set of edges that connects two nodes iff their labels differ by exactly one bit. Note that Q_n is regular because the degree of each node is equal to n, and $|E| = n2^{n-1}$. An edge $e = (u,v) \in E$ is said to be of *dimension* i if $u = b_n b_{n-1} \ldots b_i \ldots b_1$ and $v = b_n b_{n-1} \ldots \overline{b_i} \ldots b_1$, where $b_j \in \{0,1\}$ for $j = 1, 2, \ldots, n$, and $\overline{b_i}$ is the *one's complement of* b_i, i.e., $\overline{b_i} = 1 - b_i$. Note that there are 2^{n-1} edges in each dimension.

Let $x = x_n x_{n-1} \ldots x_1$ be an n-bit binary string. For $1 \leq k \leq n$, we use x^k to denote the binary strings $y_n y_{n-1} \ldots y_1$ such that $y_k = 1 - x_k = \overline{x_k}$ and $x_i = y_i$ for all $i \neq k$. The *Hamming weight* $hw(x)$ of x is the number of i's that make $x_i = 1$. Let $x = x_n x_{n-1} \ldots x_1$ and $y = y_n y_{n-1} \ldots y_1$ be two n-bit binary strings. The *Hamming distance* $h(x,y)$ between two nodes x and y is the number of different bits in the corresponding strings of both nodes. Note that Q_n is a bipartite graph with a bipartition $\{x|\ hw(x) \text{ is odd}\}$ and $\{x|\ hw(x) \text{ is even}\}$. Let $d_G(x,y)$ be the distance of the shortest path between two nodes x and y in graph G. It is known that $d_{Q_n}(x,y) = h(x,y)$.

An *n-dimensional folded hypercube* (*folded n-cube* for short) FQ_n is a regular n-dimensional hypercube augmented by adding more edges between its nodes. More specifically, a folded n-cube is obtained by adding an edge between two nodes whose addresses are complementary, i.e., a node whose address is $b = b_n b_{n-1} \ldots b_1$ now has one more edge to node $\overline{b} = \overline{b_n b_{n-1}} \ldots \overline{b_1}$, in addition to its original n edges. Hence, FQ_n has 2^{n-1} more edges than Q_n. We call these augmented edges *skips*, to distinguish them from regular edges.

For convenience, FQ_n can be represented by $\underbrace{**\ldots**}_{n} = *^n$, where $* \in \{0,1\}$ means "*don't care*". An *i-partition on* $FQ_n = *^n$ partitions FQ_n along dimension i for some $i \in \{1, 2, \ldots, n\}$, into two *subcubes*, $Q^0_{n-1} = *^{n-i}0*^{i-1}$ and $Q^1_{n-1} = *^{n-i}1*^{i-1}$, where Q^0_{n-1} and Q^1_{n-1} are the subgraphs of FQ_n induced by $\{x_n \ldots x_i \ldots x_1 \in V(FQ_n)|\ x_i = 0\}$ and $\{x_n \ldots x_i \ldots x_1 \in V(FQ_n)|\ x_i = 1\}$), respectively. Note that each Q^j_{n-1}, where $j \in \{0,1\}$, is isomorphic to an $(n-1)$-cube. An *i-partition of an n-cube* Q_n can be defined similarly.

The following lemmas are useful in our method.

Lemma 1. [8] *Let x and y be two distinct nodes in Q_n; and let $h(x,y) = d$. There are x-y paths in Q_n whose lengths are $d, d+2, d+4, ..., c$, where $c = 2^n - 1$ if d is odd, and $c = 2^n - 2$ if d is even.*

Lemma 2. [18] *The following two statements hold:*

1. *Q_n is $(n-2)$-edge-fault-tolerant strongly Hamiltonian-laceable.*
2. *Q_n is $(n-3)$-edge-fault-tolerant hyper Hamiltonian-laceable.*

It is well known that Q_n (FQ_n for odd n) is bipartite. Thus, the following proposition is used in each proof presented in this paper.

Proposition 1. *For two distinct nodes x and y in Q_n (or FQ_n for odd n), $h(x, y)$ is odd (even) iff x and y are in different partite sets (the same partite set).*[2]

Two paths are said to be *node-disjoint* if they have no common node.

Lemma 3. [19] *Let V_0 and V_1 be partite sets of a fault-free Q_n, where $n \geq 2$. Let a and b be two distinct nodes of V_0, and a' and b' be two distinct nodes of V_1. Then, there exist two node-disjoint paths $P[a, a']$ and $P[b, b']$ spanning $V(Q_n)$, i.e., $V(P[a, a']) \cup V(P[b, b']) = V(Q_n)$.*

3 Three Edge-Fault-Tolerant Properties

Let $Q^0_{n-1}(= *^{n-i}0*^{i-1})$ and $Q^1_{n-1} = (*^{n-i}1*^{i-1})$ be two subcubes after executing an i-partition on FQ_n. We define the set of *crossing edges* as $E_C = \{(u, v) \in E(FQ_n)|\ u \in V(Q^0_{n-1}), v \in V(Q^1_{n-1}),$ and $h(u, v) \neq n\}$, and the set of skips as $E_S = \{(u, v) \in E(FQ_n)|\ u \in V(Q^0_{n-1}), v \in V(Q^1_{n-1}),$ and $h(u, v) = n\}$. Moreover, let F be the set of faulty edges of FQ_n; $F_0 = F \cap E(Q^0_{n-1})$; $F_1 = F \cap E(Q^1_{n-1})$; $F_C = F \cap E_C$; and let $F_S = F \cap E_S$. For a node $u = u_n u_{n-1} \dots u_1$ in FQ_n, recall that $\overline{u} = \overline{u_n} \dots \overline{u_1}$.

3.1 Edge-Fault-Tolerant Hamiltonian-Connectivity

In this subsection, we demonstrate the edge-fault-tolerant Hamiltonian-connectivity of the folded hypercube.

Lemma 4. *FQ_n is $(n-2)$-edge-fault-tolerant Hamiltonian-connected when $n(\geq 2)$ is an even integer.*

Proof. Since FQ_2 is a complete graph comprised of four nodes, it is clearly Hamiltonian-connected. We now consider FQ_n, where $n \geq 4$ is an even integer. In the following, we attempt to construct a fault-free Hamiltonian path between two arbitrary distinct nodes x and y when $|F| = n-2$. We consider the following two cases.

Case 1. $h(x, y)$ is odd. As FQ_n is constructed from Q_n by adding skips, $FQ_n - F$ contains a subgraph G that is isomorphic to Q_n with at most $n-2$ faulty edges. Since $h(x, y)$ is odd, x and y are in different partite sets in H. By Lemma 2(1), G contains a fault-free Hamiltonian path between x and y, and so as $FQ_n - F$.

Case 2. $h(x, y)$ is even. We have the following scenarios.

[2] Hereafter, the terms "$h(x, y)$ is odd" and "x and y are in different partite sets" are used interchangeably; and "$h(x, y)$ is even" and "x and y are in the same partite set" are used similarly.

Case 2.1. There is at least one faulty skip, i.e., $F_S \neq \emptyset$. We can execute an i-partition on FQ_n for some $i \in \{1, 2, \ldots, n\}$ such that x and y are in different subcubes. Without loss of generality, we assume that $x \in V(Q^0_{n-1})$ and $y \in V(Q^1_{n-1})$. Since $F_S \neq \emptyset$, we have $|F_C \cup F_S| \geq 1$. Recall that both F_C and F_S are two set of edges located between Q^0_{n-1} and Q^1_{n-1} after executing an i-partition. Therefore, the number of faulty edges remaining in each of $\{Q^0_{n-1}, Q^1_{n-1}\}$ is at most $(n-2)-1 = n-3$, i.e., $|F_0| \leq n-3$, and $|F_1| \leq n-3$. Let $u(\neq x)$ be a node in Q^0_{n-1} such that $h(x,u)$ is odd, $\overline{u} \neq y$, and $(u, \overline{u})$ is fault-free. (If such a node u does not exist, then $|F_S| \geq 2^{n-2} - 1 > n-2$ for $n \geq 4$, which is a contradiction.) Clearly, $h(x, \overline{u}) = n - h(x,u)$ is odd because n is even, i.e., x and $\overline{u}$ are in different partite sets in the subgraph Q_n according to Proposition 1.[3] Moreover, since x and y are in the same partite set in Q_n because of even $h(x,y)$, y and $\overline{u}$ are in different partite sets in Q_n. Since $|F_j| \leq n-3$ for $j = 0, 1$, by Lemma 2(1), $Q^0_{n-1} - F_0$ ($Q^1_{n-1} - F_1$) contains a fault-free Hamiltonian path $P_0[x,u]$ ($P_1[\overline{u}, y]$). A desired Hamiltonian x-y path can be constructed as $P_0[x,u] \oplus (u, \overline{u}) \oplus P_1[\overline{u}, y]$, where $\oplus$ denotes a path-concatenation operation.[4]

Case 2.2. There are no faulty skips, i.e., $F_S = \emptyset$. In this case, no faulty edges are skips. Let e be an arbitrary faulty edge whose dimension is i, where $i \in \{1, 2, \ldots, n\}$. We can execute an i-partition on FQ_n such that $|F_C \cup F_S| = |F_C| \geq 1$. Using an argument similar to that in Case 2.1, we have $|F_0| \leq n-3$ and $|F_1| \leq n-3$.

