Lecture Notes in Computer Science 1767
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Giancarlo Bongiovanni   Giorgio Gambosi
Rossella Petreschi (Eds.)

Algorithms and Complexity

4th Italian Conference, CIAC 2000
Rome, Italy, March 1-3, 2000
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Giancarlo Bongiovanni
Rossella Petreschi
University of Rome "La Sapienza", Department of Computer Science
Via Salaria 113, 00198 Rome, Italy
E-mail: {bongiovanni/petreschi}@dsi.uniroma1.it

Giorgio Gambosi
University of Rome "Tor Vergata", Department of Mathematics
Via della Ricerca Scientifica 1, 00133 Rome, Italy
E-mail: gambosi@mat.uniroma2.it

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Algorithms and complexity : 4th Italian conference ; proceedings / CIAC 2000, Rome, Italy, March 1-3, 2000. Giancarlo Bongiovanni . . . (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1767)
ISBN 3-540-67159-5

CR Subject Classification (1998): F.2, F.1, E.1, I.3.5, G.2

ISSN 0302-9743
ISBN 3-540-67159-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 10719716 06/3142 5 4 3 2 1 0

Preface

The papers in this volume were presented at the Fourth Italian Conference on Algorithms and Complexity (CIAC 2000). The conference took place on March 1-3, 2000, in Rome (Italy), at the conference center of the University of Rome \La Sapienza".

This conference was born in 1990 as a national meeting to be held every three years for Italian researchers in algorithms, data structures, complexity, and parallel and distributed computing. Due to a signi cant participation of foreign reaserchers, starting from the second conference, CIAC evolved into an international conference.

In response to the call for papers for CIAC 2000, there were 41 submissions, from which the program committee selected 21 papers for presentation at the conference. Each paper was evaluated by at least three program committee members. In addition to the selected papers, the organizing committee invited Giorgio Ausiello, Narsingh Deo, Walter Ruzzo, and Shmuel Zaks to give plenary lectures at the conference.

We wish to express our appreciation to all the authors of the submitted papers, to the program committee members and the referees, to the organizing committee, and to the plenary lecturers who accepted our invitation.

March 2000

Rossella Petreschi
Giancarlo Bongiovanni
Giorgio Gambosi

Organizing Committee:

G. Bongiovanni (Chair, Rome)
T. Calamoneri (Rome)
A. Clementi (Rome)
S. De Agostino (Rome)
I. Finocchi (Rome)
G. Gambosi (Rome)
P. Penna (Rome)
P. Vocca (Rome)

Program Committee:

R. Petreschi (Chair, Rome)
P. Crescenzi (Florence)
S. Even (Haifa)
L. Gargano (Salerno)
G. Italiano (Rome)
F. Luccio (Pisa)
Y. Manoussakis (Paris)
J. Nesetril (Prague)
S. Olariu (Norfolk)
R. Silvestri (L'Aquila)
R. Tamassia (Providence)
P. Widmayer (Zurich)

Additional Referees:

R. Battiti
C. Blundo
D. Boneh
G. Bongiovanni
M. Bonuccelli
T. Calamoneri
Y. Censor
M. Cieliebak
A. Clementi
P. DArco
S. De Agostino
G. De Marco
C. Demetrescu
M. Di Ianni
S. Eidenbenz
E. Feuerstein
I. Finocchi
R. Friedman
C. Galdi
A. Itai
M. Kaminsky
M. Leoncini
Z. Liptak
P. McKenzie
J. Marion
A. Massini
B. Masucci
A. Monti
M. Napoli
P. Penna
G. Pucci
A. Rescigno
M. Ruszinko
K. Schlude
H. Shachnai
C. Stamm
A. Sterbini
M. Sviridenko
U. Vaccaro
C. Verri
P. Vocca
M. Zapolotsky

Sponsored by:

Banca di Credito Cooperativo di Roma
Dipartimento di Scienze dell'Informazione, Universita di Roma \La Sapienza"
European Association for Theoretical Computer Science
Gruppo Nazionale per l'Informatica Matematica
Italsoft Ingegneria dei Sistemi S.r.l.
Universita degli Studi di Roma \La Sapienza"
Universita degli Studi di Roma \Tor Vergata"

Table of Contents

On Salesmen, Repairmen, Spiders, and Other Traveling Agents

Giorgio Ausiello, Stefano Leonardi, and Alberto Marchetti-Spaccamela

Dipartimento di Informatica Sistemistica, Universita di Roma \La Sapienza",
via Salaria 113, 00198-Roma, Italia.

Abstract. The Traveling Salesman Problem (TSP) is a classical problem in discrete optimization. Its paradigmatic character makes it one of the most studied in computer science and operations research and one for which an impressive amount of algorithms (in particular heuristics and approximation algorithms) have been proposed. While in the general case the problem is known not to allow any constant ratio approximation algorithm and in the metric case no better algorithm than Christo des' algorithm is known, which guarantees an approximation ratio of 3/2, recently an important breakthrough by Arora has led to the de nition of a new polynomial approximation scheme for the Euclidean case. A growing attention has also recently been posed on the approximation of other paradigmatic routing problems such as the Travelling Repairman Problem (TRP). The altruistic Travelling Repairman seeks to minimimize the average time incurred by the customers to be served rather than to minimize its working time like the egoistic Travelling Salesman does. The new approximation scheme for the Travelling Salesman is also at the basis of a new approximation scheme for the Travelling Repairman problem in the euclidean space. New interesting constant approximation algorithms have recently been presented also for the Travelling Repairman on general metric spaces. Interesting applications of this line of research can be found in the problem of routing agents over the web. In fact the problem of programming a \spider" for e ciently searching and reporting information is a clear example of potential applications of algorithms for the above mentioned problems. These problems are very close in spirit to the problem of searching an object in a known graph introduced by Koutsoupias, Papadimitriou and Yannakakis [14]. In this paper, motivated by web searching applications, we summarize the most important recent results concerning the approximate solution of the TRP and the TSP and their application and extension to web searching problems.