Case 2.2.1. x and y are in the same subcube. Without loss of generality, we assume that $x, y \in V(Q^0_{n-1})$.

Case 2.2.1.1. $|F_j| = n-3$ for some $j \in \{0, 1\}$. Without loss of generality, we assume that $|F_0| = n-3$. Then, $|F_C| = 1$ and $|F_1| = 0$. We first select an arbitrary node w in Q^0_{n-1} such that w is in a different partite set to the partite set that $\{x, y\}$ belongs to. We then select one arbitrary faulty edge $(u,v) \in F_0$. By Lemma 2(2), $Q^0_{n-1} - w - (F_0 - (u,v))$ contains a Hamiltonian path $P_0[x,y]$. We then have the following scenarios.

Case 2.2.1.1.1. $P_0[x,y]$ contains (u,v). Path $P_0[x,y]$ can be represented by $P_0[x,u] \oplus (u,v) \oplus P_0[v,y]$. Consider four nodes $\overline{u}, \overline{v}, \overline{w}$, and w^i. Since $\overline{w} \neq \overline{u}$ and $\overline{w} \neq \overline{v}$, we have $|\{\overline{u}, \overline{v}\} \cap \{\overline{w}, w^i\}| \leq 1$.

Case 2.2.1.1.1.1. $|\{\overline{u}, \overline{v}\} \cap \{\overline{w}, w^i\}| = 0$. Clearly, $\overline{u}$ and $\overline{v}$ belong to different partite sets in Q^1_{n-1}. Moreover, since $h(x, w^i)$ is even and $h(x, \overline{w}) = n - h(x,w)$ is odd, w^i and $\overline{w}$ belong to different partite sets in Q^1_{n-1}; Hence, there are two nodes, one derived from $\{\overline{u}, \overline{v}\}$ and the other from $\{w^i, \overline{w}\}$,

[3] For convenience, we adopt the notation Q_n to represent a subgraph in FQ_n that is isomorphic to an n-dimensional hypercube.

[4] Throughout the paper, we use the notation "$\oplus$" to represent a path-concatenation operation in order to distinguish it from an ordinary addition "+" operation.

which come from different partite sets. Without loss of generality, we assume that $\overline{u}$ and $\overline{w}$ are in different partite sets, and $\overline{v}$ and w^i are in different partite sets. By Lemma 3, Q^1_{n-1} contains two node-disjoint paths $P_1[\overline{u}, \overline{w}]$ and $P_1[\overline{v}, w^i]$ spanning $V(Q^1_{n-1})$. A desired Hamiltonian x-y path can be constructed as $P_0[x, u] \oplus (u, \overline{u}) \oplus P_1[\overline{u}, \overline{w}] \oplus (\overline{w}, w) \oplus (w, w^i) \oplus P_1[w^i, \overline{v}] \oplus (\overline{v}, v) \oplus P_0[v, y]$.

Case 2.2.1.1.2. $|\{\overline{u}, \overline{v}\} \cap \{\overline{w}, w^i\}| = 1$. Without loss of generality, we assume that $\overline{v} = w^i$. Recall that $\overline{u}$ and $\overline{v}$ are in different partite sets, and w^i and $\overline{w}$ are in different partite sets. Therefore, $\overline{u}$ and $\overline{w}$ are in the same partite set in Q^1_{n-1}. By Lemma 2(2), $Q^1_{n-1} - w$ contains a fault-free Hamiltonian path $P_1[\overline{u}, \overline{w}]$. A desired Hamiltonian x-y path can be constructed as $P_0[x, u] \oplus (u, \overline{u}) \oplus P_1[\overline{u}, \overline{w}] \oplus (\overline{w}, w) \oplus (w, \overline{v}) \oplus (\overline{v}, v) \oplus P_0[v, y]$.

Case 2.2.1.1.2. $P_0[x, w]$ does not contain (u, v). In this case, we can select an arbitrary edge in place of (u, v). A desired Hamiltonian x-y path can then be constructed using a method similar to that in Case 2.2.1.1.1.

Case 2.2.1.2. $|F_0| \leq n-4$ and $|F_1| \leq n-4$. We first select a node $w \in V(Q^0_{n-1})$ such that (w, w^i) is fault-free and w is in a different partite set to the partite set that x and y belong to. (If such a w does not exist, then $|F_c| > 2^{n-2} > n-2$ for $n \geq 4$, which is a contradiction.) By Lemma 2(2), $Q^0_{n-1} - w - F_0$ contains a Hamiltonian path $P_0[x, y]$. Let v be a unique node in $P_0[x, y]$ such that $\overline{v} = w^i$, and let $u \in P_0[x, y]$ be a unique neighbor of v such that $P_0[x, y] = P_0[x, u] \oplus (u, v) \oplus P_0[v, y]$. Using an argument similar to that applied in Case 2.2.1.1.1, we know that $\overline{u}$ and $\overline{w}$ are in different partite sets. Again, by Lemma 2(2), $Q^1_{n-1} - w - F_0$ contains a fault-free Hamiltonian path $P_1[\overline{u}, \overline{w}]$. Therefore, a desired Hamiltonian x-y path can be constructed as $P_0[x, u] \oplus (u, \overline{u}) \oplus P_1[\overline{u}, \overline{w}] \oplus (\overline{w}, w) \oplus (w, \overline{v}) \oplus (\overline{v}, v) \oplus P_0[v, y]$.

Case 2.2.2. x and y are in different subcubes. Without loss of generality, we assume that $x \in Q^0_{n-1}$ and $y \in Q^1_{n-1}$. Let $w(\neq x)$ be a node in Q^0_{n-1} such that $h(x, w)$ is odd. Note that $h(x, \overline{w}) = n - h(x, w)$ is also odd. Moreover, since x and y are in the same partite set, $\overline{w}$ and y are in different partite sets (restricted to Q_n). Since both $|F_0|$ and $|F_1|$ are at most $n-3$, $Q^0_{n-1} - F_0$ ($Q^1_{n-1} - F_1$) contains a fault-free Hamiltonian path $P_0[x, w]$ ($P_1[\overline{w}, y]$) by Lemma 2(1). Therefore, a desired Hamiltonian x-y path can be constructed as $P_0[x, w] \oplus (w, \overline{w}) \oplus P_1[\overline{w}, y]$.

By combining the above cases, we complete the proof.

Due to the space limitation, we omit the proof for FQ_n being $(n-1)$-edge-fault-tolerant strongly Hamiltonian-laceable when $n(\geq 1)$ is odd.

3.2 Edge-Fault-Tolerant Hyper Hamiltonian-Laceability

In this subsection, we demonstrate the edge-fault-tolerant hyper Hamiltonian-laceability of the folded hypercube.

Lemma 5. *FQ_n is $(n-2)$ edge-fault-tolerant hyper Hamiltonian-laceable, where $n(\geq 3)$ is an odd integer.*

Proof. Suppose that $FQ_n = (V_0 \cup V_1, E)$, where $n(\geq 3)$ is odd. Let f be any node in V_i, $i \in \{0,1\}$. In the following, for two arbitrary distinct nodes $x, y \in V_{1-i}$, we attempt to construct a fault-free Hamiltonian x-y path in $FQ_n - F - f$, where $|F| = n-2$. We consider the following two cases.

Case 1. $F_S \neq \emptyset$. Since $FQ_n - F$ contains a subgraph isomorphic to Q_n with at most $n-3$ faulty edges, the result holds by Lemma 2(2).

Case 2. $F_S = \emptyset$. In this case, all faulty edges are not skips. We can execute an i-partition on FQ_n, for some $i \in \{1, 2, \ldots, n\}$, such that $|F_C \cup F_S| = |F_C| \geq 1$, $|F_0| \leq n-3$, and $|F_1| \leq n-3$. There are the following scenarios.

Case 2.1. $|F_C| = 1$ and $|F_j| = n-3$ for some $j \in \{0,1\}$. Without loss of generality, we assume that $|F_0| = n-3$. Then, $|F_1| = 0$.

Case 2.1.1. $x, y \in V(Q^0_{n-1})$ and $f \in V(Q^1_{n-1})$.

By Lemma 2(1), $Q^0_{n-1} - F_0$ contains a fault-free Hamiltonian cycle $C_0 = \langle u_0, u_1, \ldots u_{2^{n-1}-1}, u_0\rangle$ of length 2^{n-1}, where $x = u_0$ and $y = u_k$ for some $k \in \{1, \ldots, 2^{n-1}-1\}$. As $h(x,y)$ is even, $k \geq 2$. Let $x' = u_{k+1 \pmod{2^{n-1}}}$ and $y' = u_1$. Clearly, $Q^0_{n-1} - F_0$ contains two fault-free paths, $P_0[x,x'] = \langle u_0, u_{2^{n-1}-1}, u_{2^{n-1}-2}, \ldots, u_{k+1 \pmod{2^{n-1}}}\rangle$ and $P_0[y',y] = \langle u_1, u_2, \ldots, u_k\rangle$, which spans $V(Q^0_{n-1})$. Since x and y are in the same partite set and x' and y are in different partite sets, x and x' are in different partite sets, i.e., $h(x,x')$ is odd. Similarly, y and y' are in different partite sets, i.e., $h(y,y')$ is odd. Moreover, $h(x,\overline{x'}) = n - h(x,x')$ and $h(y,\overline{y'}) = n - h(y,y')$ are both even, i.e., $x, y, \overline{x'}$, and $\overline{y'}$ are in the same partite set and thus $h(\overline{x'},\overline{y'})$ is even. Note that f and $\{\overline{x'},\overline{y'}\}$ are in different partite sets. Since Q^1_{n-1} is fault-free, by Lemma 2(2), $Q^1_{n-1} - f$ contains a fault-free Hamiltonian path $P_1[\overline{x'},\overline{y'}]$ of length $2^{n-1}-2$. A desired x-y path can be constructed as $P_0[x,x'] \oplus (x',\overline{x'}) \oplus P_1[\overline{x'},\overline{y'}] \oplus (\overline{y'},y') \oplus P_0[y',y]$, which has length $2^{n-1} + 2^{n-1} - 2 = 2^n - 2$.

Case 2.1.2. $f, x, y \in V(Q^0_{n-1})$. Let (u,v) be an arbitrary faulty edge in F_0. Note that $(u,\overline{u})$ and $(v,\overline{v})$ are both fault-free. By Lemma 2(2), $Q^0_{n-1} - f - (F_0 - (u,v))$ contains a path $P_0[x,y]$ of length $2^{n-1}-2$.

Case 2.1.2.1. $P_0[x,y]$ contains (u,v). Thus $P_0[x,y] = P_0[x,u] \oplus (u,v) \oplus P_0[v,y]$. By Lemma 1, Q^1_{n-1} contains a fault-free Hamiltonian path $P_1[\overline{u},\overline{v}]$ of length $2^{n-1}-1$. A desired x-y path can be constructed as $P_0[x,u] \oplus (u,\overline{u}) \oplus P_1[\overline{u},\overline{v}] \oplus (\overline{v},v) \oplus P_0[v,y]$, which has length $(2^{n-1}-2) - 1 + 2 + (2^{n-1}-1) = 2^n - 2$.

Case 2.1.2.2. $P_0[x,y]$ does not contain (u,v). In this case, we select an arbitrary edge in $P_0[x,y]$ instead of (u,v) in Case 2.1.2.1. The construction of a desired path is similar to that of Case 2.1.2.1.

Case 2.1.3. $f, x \in V(Q^0_{n-1})$ and $y \in V(Q^1_{n-1})$. By Lemma 2(1), $Q^0_{n-1} - F_0$ contains a fault-free Hamiltonian path $P_0[x, f]$. Let $u \in P_0[x, f]$ be the neighbor of f. Thus $P_0[x, f] = P_0[x, u] \oplus (u, f)$. Note that u and x are in the same partite set, and $\overline{u}$ and y are in different partite sets. By Lemma 1, Q^1_{n-1} contains a fault-free Hamiltonian path $P_1[\overline{u}, y]$. A desired x-y path can be constructed as $P_0[x, u] \oplus (u, \overline{u}) \oplus P_1[\overline{u}, y]$, which has length $2^n - 2$.

Case 2.1.4. $x \in V(Q^0_{n-1})$ and $f, y \in V(Q^1_{n-1})$. Let $w(\neq x)$ be the node in Q^0_{n-1} such that $h(x, w)$ is odd and $\overline{w} \notin \{f, y\}$. Since $h(\overline{w}, x) = n - h(x, w)$ is even and $h(x, y)$ is even, y and $\overline{w}$ are in the same partite set. Since $|F_0| = n - 3$, by Lemma 2(1), $Q^0_{n-1} - F_0$ contains a fault-free Hamiltonian path $P_0[x, w]$. Moreover, by Lemma 2(2), $Q^1_{n-1} - f - F_1$ contains a fault-free Hamiltonian path $P_1[\overline{w}, y]$. A desired x-y path can be constructed as $P_0[x, w] \oplus (w, \overline{w}) \oplus P_1[\overline{w}, y]$, which has length $2^n - 2$.

Case 2.1.5. $f, x, y \in V(Q^1_{n-1})$. By Lemma 2(2), $Q^1_{n-1} - f$ contains a fault-free Hamiltonian path $P_1[x, y]$. Let (u, v) be an edge in $P_1[x, y]$. Thus $P_1[x, y] = P_1[x, u] \oplus (u, v) \oplus P_1[v, y]$. By Lemma 2(1), $Q^0_{n-1} - F_0$ contains a fault-free Hamiltonian path $P_0[\overline{u}, \overline{v}]$. A desired x-y path can be constructed as $P_1[x, u] \oplus (u, \overline{u}) \oplus P_0[\overline{u}, \overline{v}] \oplus (\overline{v}, v) \oplus P_1[v, y]$, which has length $2^n - 2$.

Case 2.1.6. $f \in V(Q^0_{n-1})$ and $x, y \in V(Q^1_{n-1})$. Let $u \neq f$ be a node in Q^0_{n-1} whose partite set is the same with the partite set to which x and y belong. Therefore, u and f are in different partite sets. Since $|F_0| = n - 3$, by Lemma 2(1), $Q^0_{n-1} - F_0$ contains a Hamiltonian path $P_0[u, f]$. Let $v \in P_0[u, f]$ be the neighbor of f. Thus $P_0[u, f] = P_0[u, v] \oplus (v, f)$. Clearly, u and v are in the same partite set, and $\overline{u}$ and $\overline{v}$ are in the same partite set. Further, the partite set of $\{\overline{u}, \overline{v}\}$ is different from the partite set to which $\{x, y\}$ belongs. By Lemma 3, Q^1_{n-1} contains two node-disjoint paths $P_1[x, \overline{u}]$ and $P_1[\overline{v}, y]$ spanning $V(Q^1_{n-1})$. A desired x-y path can be constructed as $P_1[x, \overline{u}] \oplus (\overline{u}, u) \oplus P_0[u, v] \oplus (v, \overline{v}) \oplus P_1[\overline{v}, y]$, which has length $2^n - 2$.

Case 2.2. $|F_C| > 1$, $|F_0| \leq n - 4$, and $|F_1| \leq n - 4$.

Case 2.2.1. $f, x, y \in V(Q^j_{n-1})$ for some $j \in \{0, 1\}$. Without loss of generality, we assume that $f, x, y \in V(Q^0_{n-1})$. By Lemma 2(2), $Q^0_{n-1} - f - F_0$ contains a fault-free Hamiltonian path $P_0[x, y]$. Let (u, v) be an arbitrary edge in $P_0[x, y]$ and thus $P_0[x, y] = P_0[x, u] \oplus (u, v) \oplus P_0[v, y]$. By Lemma 2(1), Q^1_{n-1} contains a fault-free Hamiltonian path $P_1[\overline{u}, \overline{v}]$. A desired x-y path can be constructed as $P_0[x, u] \oplus (u, \overline{u}) \oplus P_1[\overline{u}, \overline{v}] \oplus (\overline{v}, v) \oplus P_0[v, y]$, which has length $2^n - 2$.