1 Introduction

In computer applications involving the use of mobile virtual agents (sometimes called "spiders") that are supposed to perform some task in a computer network, a fundamental problem is to design routing strategies that allow the agents to complete their tasks in the most e cient way ([3],[4]). In this context a typical scenario is the following: an agent generated at node 0 of the network searches a portion of the web formed by n sites, denoted by $1, \ldots, n$, looking for an information. At each site i it is associated a probability p_i that the required information is at that site. The distance from site i to site j is given by a metric function $d(i, j)$. The aim is to find a path $\pi(1), \ldots, \pi(n)$ that minimizes the quantity $\sum_{i=1}^{n} p_i \sum_{k=1}^{i} d(k-1, k)$. In [14] this problem is called the Graph Searching problem (GSP in the following). The GSP is shown to be strictly related to the Travelling Repairman Problem (TRP), also called the Minimum Latency Problem (MLP), in which a repairman is supposed to visit the nodes of a graph in a way to minimize the overall waiting time of the customers sitting in the nodes of the graph. More precisely in the TRP we wish to minimize the quantity $\sum_{i=1}^{n} \sum_{k=1}^{i} d(k-1, k)$.

The Minimum Latency problem is known to be MAX-SNP-hard for general metric spaces as a result of a reduction from the TSP where all the distances are either 1 or 2, while it is solvable in polynomial time for the case of line networks [1].

In this paper we present the state of the art of the approximability of the TRP and present the extension of such results to the Graph Searching problem. The relationship between GSP and TRP is helpful from two points of view. In some cases, in fact, an approximation preserving reduction from GSP to TRP can be established [14] under the assumption that the probabilities associated with the vertices are polynomially related, by replacing every vertex with a polynomial number of vertices of equal probability. This allows to apply the approximation algorithms developed for TRP to GSP. Among them, particularly interesting are the constant approximation algorithms for general metric spaces given by Blum et al. [9] and Goemans and Kleinberg [13], later improved in combination with a result of Garg [11] on the k-MST problem .

More recently a quasi-polynomial $O(n^{O(\log n)})$ approximation scheme for tree networks and Euclidean spaces has been proposed by Arora and Karakostas [5]. This uses the same technique as in the quasi-polynomial approximation scheme of Arora for the TSP [6]. The case of tree networks seems particularly interesting since one is often willing to run the algorithm on a tree covering a portion of the network that hopefully contains the required information. In the paper we also show how to extend approximation schemes for the TRP to the Graph Searching problem.

In conclusion, the Graph Searching problem, beside being an interesting problem "per se", motivated by the need to design efficient strategies for moving "spiders" in the web, has several interesting connections with two of the most intriguing and paradigmatic combinatorial graph problems, the Traveling Repairman problem and the Traveling Salesman problem. Therefore the study of the former problem naturally leads to the study of the results obtained for the latter problems, which may be classified among the most interesting breakthrough achieved in the recent history of algorithmics.

This paper is organized as follows: in section 2 we formally define the GSP and in section 3 we review approximation algorithms for the TRP. In sections 4 and 5 we present approximation algorithms and approximation schemes for the GSP.

2 Preliminaries and Notation

The Graph Searching Problem (GSP), introduced by Koutsoupias, Papadimitriou and Yannakakis [14], is defined on a set of n vertices $V = \{1, \dots, n\}$ of a metric space M, plus a distinguished vertex of M at which the travelling agent is initially located and will return after the end of the tour. The starting point is also denoted as the root and indicated as vertex 0. A metric distance $d(i, j)$ defines the distance between any pair of vertices i, j. With every vertex i is also associated a probability or weight $w_i > 0$ (the vertices with weight 0 are simply ignored). We assume that the object is in exactly one site for which $\sum_{i=1}^{n} w_i = 1$. A solution to the GSP is a permutation $\pi(1), \dots, \pi(n)$ indicating the tour to be followed. The distance of vertex i to the root along the tour is given by $l(i) = \sum_{j=1}^{i} d(\pi(j-1), \pi(j))$. The objective of the GSP is to minimize the expected time spent to locate the object in the network, namely $\sum_{i=1}^{n} w_i l(i)$.

We will measure the performance of algorithms by their approximation ratio, that is the maximum ratio over all input instances between the cost of the algorithm's solution and the optimal solution.

3 Approximation Algorithms for the TRP

In this section we will present the approximation algorithms developed in the literature for the TRP that was first introduced in [1]. These results are relevant to the solution of the GSP that reduces to the TRP when all the vertices have equal probability or are polynomially related.

In the case of line networks the problem is polynomial.