Case 2.2.2. $x, y \in V(Q^j_{n-1})$ and $f \in V(Q^{1-j}_{n-1})$ for some $j \in \{0, 1\}$. The proof is similar to that of Case 2.1.1 and thus we omit here.

Case 2.2.3. $x, f \in V(Q^j_{n-1})$ and $y \in V(Q^{1-j}_{n-1})$ for some $j \in \{0, 1\}$. Without loss of generality, we assume that $x, f \in V(Q^0_{n-1})$ and $y \in V(Q^1_{n-1})$. Let $w(\notin \{f, x\})$ be the node in Q^0_{n-1} such that $h(x, w)$ is even. Since $h(x, \overline{w}) = n - h(x, w)$ is odd, $\overline{w} \neq y$ because $h(x, y)$

is even. This implies that $\overline{w}$ and y are in different partite set. By Lemma 2(2), $Q^0_{n-1} - f - F_0$ contains a fault-free Hamiltonian path $P_0[x, w]$. Moreover, by Lemma 2(1), $Q^1_{n-1} - F_1$ contains a fault-free Hamiltonian path $P_1[\overline{w}, y]$. A desired x-y path can be constructed as $P_0[x, w] \oplus (w, \overline{w}) \oplus P_1[\overline{w}, y]$, which has length $2^n - 2$.

By combining the above cases, we complete the proof.

We now present our main result.

Theorem 1. *There are three edge-fault-tolerant properties for FQ_n as follows:*

P1. FQ_n is $(n-2)$-edge-fault-tolerant Hamiltonian-connected, where $n(\geq 2)$ is an even integer.

P2. FQ_n is $(n-1)$-edge-fault-tolerant strongly Hamiltonian-laceable, where $n(\geq 1)$ is an odd integer.

P3. FQ_n is $(n-2)$-edge-fault-tolerant hyper Hamiltonian-laceable, when $n(\geq 3)$ is an odd integer.

4 Concluding Remarks

The path (linear array) is the most fundamental network for parallel and distributed computation, which is suitable for designing simple algorithms with low communication costs. Numerous efficient algorithms designed on the path for solving various algebraic problems and graph problems can be found in [2,14]. The path can also be used as control/data flow structure for distributed computation in arbitrary networks. Another application for the longest path to a practical problem was addressed in the on-line optimization of a complex flexible manufacturing system [4]. These applications motivate the embedding of paths in networks. Our result implies that those algorithms designed for paths can also be executed well on the folded hypercube with faulty edges.

References

1. S. B. Akers, D. Harel, and B. Krishnamurthy, The star graph: an attractive alternative to the n-cube, Proceedings of International Conference on Parallel Processing, St. Charles, IL, 1987, pp. 555–556.
2. S. G. Akl, Parallel Computation: Models and Methods, Prentice Hall, NJ, 1997.
3. Ahmed El-Amawy and Shahram Latifi, Properties and performance of folded hypercubes, IEEE Transactions on Parallel and Distributed Systems 2(1991), 31–42.
4. N. Ascheuer, Hamiltonian path problems in the on-line optimization of flexible manufacturing systems, Ph.D. Thesis, University of Technology, Berlin, Germany, 1995 (also available from ⟨ ftp://ftp.zib.de/pub/zib-publications/reports/TR-96-03.ps⟩).
5. J. C. Bermond, Ed., "Interconnection networks," a special issue of Discrete Applied Mathematics, 1992, Vol. 37–38.
6. L. Bhuyan and D. P. Agrawal, Generalized hypercubes and hyperbus structure for a computer network, IEEE Transactions on Computers c33(1984), 323–333.

7. A. H. Esfahanian, L. M. Ni, and B. E. Sagan, The twisted n-cube with application to multiprocessing, IEEE Transactions on Computers 40(1991), 88–93.
8. J. S. Fu and G. H. Chen, Hamiltonicity of the hierarchical cubic network, Theory of Computing Systems 35(2002), 59–79.
9. D. F. Hsu, "Interconnection networks and algorithms," a special issue of *Networks*, 1993, Vol. 23, No. 4.
10. S. Y. Hsieh, G. H. Chen, and C. W. Ho, Fault-free hamiltonian cycles in faulty arrangement graphs, IEEE Transactions on Parallel Distributed Systems 10(1999), 223–237.
11. S. Y. Hsieh, G. H. Chen, and C. W. Ho, Hamiltonian-laceability of star graphs, Networks 36(2000), 225–232.
12. J. S. Jwo, S. Lakshmivarahan, and S. K. Dhall, Embedding of cycles and grids in star graphs, Journal of Circuits, Systems, and Computers 1(1991), 43–74.
13. S. Latifi, S. Q. Zheng, and N. Bagherzadeh, Optimal ring embedding in hypercubes with faulty links, Proceedings of the Twenty-Second Annual International Symposium on Fault-Tolerant Computing, Boston, Massachusetts, USA, 1992, pp. 178–184.
14. F. T. Leighton, Introduction to Parallel Algorithms and Architecture: Arrays· Trees· Hypercubes, Morgan Kaufmann, San Mateo, CA, 1992.
15. M. Lewinter and W. Widulski, Hyper-Hamiltonian laceable and caterpillar-spannable product graphs, Computer and Mathematics with Applications 34(1997), 99–104.
16. F. P. Preparata and J. Vuillemin, The cube-connected cycles: a versatile network for parallel computation, Communication of the ACM 24(1981), 300–309.
17. G. Simmons, Almost all n-dimensional retangular lattices are Hamiltonian laceable, Congressus Numerantium 21(1978), 103–108.
18. Chang-Hsiung Tsai, Jimmy J. M. Tan, T. Liang, and L. H. Hsu, Fault-tolerant hamiltonian laceability of hypercubes, Information Processing Letters 83(2002), 301–306.
19. Chang-Hsiung Tsai, Linear array and ring embedding in conditional faulty hypercubes, Theoretical Computer Science 314(2004), 431–443.
20. Dajin Wang, Embedding Hamiltonian cycles into folded hypercubes with faulty links, Journal of Parallel and Distributed Computing 61(2001), 545–564.
21. D. B. West, Introduction to Graph Theory, Prentice-Hall, Upper Saddle River, NJ 07458, 2001.
22. Junming Xu, Topological Structure and Analysis of Interconnection Networks, Kluwer academic publishers, Dordrecht, The Netherlands, 2001.
23. Junming Xu, Cycles in folded hypercubes, Applied Mathematics Letters 19(2006), 140–145.

An Approximation Algorithm Based on Chain Implication for Constrained Minimum Vertex Covers in Bipartite Graphs*

Jianxin Wang[1], Xiaoshuang Xu[1], and Jianer Chen[1,2]

[1] School of Information Science and Engineering, Central South University, Changsha 410083, China
jxwang@mail.csu.edu.cn
[2] Department of Computer Science Texas A&M University
College Station, TX 77843, USA
chen@cs.tamu.edu

Abstract. The constrained minimum vertex cover problem on bipartite graphs (the Min-CVCB problem) is an NP-complete problem. This paper presents a polynomial time approximation algorithm for the problem based on the technique of chain implication. For any given constant $\varepsilon > 0$, if an instance of the Min-CVCB problem has a minimum vertex cover of size (k_u, k_l), our algorithm constructs a vertex cover of size (k_u^*, k_l^*), satisfying $\max\{k_u^*/k_u, k_l^*/k_l\} \leq 1 + \varepsilon$.

1 Introduction

With the development of VLSI technology, the scale of electric circuit chip becomes larger and larger, and the possibility of introducing defects also increases along with the manufacture craft. With the increasing in the chip integration, it is not allowed that the wrong memory element appears in the manufacture process. A better solution is to use reconfigurable arrays. A typical reconfigurable memory array consists of a rectangular array plus a set of spare rows and spare columns. A defective element is repaired by replacing the row or the column containing the element with a spare row or a spare column. Since the cost of reconfiguration is proportional to the number of replaced rows and columns, people often replace as few as possible rows and columns to repair the array. This problem can be formulated as a constrained minimum vertex cover problem on bipartite graphs [3,7], which is formulated as follows.

Definition 1. [Constrained minimum vertex cover in bipartite graphs (Min-CVCB)] Given a bipartite graph $G = (V, E)$ with the vertex bipartition $V = U \cup L$ and two integers k_u and k_l, determine whether there is a minimum vertex cover of G with at most k_u vertices in U and at most k_l vertices in L.

* This work is supported by the National Natural Science Foundation of China (60433020) and the Program for New Century Excellent Talents in University (NCET-05-0683).

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 760–769, 2007.
© Springer-Verlag Berlin Heidelberg 2007

For simplify our description, we will say that a bipartite graph $G = (U \cup L, E)$ has a minimum vertex cover of size (k_u, k_l) if G has a minimum vertex cover with at most k_u vertices in U and at most k_l vertices in L.

The Min-CVCB problem is NP-complete [3]. The problem was proposed by Hasan and Liu [7] in 1988, who introduced the concept of the critical set to develop a branch-and-bound algorithm for solving the Min-CVCB problem, based on the A^* algorithm [12]. No explicit analysis was giving in [7] for the running time of their algorithm, but it is not hard to see that in the worst-case the running time of the algorithm is at least $O(2^{k_u+k_l} + mn^{1/2})$. The Min-CVCB problem has been extensively studied in last two decades, mainly on heuristic algorithms. Interested readers are referred to [1,2,8,9,11,13] for the progress.

Recently, people used the parameterized computation theory to study exact algorithms for the Min-CVCB problem [3,6]. After analyzing the structures of bipartite graphs, Fernau and Niedermeier [6] used the branching search technology to develop an algorithm of time complexity $O((k_u + k_l)n + 1.4^{k_u+k_l})$. Chen and Kanj [3] proved that the Min-CVCB problem is NP-complete. By using classical results in matching theory and recently developed techniques in parameterized computation theory, they proposed an exact algorithm with time complexity $O((k_u + k_l)|G| + 1.26^{k_u+k_l})$, which is currently the best exact algorithm for the Min-CVCB problem.

Both heuristic algorithms and exact algorithms for the Min-CVCB problem have drawbacks. Heuristic algorithms are unable to provide solutions in a guaranteed time; while exact algorithms could get optimal solutions but run in exponential time. In this paper, we are interested in polynomial time approximation algorithms for the Min-CVCB problem. While a polynomial time approximation algorithm may not be able to find an optimal solution, it finds a near-optimal solution with a guaranteed error bound. In practice, near-optimality is often good enough, and acceptable in many application fields.

Our algorithm proceeds as follows. We firstly reduce the input size of a given instance of the Min-CVCB problem by applying the technique of kernelization proposed in parameterized computation theory [5]. Then we make full use of the classic matching theory on bipartite graphs to give a deep analysis of the structures of bipartite graphs. Based on the analysis, we get a $1+\varepsilon$ approximation algorithm based on the technique of chain implication for the Min-CVCB problem, in the following sense: for a given constant $\varepsilon > 0$, if the instance of the Min-CVCB problem has a constrained minimum vertex cover of size (k_u, k_l), then our algorithm produces a vertex cover of size (k_u^*, k_l^*) satisfying $\max\{k_u^*/k_u, k_l^*/k_l\} \leq 1 + \varepsilon$.

The paper is organized as follows. In section 2, we formally define what is an approximation algorithm for the Min-CVCB problem. Section 3 describes some related definitions and lemmas. The heart of this paper is section 4, which proposes a $1 + \varepsilon$ approximation algorithm for the Min-CVCB problem based on the formulation of a directed acyclic graph. Section 5 gives the description of our algorithm and an explicit analysis of its approximation ratio and time complexity. Conclusions and future research are given in section 6.

2 Approximation Algorithms for the Min-CVCB Problem

First, we give some basic concepts in approximation algorithm theory. More details could be found in [4].

Definition 2. Given an NP optimization problem Q, we say that an algorithm A for Q has an *approximation ratio* $f(n)$ if, for any given instance I of Q, the value $A(I)$ of the solution produced by the algorithm A is within a factor of $f(|I|)$ of the value $Opt(I)$ of an optimal solution:

$$\max\{A(I)/Opt(I), Opt(I)/A(I)\} \leq f(|I|)$$

Note that $f(n)$ can be a constant.

Definition 3. Let Q be an NP optimization problem. If there exists an approximation algorithm A for Q that takes (x, ε) as its input, where x is an instance of Q and $\varepsilon > 0$ is a constant, and outputs a solution y to x such that the approximation ratio of y over the optimal solution to x is bounded by $1 + \varepsilon$, and for any fixed constant $\varepsilon > 0$, the algorithm A runs in time polynomial in the size of its input instance, then we say that the approximation algorithm A is a *polynomial-time approximation scheme* (shortly a PTAS) for the problem Q.

Therefore, approximation algorithms and PTAS are for computational optimization problems, in which a solution of optimal value is searched. On the other hand, the Min-CVCB problem is a decision problem, in which we only need to answer "yes" or "no" by determining if a given bipartite graph $G = (U \cup L, E)$ has a minimum vertex cover of size (k_u, k_l). To study approximability of the Min-CVCB problem, we formally define, as follows, what is a PTAS for the Min-CVCB problem.

Definition 4. An algorithm A is a *polynomial time approximation scheme* (i.e., PTAS) for the Min-CVCB problem if for any instance $(G = (U \cup L); k_u, k_l)$ of the Min-CVCB problem, and for any given constant $\varepsilon > 0$, the algorithm A either claims that the bipartite graph G has no minimum vertex cover of size (k_u, k_l), or, in case G has a minimum vertex cover of size (k_u, k_l), constructs a vertex cover of size (k_u^*, k_l^*) for G, satisfying $\max\{k_u^*/k_u, k_l^*/k_l\} \leq 1 + \varepsilon$, and the algorithm A runs in polynomial time for any fixed $\varepsilon > 0$.

3 Related Definitions and Lemmas

Our algorithm that searches for a near-optimal solution for the Min-CVCB problem contains the following two steps:

(1) Reducing to the kernel. The main idea of reducing to the kernel is to reduce the algorithm's search space size. To reduce to the Min-CVCB problem's kernel is to reduce the size of the input bipartite graph. In Lemma 1, by using a polynomial time preprocessing algorithm, we reduce a given instance of the

Min-CVCB problem to an equivalent instance $(G'; k'_u, k'_l)$ in which the bipartite graph G' has a perfect matching and has at most $2(k'_u + k'_l)$ vertices. Based on this, Lemma 2 applies the classical matching theory to convert it into a directed acyclic graph consisting of elementary bipartite graphs.

(2) Making full use of the technique of chain implication to enumerate the minimum vertex covers of the elementary bipartite graphs whose size is larger than a constant to search for a vertex cover for the given graph. It makes sure that, if the input bipartite graph G' has a minimum vertex cover of size (k'_u, k'_l), then the algorithm will find a close enough vertex cover for G'.

We start with some related definitions and related known results.

Definition 5. A graph G is *bipartite* if its vertex set can be partitioned into two sets U (the "upper part") and L (the "lower part") such that every edge in G has one endpoint in U and the other endpoint in L. A bipartite graph is written as $G{=}(U \cup L, E)$ to indicate the vertex bipartition. The vertex sets U and L are called the *U-part* and the *L-part* of the graph, and a vertex is a *U-vertex* (resp. an *L-vertex*) if it is in the U-part (resp. in the L-part) of the graph.

Let $G = (U \cup L, E)$ be a bipartite graph with a perfect matching. The graph G is *elementary* [10] if every edge in G is contained in a perfect matching in G. It is known that an elementary bipartite graph has only two minimum vertex covers, namely U and L [10].

Lemma 1. ([3]) *The time complexity for solving an instance $(G; k_u, k_l)$ of the Min-CVCB problem, where G is a bipartite graph of n vertices and m edges, is bounded by $O(mn^{1/2} + t(k_u + k_l))$, where $t(k_u + k_l)$ is the time complexity for solving an instance (G', k'_u, k'_l) of the problem, with $k'_u \leq k_u$, $k'_l \leq k_l$, and G' has a perfect matching and contains at most $2(k'_u + k'_l)$ vertices.*

Lemma 2. (The Dulmage-Mendelsohn Decomposition theorem [10]) *A bipartite graph $G = (U \cup L, E)$ with perfect matching can be decomposed and indexed into elementary subgraphs $B_i = (U_i \cup L_i, E_i)$, $i = 1, 2, \ldots, r$, such that every edge in G from a subgraph B_i to a subgraph B_j with $i < j$ must have one endpoint in the U-part of B_i and the other endpoint in the L-part of B_j. Such a decomposition can be constructed in time $O(|E|^2)$.*

An elementary subgraph B_i will be called an (elementary) *block*. The *weight* of a block $B_i = (U_i \cup L_i, E_i)$ is defined to be $|U_i| = |L_i|$. Edges connecting vertices in two different blocks will be called *inter-block edges*.