Theorem 1. *[1] There exists a $O(n^2)$ optimal algorithm for the TRP on line networks.*

The algorithm numbers the root with 0, the vertices at the right of the root with positive integer numbers, the vertices at the left of the root with negative integer numbers. By dynamic programming the algorithm stores for every pair of vertices $(-l, r)$ with $l, r > 0$, (i) the optimal path that visits the vertices of $[-l, r]$ and ends at $-l$ and (ii) the optimal path that visits the vertices of $[-l, r]$ and ends at r. The information at point (i.) is computed in $O(1)$ time by selecting the best alternative among (a.) the path that follows the optimal path for $(-(l-1), r)$, ends at $-(l-1)$ and then moves to l, and (b.) the optimal path for $(-l, r-1)$ that ends at $r-1$ and then moves first to r then to l. The information at point (ii.) is analogously computed in $O(1)$ time.

The TRP is known to be solvable in polynomial time beyond line networks only for trees with bounded number of leaves [14]. Whether the TRP is polynomially time solvable or NP-hard for general tree networks is still a very interesting open problem.

The rst constant factor approximation for the TRP on general metric spaces and on tree networks has been presented by Blum et al [9]. The authors introduce the idea to concatenate a sequence of tours to form the nal solution. The algorithm proposed by the authors computes for every $j = 1; 2; 3 : : :$ a tree T_j of cost at most 2^{j+1} spanning the maximum number of vertices. This procedure is repeated until all the vertices have been included in a tree. The nal tour is obtained by concatenating a set of tours obtained by depth rst traversing trees T_j, $j = 1; 2; 3; : : :$. Let m_j be the number of vertices spanned by T_j. Let S_j be the set of vertices of T_j. Consider a number i such that $m_j \quad i \quad m_{j+1}$. We can state that the i-th vertex visited in the optimal tour has latency at least 2^j. On the other end, the latency of the i-th vertex visited in the algorithm's tour is at most $8 \quad 2^j$. This is because the latency of the i-th vertex in the tour of the algorithm is at most $2(\sum_{k<j} 2^{k+1} + 2^{j+1}) \quad 8 \quad 2^j$. Assume that it is available a c approximation algorithm that is able to nd a tree of minimum cost that spans k vertices of the network, this can be easily turned through a binary search procedure into a c-approximation algorithm for the problem of nding a tree of bounded cost that maximizes the number of vertices that are spanned. This immediately results in an $8c$ approximation algorithm for the TRP.

In a tree network the problem of nding a tree of k vertices of minimum cost is polynomial time solvable using a dynamic programming algorithm described in the paper of Blum et al [9]. The algorithm rst transforms the tree into a binary tree by replacing every vertex v of degree higher than 2 into a binary tree with edges of cost 0 and every leaf connected to at most 2 children of v with edges weighted by the cost of the edges from v to the corresponding children. The procedure computes for any vertex of the graph, for every integer j between 1 and k, for every $i = 0; : : : ; j$, the minimum cost tree that collects i vertices on the left subtree and $j - i$ vertices on the right subtree. This procedure can be clearly implemented in polynomial time. This implies an 8-approximation algorithm for the TRP on tree networks.

When the paper [9] appeared, no constant approximation algorithm for the k-MST problem on general metric spaces was known. A constant approximation algorithm for the TRP problem on general metric spaces was then obtained by applying the so called (;) TSP approximator. An (;) TSP approximator is an algorithm that given bounds and L, an n-point metric space M and a starting point p, nds a tour starting at p of length at most L which visits at least $(1 - \quad)n$ vertices when there exists a tour of length L which visits $(1 - \quad)n$ vertices. The existence of an (;) TSP approximator ensures the existence of an 8 approximation algorithm for the TRP. A $(3; 6)$ and a $(4; 4)$ TSP approximator were proposed in [9], a $(2; 4)$ and a $(2; 3)$ TSP approximator were later proposed by Goemans and Kleinberg in the paper [13].

The paper of Goemans and Kleinberg also presents a new technique to select a sequence of tours of growing length to concatenate to form a solution. The procedure proposed by Goemans and Kleinberg computes for every number j from 1 to n the tour T_j of minimum length that visits j vertices. Let d_j be the length of tour T_j. The goal is to select values $j_1; : : : ; j_m = n$ in order to minimize the latency of the nal tour. Let p_i be the number of new vertices visited during tour i. Since the number of vertices discovered up to the ith tour is certainly no smaller than j_i the following claim of [13] holds

$$\sum_{i=1}^{m} p_i d_i \qquad \sum_{i=1}^{m} (j_i - j_{i-1}) d_i.$$

It follows that for a number of vertices equal to $\sum_{k=1}^{i} p_k - j_i$ we sum a contribution at most d_k on the left side of the equation while a contribution larger than d_k on the right side of the equation. Moreover, each tour T_i is traversed in the direction that minimizes the total latency of the vertices discovered during tour T_i. This allows to rewrite the total latency of the tour obtained from concatenating $T_{j_1}, \ldots, Tj_m$ as:

$$\sum_i (n - \sum_{k=1}^{i} p_k) d_{j_i} + \frac{1}{2} \sum_i p_i d_{j_i}$$
$$\sum_i (n - j_i) d_{j_i} + \frac{1}{2} \sum_i (j_i - j_{i-1}) d_{j_i}$$
$$= \sum_i (n - \frac{j_{i-1} + j_i}{2}) d_{j_i}.$$

The formula above allows to rewrite the total latency of the algorithm only in terms of the indices j_i and of the length d_{j_i}, independently from the number of new vertices discovered during each tour. A complete graph of n vertices is then constructed in the following way. Arc (i, j) is turned into a directed edge from $min(i, j)$ to $max(i, j)$. Arc (i, j) has length $(n - \frac{i+j}{2})d_j$. The algorithm computes a shortest path from node 0 to node n. Assume that the path goes through vertices $0 = j_0 < j_1 <, \ldots, < j_m = n$. The tour is then obtained by concatenating $T_{j_1}, \ldots, T_{j_m}$.