Lemma 3. ([3]) *Let G be a bipartite graph with perfect matching, and let B_1, B_2, ..., B_r be the blocks of G given by the Dulmage-Mendelsohn Decomposition. Then every minimum vertex cover for G is a union of minimum vertex covers of the blocks B_1, B_2, ..., B_r.*

We say that an instance $(G; k_u, k_l)$ of the Min-CVCB problem is *normalized* if $G = (U \cup L, E)$ has a perfect matching, $|L| = |U| \leq (k_u + k_l)$, and a Dulmage-Mendelsohn Decomposition of G is given. By Lemmas 1 and 2, we only need to concentrate on normalized instances of the Min-CVCB problem. Formally, this approach is validated by the following results

Lemma 4. ([3]) *There is an algorithm that, on a given instance $(G; k_u, k_l)$ of the Min-CVCB problem, constructs a subset Z_0 of vertices in G and a normalized instance $(G'; k'_u, k'_l)$ of the problem, where G' is a subgraph of G, such that*

(1) Every vertex cover of G' plus Z_0 is a vertex cover of G; and

(2) A vertex set Y in G' is a constrained minimum vertex cover of size (k'_u, k'_l) for G' if and only if the set $Y \cup Z_0$ is a constrained minimum vertex cover of size (k_u, k_l) for G.

The running time of the algorithm is $O(|G|^2)$.

By Lemma 4, if we want to develop a polynomial time approximation scheme for the Min-CVCB problem, we can simply concentrate on normalized instances of the problem, as shown by the following corollary.

Corollary 1. *Let $(G; k_u, k_l)$ be an instance of the Min-CVCB problem. Then a normalized instance $(G'; k'_u, k'_l)$ of the problem can be constructed in time $O(|G|^2)$, such that from a vertex cover of size (k''_u, k''_l) for G' satisfying the condition $\max\{k''_u/k'_u, k''_l/k'_l\} \leq 1+\varepsilon$, we can construct in time $O(|G|)$ a vertex cover of size (k^*_u, k^*_l) for G satisfying $\max\{k^*_u/k_u, k^*_l/k_l\} \leq 1+\varepsilon$.*

Proof. For the given instance $(G; k_u, k_l)$ of the Min-CVCB problem, where $G = (U \cup L, E)$ is a bipartite graph, we apply the algorithm in Lemma 4 to construct, in time $O(|G|)^2)$, the set Z_0 and the normalized instance $(G'; k'_u, k'_l)$ satisfying the conditions in the lemma. Suppose that the set Z_0 contains u U-vertices and l L-vertices in the bipartite graph G. By condition (2) in Lemma 4, we must have $k'_u + u = k_u$ and $k'_l + l = k_l$.

Suppose that Y is a vertex cover of size (k''_u, k''_l) for the graph G' satisfying $\max\{k''_u/k'_u, k''_l/k'_l\} \leq 1+\varepsilon$. By condition (1) in Lemma 4, the set $Y \cup Z_0$ is a vertex cover of size (k^*_u, k^*_l) for the graph G, where $k^*_u = k''_u + u$ and $k^*_l = k''_l + l$. Moreover, we have

$$k^*_u/k_u = (k''_u + u)/(k'_u + u) \leq \max\{1, k''_u/k'_u\} \leq 1+\varepsilon, \text{ and}$$
$$k^*_l/k_l = (k''_l + l)/(k'_l + l) \leq \max\{1, k''_l/k'_l\} \leq 1+\varepsilon.$$

That is, $Y \cup Z_0$ is a vertex cover of size (k^*_u, k^*_l) for the graph G that can be constructed from the vertex cover Y of the graph G' in time $O(|G|)$ and satisfies the condition $\max\{k^*_u/k_u, k^*_l/k_l\} \leq 1+\varepsilon$. □

4 The AACI-D Algorithm

We concentrate on normalized instances $(G; k_u, k_l)$ of the Min-CVCB problem, by which we assume that the bipartite graph $G = (U \cup L, E)$ has a perfect matching, $|L| = |U| \leq (k_u + k_l)$, and a Dulmage-Mendelsohn Decomposition $\{B_1, \ldots, B_r\}$ of G is given, where each B_i is a block in G, and every edge between two blocks B_i and B_j with $i < j$ has one end in the U-part of B_i and the other end in the L-part of B_j.

We construct a directed acyclic graph (a DAG) D from the decomposition $\{B_1, \ldots, B_r\}$ of G, as follows. Each block B_i in G corresponds to a vertex w_i in D, and there is a directed arc from w_i to w_j in the DAG D if and only if there is an edge from the U-part of the block B_i to the L-part of the block B_j in the graph G.

Let $\varepsilon > 0$ be a fixed constant. The PTAS for the Min-CVCB problem on normalized instances is given in Figure 1. Without loss of generality, we assume that $k_u \geq k_l$ (otherwise, we simply exchange U and L).

Algorithm AACI-D$(G, k_u, k_l, \varepsilon)$
input: a normalized instance (G, k_u, k_l) of Min-CVCB, and a constant $\varepsilon > 0$
output: a vertex cover for G, or report no vertex cover of size (k_u, k_l) for G

1. construct the DAG D based on the decomposition $\{B_1, \ldots, B_r\}$ of G;
2. let $\mathcal{B}$ be the collection of the blocks B_i whose weight is at least εk_u;
3. **for** each union Y of minimum vertex covers of the blocks in $\mathcal{B}$ **do**
3.1 $Y = Y \cup Y'$, where Y' is the set of vertices that are forced by Y;
3.2 **if** Y is consistent, with u U-vertices and l L-vertices **then**
let $B'_1, \ldots, B'_t$ be the rest of the blocks, sorted as in the DAG D;
let h be the first index such that $u + |B'_1| + \cdots |B'_h| > k_u$;
add the U-parts of $B'_1, \ldots, B'_h$ and the L-parts of $B'_{h+1}, \ldots, B'_t$ to Y;
return Y;
4. return ("no vertex cover of size (k_u, k_l) for G").

Fig. 1. The AACI-D Algorithm for normalized instances of Min-CVCB

We first give some explanations to the algorithm. Let $B_i = (U_i \cup L_i, E_i)$ be a block in the graph G. By Lemma 3, every minimum vertex cover of G either includes U_i but excludes L_i, or includes L_i but excludes U_i. Thus, for two blocks $B_i = (U_i \cup L_i, E_i)$ and $B_j = (U_j \cup L_j, E_j)$ in the graph G with an edge from U_i to L_j (thus $i < j$), if a minimum vertex cover of G includes L_i then it must also include L_j (since it excludes U_i and there is an edge from U_i to L_j). In this case, we say that the set L_j is *forced* (to be in the minimum vertex cover) by the set L_i. Similar derivation shows that the set U_i is forced by the set U_j. More generally, let $B_i = (U_i \cup L_i, E_i)$ be a block in the graph G, and let $\mathcal{B}_i$ be the collection of blocks B_j such that there is a directed path from B_i to B_j in the DAG D. Then all L-parts of the blocks in $\mathcal{B}_i$ are forced by the set L_i. Similarly, let $\mathcal{B}'_i$ be the collection of blocks B_j such that there is a directed path from B_j to B_i in the DAG D, then all U-parts of the blocks in $\mathcal{B}'_i$ are forced by the set U_i. Even more generally, let Y be a set that is a union of U-parts and L-parts of some blocks in G that does not contain both U-part and L-part of any block, then we say that a set Y' is *forced* by Y if the set Y' consists of those vertices that are either L-vertices forced by L-vertices in Y or U-vertices that is forced by U-vertices in Y. This technique of forcing more vertices into a minimum vertex cover using the existing vertices in the minimum vertex cover is called the *chain implication* technique [3].

For the given instance $(G; k_u, k_l)$ of the Min-CVCB problem, we say that a set Y of vertices in the graph G is *consistent* if each block of the graph G either has no intersection with Y, or has exactly one of its U-part or L-part entirely contained in the set Y, and if the number of U-vertices in Y is bounded by k_u and the number of L-vertices in Y is bounded by k_l.

Now we are ready to discuss the algorithm. Let $\{B_1, \ldots, B_r\}$ be the Dulmage-Mendelsohn Decomposition of the graph G, sorted in the order that every edge between two blocks B_i and B_j with $i < j$ has one end in the U-part of B_i and the other end in the L-part of B_j. As described in the algorithm, let $\mathcal{B}$ be the collection of the blocks B_i whose weight is at least εk_u.

By Lemma 3, every minimum vertex cover of the graph G is a union of minimum vertex covers of its blocks. Therefore, if the graph G has a minimum vertex cover Y_0 of size (k_u, k_l), then in the enumeration of step 3, we will eventually get a set Y in which the minimum vertex cover for each block B_i in $\mathcal{B}$ is the same as that in the set Y_0. For such a set Y, after forcing further vertices in Y' to the set Y, we still get a set Y that is a subset of the set Y_0. In particular, this set Y should be consistent. We show that on this set Y, the algorithm will return a vertex cover of the graph G with the desired properties. Suppose that Y contains u U-vertices and l L-vertices, $u \leq k_u$ and $l \leq k_l$.

Let B'_1, ..., B'_t be the blocks in which no vertices are in the set Y. Note that none of these blocks B'_i is in the collection $\mathcal{B}$ since every block in $\mathcal{B}$ has either its U-part or its L-part included in the set Y by step 3. In particular, the weight of each B'_i is smaller than εk_u. Since h is the first index such that $u + |B'_1| + \cdots + |B'_h| > k_u$, we must have

$$u + |B'_1| + \cdots + |B'_{h-1}| \leq k_u$$

Combining this with $|B'_h| \leq \varepsilon k_u$, we get

$$u + |B'_1| + \cdots + |B'_{h-1}| + |B'_h| \leq k_u + \varepsilon k_u < (1 + \varepsilon) k_u$$

Therefore, in the final returned set Y, the number k^*_u of U-vertices in Y is bounded by $(1 + \varepsilon)k_u$. Moreover, since every block has exactly one of its U-part and L-part in Y, the number k^*_l of L-vertices in the final returned set Y is actually smaller than k_l since $k_u + k_l \geq k^*_u + k^*_l$. In consequence, the numbers k^*_u and k^*_l satisfy the condition $\max\{k^*_u/k_u, k^*_l/k_l\} \leq 1 + \varepsilon$.

We still need to prove that in this case the returned set Y is a vertex cover of the graph G. Let $e = [v_i, v_j]$ be any edge in the graph G. If e is an edge in a block B_i, then since every block in G has either its U-part or its L-part in the set Y, one of the vertices v_i and v_j must be contained in the set Y. If e is an inter-block edge, let v_i be a U-vertex in a block B_i and v_j be an L-vertex in a block B_j, $i < j$. If B_i is a block of weight at least εk_u, and if the U-part of B_i is not in Y, then the L-part of B_i must be in Y by our construction, and the L-part of B_i also forces the L-part of U_j into Y. So the vertex v_j is in the set Y. Similarly, if B_j has weight at least εk_u, then one of the vertices v_i and v_j must be in the set Y. The only remaining case is that both blocks B_i and B_j have weight smaller than εk_u. In this case, if B_i is one of the blocks B'_1, ..., B'_h,

then the U-part of B_i, thus the vertex v_i, is in the set Y. On the other hand, if B_j is one of the blocks B'_{h+1}, ..., B'_t, then the L-part of B_j, thus the vertex v_j, is in the set Y. Note that because the sets B'_1, ..., B'_t are topologically sorted as in the DAG D, these are the only two possible cases: if the block B_j is in the collection $\{B'_1, \ldots, B'_h\}$ then so is the block B_i, and if the block B_i is in the collection $\{B'_{h+1}, \ldots, B'_t\}$ then so is the block B_j . In summary, it is always the case that one of the ends of the edge e is in the set Y. Since e is an arbitrary edge in G, we conclude that the set Y is a vertex cover of the graph.

This proves the correctness of the algorithm. We have the following theorem.

Theorem 1. *The algorithm AACI-D runs in time $O(m^2 + 2^{2/\varepsilon}m)$, where m is the number of edges in the graph G. If the input instance $(G; k_u, k_l)$ has a minimum vertex cover of size (k_u, k_l), then the algorithm produces a vertex cover for the graph G of size (k^*_u, k^*_l) satisfying $\max\{k^*_u/k_u, k^*_l/k_l\} \leq 1 + \varepsilon$.*

Proof. The second part of the theorem has been proved by the previous discussion. What remains is to prove the time complexity of the algorithm.

By Lemma 2, step 1 of the algorithm takes time $O(m^2)$. It is also easy to verify that step 2 of the algorithm takes no more than time $O(m^2)$.

The key observation is that the number of blocks in the collection $\mathcal{B}$ is bounded by $2/\varepsilon$. In fact, by our assumption, $k_u \geq k_l$. Therefore, the total number of U-vertices in the graph G is bounded by $k_u + k_l \leq 2k_u$ (note that the size of a minimum vertex cover of the graph G is equal to the number of U-vertices, which is equal to the number of L-vertices in G). Since all blocks in G are disjoint, and each block in $\mathcal{B}$ has weight (i.e., the number of U-vertices) at least εk, the total number of blocks in $\mathcal{B}$ is not larger than

$$(k_u + k_l)/(\varepsilon k_u) \leq 2k_u/(\varepsilon k_u) = 2/\varepsilon$$

Since each enumeration in step 3 takes either the U-part or the L-part of each block in $\mathcal{B}$, the total number of possible sets Y enumerated in step 3 is bounded by $2^{2/\varepsilon}$. Each execution of the body of step 3 obviously takes time no more than $O(m)$. In conclusion, the total running time of step 3 of the algorithm is bounded by $O(2^{2/\varepsilon}m)$. This concludes the theorem. □

5 Putting All Together

We summarize all the discussions given in the previous sections and present the AACI algorithm in Figure 2.

Some explanation is needed.

Let u be a U-vertex of degree larger than k_l. If u is not included in the vertex cover, then all neighbors of u must be in the vertex cover. Since u has more than k_l neighbors, which are all L-vertices, this implies that the number of L-vertices in the vertex cover would exceed the given bound k_l, which is not allowed in the minimum vertex cover of size (k_u, k_l). This justifies step 2: a U-vertex of degree larger than k_l should be directly included in the vertex cover. Similarly, in step 3,

Algorithm AACI$(G, k_u, k_l, \varepsilon)$
input: an instance (G, k_u, k_l) of Min-CVCB, and a constant $\varepsilon > 0$
output: a vertex cover for G, or report no vertex cover of size (k_u, k_l) for G

1. $Y_0 = \emptyset$;
2. **for** each U-vertex u of degree larger than k_l **do**
   $Y_0 = Y_0 \cup \{u\}$; $k_u = k_u - 1$;
3. **for** each L-vertex l of degree larger than k_u **do**
   $Y_0 = Y_0 \cup \{l\}$; $k_l = k_l - 1$;
4. construct an equivalent normalized instance $(G'; k'_u, k'_l)$ by Corollary 1;
5. call algorithm AACI-D on instance $(G'; k'_u, k'_l, \varepsilon)$;
6. **if** step 5 returns a vertex cover Y' for the graph G'
   then construct a desired vertex cover Y for G from Y' and Y_0
   else return ("no vertex cover of size (k_u, k_l) for G").

Fig. 2. The AACI-D Algorithm for normalized instances of Min-CVCB

all L-vertices of degree larger than k_u are directly included in the vertex cover. The running time of steps 1-3 is obviously bounded by $O((n+m)^2)$, where n and m are the number of vertices and number of edges in the graph G.

By Corollary 1, an equivalent normalized instance $(G'; k'_u, k'_l)$ can be constructed in time $O((n+m)^2)$ in step 4.

By Theorem 1, step 5 takes time $O(m^2 + 2^{2/\varepsilon}m)$, which either claims that the graph G' has no constrained minimum vertex cover of size (k'_u, k'_l), or returns a vertex cover Y' of size (k''_u, k''_l) satisfying $\max\{k''_u/k'_u, k''_l/k'_l) \leq 1 + \varepsilon$. By Lemma 4, if the graph G' has no constrained minimum vertex cover of size (k'_u, k'_l), then the graph G has no constrained minimum vertex cover of size (k_u, k_l). On the other hand, if G' has a minimum vertex cover of size (k'_u, k'_l), then by Theorem 1, step 5 of the algorithm returns a vertex cover of size (k''_u, k''_l) satisfying $\max\{k''_u/k'_u, k''_l/k'_l) \leq 1 + \varepsilon$. Now from Corollary 1, step 6 of the algorithm AACI will return a vertex cover of size (k^*_u, k^*_l) for the input graph G satisfying the condition $\max\{k^*_u/k_u, k^*_l/k_l) \leq 1 + \varepsilon$.