The obtained solution is compared against the following lower bound OPT $\sum_{k=1}^{n} \frac{d_k}{2}$. This lower bound follows from the observation that the kth vertex cannot be visited in any optimal tour before $d_k/2$. The approximation ratio of the algorithm is determined by bounding the maximum over all the possible set of distances $d_1, \ldots, d_n$ of the ratio between the shortest path in G_n and the lower bound on the optimal solution. This value results to be smaller than 3.5912 thus improving over the ratio of 8 in [9].

Theorem 2. *[13] Given a c approximation algorithm for the problem of nding a tour of minimum length spanning k vertices on a speci c metric space, then there exists an $3.5912c$ approximation ratio for the TRP on the same metric space.*

The method described above allows to obtain a 3.5912 approximation for tree networks. For general metric spaces, a 3 approximation algorithm for the k-MST problem and for the problem of devising a tour of minimum cost spanning

k vertices has been later proposed by Garg [11]. This allows to obtain a 10.7796 approximation algorithm for the TRP on general metric spaces. This bound can be furtherly improved by applying the more recent 2.5 approximated k-MST algorithm of Arya and Kumar[7]. We will describe the algorithm of [11] for the k-MST in the following section where we study the extension of the algorithm for the TRP to the GSP.

4 Approximation Algorithms for the GSP

In this section we will study the extension of the algorithms for the TRP to the GSP.

The algorithm for the TRP on line networks can be extended to provide a polynomial time algorithm for the GSP on line networks. The dynamic programming algorithm presented in the previous section is simply modi ed in order to increase the cost of a solution by the latency of a vertex weighted by its probability rather than just by the latency of a vertex.

As we mentioned in the introduction, the GSP problem has been introduced by Koutsoupias, Papadimitriou and Yannakakis [14]. In that paper the authors show a simple reduction from the GSP to the TRP under some restrictive conditions. They show that the metric GSP can be reduced to the metric TRP under the assumption that all the weights/probabilities are *rational numbers with small coe cients and common denominators*. This assumption allows to split every vertex into a polynomial number of vertices with weight equal to the common denominator of all the weights of the vertices of the graph. If two vertices in the instance of the TRP derive from the splitting of the same vertex in the instance of the GSP, their internode distance is 0, if the two vertices derive from two di erent vertices in the instance of the GSP, say i and j, their distance is $d(i,j)$. A solution to the instance of the TRP obtained from an instance of the GSP can be easily turned into a solution of equal cost to the original GSP instance, since all the vertices at distance 0 in the TRP can be visited at the same time.

Unfortunately, this reduction does not apply to the general case. In this section we will consider algorithms for the general case of the metric GSP and of the GSP on tree networks. When trying to extend the general approach for the TRP to the GSP, we need to solve two kind of problems: (i.) Find a sequence of tours to be concatenated to obtain the nal tour; (ii.) Compute every tour to be concatenated. We will see that the solution of Goemans and Kleinberg for point (i.) seems not to be easily extendible to the GSP, and that the computation of every tour to be concatenated can be strictly more di cult than for the TRP.

Kleinberg and Goemans propose to compute for every $k = 1, \ldots, n$ a tour of minimum cost that spans k vertices. The application of this approach to the GSP requires to nd a tour of minimum cost that spans at least a given weight for every possible amount of weight. This approach when extended to the GSP, requires to compute the minimum cost tour that covers an amount of weight i for every $i = 1, \ldots, W = \sum_j w_j$. This clearly results in a pseudo-polynomial time algorithm. Alternatively we can think of partitioning the interval $[0; W]$ into a polynomial number of intervals of larger size x, $[ix; (i+1)x]$, $i = 0; :::; dW=x-1e$. The drawback of a similar solution is a weaker lower bound. Let $w = \min_j w_j$ be the minimum weight of a vertex. We can state a lower bound of OPT $\sum_{j=1}^{m} \min_j w \frac{d_j}{2}$ over the optimal solution, but we cannot state that the optimal solution will cover the $(j+1)$-th amount of weight x before time $d_j=2$. However in this section we will follow the approach of Blum et al [9], that repeatedly nds a tree of exponentially increasing length spanning the biggest amount of weight until the whole weight has been collected. Their result on the relationship between TRP and k-MST can be easily extended to the GSP problem; we de ne the W-MST problem as the problem of nding a tree of minimum cost that covers a weigh of at least W.

Theorem 3. *[9] Given a c-approximation algorithm for W-MST problem there exists a $8c$ approximation algorithm for the GSP.*

Let W_l be the total weight collected in the l-th tour of length at most 2^l. Let $_l = W_l - W_{l-1}$. It is possible to see that any algorithm will pay for the weight that is collected between W_{l-1} and W_l a latency of at least 2^l. We then obtain an algorithm with approximation ratio $8c$ if we have a c-approximation algorithm for nding a tree spanning a maximum amount of weight with cost bounded by a given value L, or alternatively an algorithm for nding a tree of minimum cost that covers at least a given weight, say W.