This proves the correctness of the algorithm AACI.

The time complexity of the algorithm also follows from the previous discussion. In particular, step 4 of the algorithm takes time $O((n+m)^2)$ by Corollary 1, and step 5 takes time $O((n+m)^2 + 2^{2/\varepsilon}(n+m))$ by Theorem 1.

We summarize these discussions in the following theorem.

Theorem 2. *The algorithm AACI is a polynomial time approximation scheme for the Min-CVCB problem, and its running time is bounded by* $O((n+m)(2^{2/\varepsilon} + n + m))$.

6 Conclusions

In this paper, we have studied the Min-CVCB problem that has direct applications in the area of VLSI manufacturing. Since heuristic algorithms and exact

algorithms may not meet the requirement of industry applications, we studied approximation algorithms for the problem. We developed a polynomial time algorithm with a $1+\varepsilon$ approximate ratio for the problem in the following sense: given an instance $(G;, k_u, k_l)$ for the Min-CVCB, where G is a bipartite graph and looking for a minimum vertex cover of at most k_u U-vertices and at most k_l L-vertices, our algorithm either reports that no such a minimum vertex cover exists, or constructs a vertex cover of k_u^* U-vertices and k_l^* L-vertices in G satisfying the condition $\max\{k_u^*/k_u, k_l^*/k_l\} \leq 1+\varepsilon$. The running time of our algorithm is bounded by $O((n+m)(2^{2/\varepsilon}+n+m))$.

References

1. D. M. Blough, On the reconfigurable of memory arrays containing clustered faults. *Proc. 21th Int. Symp. on Fault-Tolerant Computing* (FTCS'91), (1991), pp. 444-451. .
2. D. M. Blough and A. Pelc, Complexity of fault diagnosis in comparison models. *IEEE Trans.Comput.,41(3)* (1992), pp. 318-323.
3. J. Chen and I. A. Kanj, Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms, *Journal of Computer and System Science 67*, (2003), pp. 833-847.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rovest *Introduction to Algorithms*, McGraw-Hill Book Company, New York, 1992.
5. R. Downey and M. Fellows, *Parameterized Complexity*, Springer, New York, 1999.
6. H. Fernau, and R. Niedermeier, An efficient exact algorithm for constraint bipartite vertex cover, *J. Algorithms 38*, (2001), pp. 374-410.
7. N. Hasan and C. L. Liu, Minimum Fault Coverage in Reconfigurable Arrays, *Proc. 18th Int. Symp. on Fault-Tolerant Computing* (FTCS'88), (1988), pp. 348-353.
8. N. Hasan and C. L. Liu, Fault covers in reconfigurable PLAs. *Proc. 20th Int. Symp. on Fault-Tloerant Computing* (FTCS'90), (1990), pp. 166-173.
9. S.-Y. Kuo and W. Fuchs, Efficient spare allocation for reconfigurable arrays. *IEEE Des. Test 4* (1987), pp. 24-31.
10. L. Lovasz and M. D. Plummer, *Matching Theory*, *Annals of Discrete Mathematics Vol.29*, North-Holland, Amsterdam, 1986.
11. C. P. Low and H. W. Leong, A new class of efficient algorithms for reconfiguration of memery arrays, *IEEE Trans. Comput.45(1)*, (1996), pp. 614-618.
12. N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA, 1980.
13. M. D. Smith and P. Mazumder, Generation of minimal vertex cover for row/column allocation in self-repairable arrays, *IEEE Trans.Comput. 45*, (1996), pp. 109-115.

Author Index

Ajtai, Miklós 13
Akutsu, Tatsuya 573
Andersen, Reid 1

Bach, Eric 632
Bai, Xi 148
Barmpalias, George 89

Cai, Zhun 189
Cao, Yi 136
Chan, Joseph Wun-Tat 416
Chen, Jianer 692, 760
Chen, Jing 46
Chen, Zhenyu 386
Cheng, Qi 296
Chung, Fan 1
Cooper, S. Barry 199

Diedrich, Florian 34
Ding, Decheng 595
Dom, Michael 680
Duan, Zhenhua 521

Fan, Yun 212
Feng, Keqin 318
Feng, Rongquan 159
Feng, Wangsen 646
Feng, Ze 284
Fusco, Giordano 632

Gao, Huang-Ming 274
Graham, Ronald L. 284
Grubba, Tanja 100
Guo, Jiong 680
Gupta, Sushmita 354

Harren, Rolf 34
Hinkelmann, Markus 486
Hori, Yoshiaki 362
Hou, Haiyang 58
Hsieh, Sun-Yuan 274, 750
Hu, Xiaoming 171
Huang, Shangteng 171

Imai, Katsunobu 511
Iwamoto, Chuzo 511

Jakoby, Andreas 330, 486
Jansen, Klaus 34

Kawaguchi, Akifumi 584
Kerenidis, Iordanis 306
Kim, Kwangsoo 533
Kobayashi, Satoshi 398
Kong, Fanyu 189
Kugimoto, Yoshinori 499
Kurtz, Stuart A. 542

Lam, Tak-Wah 416
Lewis, Andrew E.M. 89
Li, Angsheng 79
Li, Daxing 148, 189
Li, Minming 284
Li, Xiang 408
Li, Xin 624
Li, Yong 440
Li, Zhimin 408
Lingas, Andrzej 256
Liśkiewicz, Maciej 330
Liu, Feng 318
Liu, Tian 624
Liu, Xingwu 715
Liu, Yunlong 692
Lu, Enyue 46

Mak, Kin-Sum 416
Mizuki, Takaaki 499
Morita, Kenichi 511
Morizumi, Hiroki 605
Morsy, Ehab 342
Mukhopadhyay, Debapriyay 738
Murray, Elizabeth 296

Nagamochi, Hiroshi 342, 573, 584
Nakano, Shin-ichi 115, 428
Niedermeier, Rolf 680

Pan, Li 222
Pei, Shi-Hui 181
Peng, Han 624
Peng, Sheng-Lung 244
Poon, Chung Keung 659
Pu, Juhua 715

Qian, Liyan 624
Qu, Wanling 646

Rao, M.V. Panduranga 450
Reischuk, Rüdiger 330
Ryoo, Hong Seo 533

Saha, Indranil 738
Sakurai, Kouichi 362
Schindelhauer, Christian 330
Shoudai, Takayoshi 67
Simon, Janos 542
Sone, Hideaki 499
Soskova, Mariya Ivanova 89, 199
Stechert, Peer 486
Sun, He 659
Sun, Hongtao 624

Tan, Xuehou 262
Tao, Zhihong 386
Tarui, Jun 128, 605
Teng, Shang-Hua 554
Thöle, Ralf 34
Thomas, Henning 34
Thurley, Marc 703
Tian, Cong 521
Tîrnăucă, Cristina 398
Trahtman, A.N. 234

Uehara, Ryuhei 115, 428
Uno, Takeaki 115, 428

Wahlen, Martin 256
Wang, Hanpin 646
Wang, Jianxin 692, 760
Wang, Jiexun 573
Wang, Jin 148
Wang, Shumei 222
Wang, Yufeng 362
Wang, Zhicheng 222
Wei, Gang 222
Weihrauch, Klaus 595
Wong, Prudence W.H. 416
Wu, Hongfeng 159
Wu, Yongcheng 595

Xia, Mingji 566
Xu, Daoyun 616
Xu, Jin 624
Xu, Ke 624
Xu, Xiaoshuang 760
Xu, Yatao 100
Xu, Zhiwei 715

Yamasaki, Hitoshi 67
Yang, Bing 46
Yang, Boting 136
Yang, Shih-Cheng 274
Yang, Yi-Chuan 244
Yao, Andrew C.C. 462, 474
Yao, Frances F. 284, 462, 474
Yi, Jin 374
Yoneda, Harumasa 511
Yong, Jun-Hai 440
Yu, Jia 148, 189

Zhang, Guochuan 58
Zhang, Li'ang 646
Zhang, Peng 670, 728
Zhang, Qingshun 616
Zhang, Wenhui 374
Zhao, Hong-Wei 181
Zhao, Liang 573
Zhao, Weidong 222
Zhao, Wenbo 670
Zhao, Yingchao 554
Zhao, Yong-Zhe 181
Zhao, Yunlei 462, 474
Zheng, S.Q. 46
Zhou, Conghua 386
Zhu, Jiaqi 624