Such algorithms are not known in literature for both tree networks and general metric spaces. The problem of nding a tree of minimum cost spanning a weight of at least W is already NP-hard for tree networks. The reduction is from Knapsack. Consider n items where the generic item i has cost c_i and bene t w_i. The corresponding instance of the GSP is obtained by constructing a star network of n leaves where the root is the center of the star, and every leaf i has weight w_i and it is connected to the center with an edge of cost c_i. The problem of nding a tree of maximum weight of bounded cost is clearly NP-hard as it is NP-hard the problem of nding a minimum cost tree that spans a weight of at least W. In the next section we will show how to provide a fully polynomial time approximation scheme for this problem on tree networks.

In the rest of this section we will show how to extend a constant approximation algorithm for the k-MST problem to the W-MST problem.

Theorem 4. *There exists a constant approximation algorithm for the W-MST problem.*

For the sake of the exposition, we limit ourself to show the extension of the 5 approximation algorithm of Garg for the k-MST problem to the case in which the goal is to collect a weight of at least W. In [11], a 3 approximation algorithm is also presented, that improves over the previous constant approximation algorithm by Blum, Ravi and Vempala [10], while an improved approximation based on the same techniques has been later proposed by Arya and Kumar [7]. These algorithms are based on the Primal-Dual method developed by Agrawal, Klein and Ravi [2] and by Goemans and Williamson [12] to design forests of minimum cost satisfying various constraints.

In the following we will highlight the main variation to the algorithm and to the analysis of [11] to extend the 5 approximation to the W-MST problem. We consider the problem in the case the vertex furthest to the root is part of the optimal solution. An algorithm for the general problem is then obtained by trying every possible vertex and selecting the best solution. It is well known that the primal-dual method uses the dual of a relaxation of the linear program formulation of the problem as a guide for the algorithm and the analysis. We need to introduce the standard notation for the primal-dual method. We denote by S the generic subset of V and by $\delta(S)$ the set of edges with exactly one endpoint inside S. Let E be the edge set of the graph, e the generic edge (i,j) of E, and $c_e = d(i,j)$ its cost. In the linear programming formulation of the problem a variable $x_e \in \{0,1\}$, $\forall e \in E$, indicates if edge e is part of the tree, a variable $x_v \in \{0,1\}$, $v \in V$, indicates if vertex v is spanned by the tree. The starting point of the tour is the root r of the tree.

The linear programming formulation of the W-MST problem after the relaxation of the integrality constraints on variables x_e and x_v is as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e\in E} c_e x_e \\
\sum_{e\in\delta(S)} x_e &\ge x_v \quad (\forall v, S : v \in S \subseteq V - \{r\}) \\
\sum_{v\in V} x_v w_v &= W \\
x_v &\le 1 \quad (\forall v \in V) \\
x_v &\ge 0 \quad (\forall v \in V) \\
x_e &\ge 0 \quad (\forall e \in E)
\end{aligned}$$

In the dual formulation a variable $y_{v,S}$ is associated to every constraint of the first set, a variable p to the second constraint and a variable p_v to every constraint of the third set. The dual formulation is as follows:

$$\begin{aligned}
\text{maximize} \quad & p\,W - \sum_{v\in V} p_v \\
\sum_{S: v\in S} y_{v,S} + p_v &\ge p w_v \quad (\forall v \in V) \\
\sum_{S: e\in\delta(S)} y_{v,S} &\le c_e \quad (\forall e \in E) \\
p_v &\ge 0 \quad (\forall v \in V) \\
y_{v,S} &\ge 0 \quad (\forall v, S : v \in S \subseteq V - \{r\})
\end{aligned}$$

De ne in a way similar to [11] the potential $_v$ of vertex v as $_v = \sum_{S:v \in S} y_{v;S}$. Observe that if $_v \quad pw_v$ then $p_v = 0$, else $p_v = pw_v - {}_v$. From the previous observation it is possible to prove that in an optimal solution p has value between the k_W-th and $(k_W + 1)$-th smallest ratio $\frac{v}{w_v}$ where k_W is the smallest integer such that $\sum_{i=1}^{k_W} w_i \quad W$. The optimal solution of the dual problem can be thought of as an assignment to the dual variables such that the sum of the rst k_W potentials is maximized. By the duality theorem it also follows that the sum of the rst k_W potentials is a lower bound to the optimal solution of the primal problem.

The primal-dual algorithm will construct a solution with cost bounded by twice the sum of the rst k_W potentials. This solution will be completed to cover a weight at least W with an extra cost bounded by a constant factor times the optimal value.

The problem is then reduced to nding a feasible assignment of potentials (v) such that the sum of the rst k_W potentials is maximized. An assignment of potentials is feasible if there exists an assignment of variables $y_{v;S}$ for which $\sum_{S;v \in S:e \in (S)} y_{v;S} \quad c_e$ and for any vertex v, $(v) \quad \sum_{S:v \in S} y_{v;S}$.

The primal-dual algorithm is run with an initial potential pw_v assigned to every vertex v apart from the root to which it is assigned a potential 0. Every subset of vertices not containing the root has associated a variable y_S. The assignment of variables y_S satis es at any time, for every vertex e, $\sum_{S:e \in (S)} y_S \quad c_e$. If for a vertex e the inequality holds with equality then the edge is said to be *tight*. At any step of the algorithm the set of vertices V is partitioned into a set of active and inactive components. A component is active if it has a positive potential and it does not contain the root, otherwise it is inactive.

The algorithm simultaneously increases for every active component S the variable y_S and decreases its potential until either the potential is 0 or one of the constraints on one of the edges is tight. If the constraint for edge $e = (i;j)$ is tight, the active components containing edges i and j are merged with potential equal to the sum of the residual potentials of the two components. The two components are made inactive, while the new component is active unless it contains the root. The set of tight edges at any stage forms a forest whose trees de ne the set of components at that stage. The procedure halts when all the components are inactive, that is the residual potential of all the components not containing the root is 0.

The tree spanning the component of the root when the algorithm halts is denoted by T_p. T_p is then pruned to remove every edge that connects to T_p a subtree that spans an inactive component at some stage of the algorithm. Let T_p^0 be the tree obtained after the pruning phase. The set of initial potentials does not necessarily form a feasible assignment of potentials. We can follow [11] in showing that a su cient condition for the set of potentials to be feasible is that the componenent containing the root has zero potential when the algorithm halts. An assignment of the potentials that satis es this requirement can be obtained by decreasing the potential of a vertex such that all the components containing that vertex have non-zero potential. We can reduce the potential of every such vertex until a component containing the vertex has 0 residual potential. Such procedure is repeated until the component containing the root has 0 potential.

Denote by $cost(T)$ the cost of tree T, and by W_p the weight spanned by the vertices of T_p. The primal-dual method ensures that the cost of T'_p is at most twice the sum of the potentials of the vertices of T'_p, namely $cost(T'_p) = \sum_{e \in T'_p} c_e \quad 2 \sum_{v \in T'_p} \ (v)$. The sum of the smallest k_W potentials is also a lower bound on the optimal tree spanning a weight of at least W. The vertices in T'_p are the only vertices in the graph with ratio $(v)/w_v < p$, then the sum of the potentials of the vertices of the tree T'_p plus $p(W - W_p)$ is also a lower bound on the optimal solution.

We select the highest value of p such that $\sum_{v \in T'_p} w_v = W_p \quad W$. We also run the algorithm for a value $p +$, thus obtaining a tree T'_{p+} for which it holds $W < W_{p+}$. A rst solution is obtained as follows. Consider the tour obtained by traversing twice T'_{p+} . We select the minimum cost path on this tour that collects a weight of at least $W - W_p$. Such path has cost bounded by $\frac{W - W_p}{W_{p+} - W_p} 2\, cost(T'_{p+}\ )$. We consider a rst solution obtained from tree T'_p, the path selected out of T'_{p+} , and an edge that joins this path to the root.

Denote by OPT the cost of the optimal tour. Remind that OPT is lower bounded by the maximum distance from a vertex to the root. The cost of the rst solution is bounded by

$$cost(T'_p) + \frac{W - W_p}{W_{p+} - W_p} 2 cost(T'_{p+}\ ) + OPT.$$

The second solution is obtained from T'_{p+} . Following the analysis of [11], we write the two following lower bounds on the optimal solution:

$$OPT \quad \sum_{v \in T'_p} {}_p(v) + p\,(W - W_p);$$

$$OPT \quad \sum_{v \in T'_{p+}} {}_{p+} \ - (p + \ )(W_{p+} - W_p).$$

We can write:

$$cost(T'_p) \quad 2\,OPT - 2\,p\,(W - W_p);$$

$$cost(T'_{p+}\ ) \quad 2\,OPT + 2(p + \ )(W_{p+} - W_p);$$

from which it follows that the smallest among the two solutions is at most $5\,OPT$.

By combining the above analysis with Theorem 3 we obtain the following Corollary.

Corollary 1. *There exists a 40 approximation algorithm for the GSP de ned in a general metric space.*

5 Approximation Schemes for the GSP

In this section we show under what conditions the GSP allows approximation schemes. Also in this case we do so by extending to GSP similar results obtained for TRP. In particular we first present the main ideas of [5] that shows how to construct an approximation scheme for the TRP in the case of tree metric and Euclidean metric whose running time is quasi polynomial; we will then show an approximation preserving reduction that allows us to obtain similar results for the GSP. As it is done in previous papers on the TRP, the algorithm of [5] finds a low latency tour by joining paths; in this case the algorithm decides at the beginning how many nodes are in each path and then uses dynamic programming for computing this set of paths.

In order to reduce the cost of dynamic programming the authors first show that distances between nodes can be rounded without affecting the approximation; namely, given an instance of the TRP such that the minimum internode distance is 1 and the maximum internode distance is d_{max} it is possible to round internode distances in such a way that the minimum internode distance is 1 and the maximum distance is cn^2/ε, where c is a constant. Given a tour T the rounding affects the contribution of each node to the latency of T by a value less than $d_{max}\varepsilon/n$; since d_{max} is a lower bound on the optimum it follows that the rounding affects the value of T by a factor of ε. The second idea is to break the optimal tour in k segments, $k = O(\log n/\varepsilon)$, T_i each one with a determined number of nodes; the number of nodes in segment i is given by

$$n_i = \lceil (1+\varepsilon)^{k-1-i} \rceil; \quad i = 1, 2, \ldots, k-1$$

$$n_k = \lceil 1/\varepsilon \rceil .$$

Let T_i be the length of T_i; clearly $\sum_{i=1}^{j-1} T_i$ is a lower bound on the latency of any node in segment j. It follows that a lower bound on the optimum latency L^* is given by:

$$L^* \geq \sum_{j=1}^{k} \Big(n_j \sum_{i=1}^{j-1} T_i\Big) = \sum_{i=1}^{k} \Big(\sum_{j>i} n_j\Big) T_i .$$

Now replace segment T_i, $i < k$, with a minimum traveling salesman tour through the same set of nodes. In this way both the lenght of the segment and the latency of nodes in subsequent segments cannot increase; the latency of nodes in T_i can increase by at most $n_i T_i$. Repeating this replacement for all segments but the last one the increase of the latency is at most

$$\sum_{i=1}^{k-1} n_i T_i .$$

Observing that $\sum_{j>i}^{k} n_j \geq n_i/\varepsilon$ it follows that the new latency is at most $(1+\varepsilon)L^*$. Note that if the above approach is applied using an α approximate solution for the TSP then the latency of the obtained approximate solution has value at most $(1+\alpha\varepsilon+\alpha)L^*$.

Let us now consider the case of tree metric. In such case a TSP tour that visits a given set of nodes can be computed in polynomial time. However the above approach requires to know the set of nodes belonging to each segment in the optimal solution. These sets can be computed in quasi polynomial time in the case of tree metric by dynamic programming. In fact the break up in k segments implies that an edge is visited in the optimal solution at most k times.

Let us consider the case of a binary tree. First identify an edge that is a 1/3 : 2/3 separator and the algorithm \guesses" the number of times this edge is visited, and for each such portion the length of the portion and the number of nodes visited. \Guessing" means that using dynamic programming the algorithm exhaustively searches for all possibilities; since there at most $k = O(\log n = ")$ portions and the length of each portion is bounded by $O(n^3 = ")$ it follows that there is a polynomial number of solutions. By recurring on each side of the separator edge it is possible to compute an " break up in segments in $n^{O(\log n = ")}$. The above idea can be applied also to nonbinary trees.

Theorem 5. *[5] For any ", " > 0, there exists a* 1 + " *approximation algorithm for the TRP in tree metric that runs in time* $n^{O(\log n)}$.

In the Euclidean case a similar result can be obtained by making use of Arora's approximation scheme [6] for the computation of the TSP paths which correspnd to the segments of the TRP.

Theorem 6. *[5] For any ", " > 0, there exists a* 1 + " *approximation algorithm for the TRP in Euclidean metric that runs in time* $n^{O(\log n = "^2)}$.

The proof of Theorem 6 will be provided the next subsection along with the proof of existence of a polynomial time approximation scheme for TSP.

Let us now see how we can apply the preceding results in order to design approximation schemes for GSP. Recall that, given an instance x of the GSP with n nodes, if the integer weights associated to the nodes are polynomially related, then it is easy to see that GSP is polynomially reducible to an instance y of TRP. On the other side it can be proved [15] that if the weights are not polynomially bounded still there exists a polynomial time reduction that preserves the approximation schemes [8].

Given an instance of GSP, with n nodes, let w_{max} be the maximum weight associated to a city and let " be any positive real number. The idea of the proof is to round the weights associated to each city by a factor k, $k = w_{max}\ = c$ where $= "^2 = n^4$ and c is a suitably chosen constant. Namely, given an instance x of GSP, we de ne a new instance x^0, with the same set of nodes and the same metric distance of x that is obtained by rounding the weight associated to each city; namely, w_i, the weight associated to city i, becomes $bw_i = kc$. Note that by the above rounding the weights associated to the nodes of x^0 are now polynomially related and, therefore, x^0 is polynomially reducible to an instance of TRP.

Assume now that we are given a tour T that is an optimal solution of x^0; we now show that T is a (1 + ") approximate solution of x. In fact, following [5] we can assume that the maximum distance between nodes is $cn^2 = "$, where c

is a constant; it follows that the rounding introduces an absolute error for the contribution of city i to the objective function that is bounded by

$$kcn^3=" = w_{max} \;\; n^3=" = w_{max}"=n:$$

By summing over all nodes we obtain that the total absolute error is bounded by $w_{max}"$; since w_{max} is a lower bound on the optimum value of instance x it follows that T is a (1 + ") approximate solution of x.

Assume now that we are given a approximate solution of x^0; analogously we can show that this solution is a (1 + ") approximate solution of x. The above reduction together with the approximation results of Theorems 5 and 6 imply the following theorem.

Theorem 7. *There exists a quasi polynomial time* (1 + ") *approximation algorithm for the GSP in the case of tree metric and Euclidean metric.*

5.1 Polynomial Time Approximation Schemes for the Euclidean TSP and TRP

Let us now insert the last missing stone which is needed to prove the existence of an approximate scheme for the GSP in the Euclidean case: the polynomial time approximation schemes for the Euclidean TSP [6] and TRP [5]. Let us rst see the result for the TSP.

The basic idea on which Arora's result is organized is the following. In order to overcome the computational complexity of the TSP in the Euclidean case we may reduce the combinatorial explosion of the solution space by imposing that the required approximate solutions should satisfy particular structural properties. Under suitable conditions the number of solutions which satisfy such properties may be reduced in such a way that we may search for the best approximate solution by means of a dynamic programming procedure which runs in polynomial time. Let us consider an Euclidean TSP instance x consisting of a set of n points in the plane and let L be the size of its bounding box B. Let us rst make the following simplifying assumptions: (i) all nodes have integral coordinates; (2) the minimum internode distance is 8; (3) the maximum internode distance is $O(n)$; (4) the size of the bounding box, L is $O(n)$ and it is a power of 2. It is not di cult to prove that if a PTAS exists for this particular type of instances, that we call well rounded instances, then it exists for general TSP instances. Now, suppose we are given a well rounded TSP instance. In order to characterize the approximate solutions that satisfy speci c structural properties we may proceed in the following way. We decompose the bounding box through a recursive binary partitioning until we have at most one point per square cell (in practice, and more conveniently, we can organize the instance into a quad-tree). Note that at stage i of the partitioning process we divide any square in the quad-tree of size $L=2^{i-1}$ which contains more than one point into 4 squares of size $L=2^i$. Then we identify $m = O(c \log L=")$ points evenly distributed on each side of any square created during the partition (plus four points in the square's corners). By slightly bending the edges of a TSP tour we will impose it to cross square boundaries only at those prespeci ed points (called \portals"). Finally we allow the partition to be shifted both horizontally and vertically by integer quantities a and b respectively. The structure theorem can then be stated in the following terms.

Theorem 8. *(Structure Theorem,[6]) Let a well rounded instance x be given, let L be the size of its bounding box B and let $'' > 0$ be a constant. Let us pick a and b, with $0 \quad a;b \quad L$, randomly and let us consider the recursive partitioning of B shifted by quantities a and b. Then with probability at least $1{=}2$ there is a salesman tour of cost at most $(1 + '')OPT$ (where OPT is the cost of an optimum solution) that crosses each edge of each square in the partition at most $r = O(1{=}'')$ times always going through one among $m = O(\log L{=}'')$ portals (such tour is called (m, r) light).*

Essentially the proof of the theorem is based on the fact that given a recursive partitioning of B shifted by quantities a and b, given an optimum TSP tour of length OPT it is possible to bend its edges slightly so that they cross square boundaries only $O(1{=}'')$ times and only at portals and the resulting increase in length of the path is at most $''\ OPT{=}2$. Over all possible shifts a, b, therefore, with probability $\quad 1{=}2$, such increase is bounded by $''\ OPT$.

On the basis of the structure theorem we can then de ne the following polynomial time approximation scheme for the solution of a well rounded TSP instance. In what follows we call a TSP path a tour, or a fragment of tour, which goes through the points in the TSP instance and whose edges are possibly bended through the portals. When we will have built a unique TSP path that goes exactly once through all points in the instance it will be immediately possible to transform it into a TSP tour by taking Euclidean shortcuts whenever possible.

Given the required approximation ratio $(1 + '')$ and given a TSP instance x the randomized algorithm performs the following steps:

1) Perturbation. Instance x is rst transformed into a well rounded instance x^0. We will then look for a $(1 + ''^0)$ approximate solution for instance x^0 where $''^0$ can be easily computed from $''$.

2) Construction of the shifted quad-tree. Given a random choice of $1 \quad a;b \quad L$ a quad-tree with such shifts is computed. The depth of the quad-tree will be $O(\log n)$ and the number of squares it will contain is $T = O(n \log n)$

3) Construction of the path by dynamic programming. The path that satis es the structure theorem can be constructed bottom-up by dynamic programming as follows. In any square there may be p paths, each connecting a pair of portals such that for any i a path goes from the rst portal to the second portal in pair p_i. Since $2p \quad 4r$ and there are $4m + 4$ portals on the borders of one square, there are at most $(4m + 4)^{4r}$ ways to chose the crossing points of the p paths and there are at most $4r!$ pairings among such crossing points. For each of the choices we compute the optimum solution which corresponds to one entry in the lookup table that the dynamic programming procedure has to construct. Since we have T squares, the total number of entries is $O(T(4m + 4)^{(4r)}(4r)!)$.

In order to determine the running time of the procedure let us see how the entries are constructed; rst note that for the leaves of the quad-tree we only have the condition that one path should go through the node (if any) in the leaf. Inductively, when we want to construct the entries for a square S at level $i - 1$ of the quad-tree, given the entries of squares S_1, S_2, S_3, S_4 at level i, we have to determine the optimum way to connect the paths in the subsquares by choosing among all possible choices on how to cross the inner borders among the four squares. Such choices are $(4m + 4)^{4r}(4r)^{4r}(4r)!$. All taken into account we have a running time $O(T(4m + 4)^{8r}(4r)^{4r}(4r!)^2)$, that is $O(n(\log n)^{O(1=")})$.

Easily enough the algorithm can be derandomized by exhaustively trying all possible shifts a, b, and picking the best path. This simply implies repeating steps 2) and 3) of the algorithm $O(n^2)$ times. In conclusion we can state the nal result.

Theorem 9. *[5] For any ", " > 0, there exists a 1 + " approximation algorithm for the TSP in Euclidean metric that runs in time* $O(n^3(\log n)^{1="})$.

The result for the TRP goes along the same line. As we have seen, in order to solve the TRP we compute $O(\log n=")$ segments consisting in as many salesman paths. Now the same algorithm as before can be constructed but this time we want to compute simultaneously the $O(\log n=")$ salesman paths. As a consequence, while in the case of the TSP we were looking for paths going at most $r = O(1=")$ times through one among $m = O(\log n=")$ portals, in the case of the TRP we construct paths that go $O(\log n=")$ times through the m portals. The same dynamic programming technique as in the case of TSP can then be applied, but now, since we may have $O(\log n=")$ crossings, the algorithm will require quasi-polynomial time $O(n^{O(\log n="^2)})$. Theorem 6 hence follows.

References

[1] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *Informatique Theoretique et Applications*, 20(1):79{87, 1986.

[2] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440{456, June 1995.

[3] Paola Alimonti and F. Lucidi. On mobile agent planning, 1999. manuscript.

[4] Paola Alimonti, F. Lucidi, and S. Triglia. How to move mobile agents, 1999. manuscript.

[5] Arora and Karakostas. Approximation schemes for minimum latency problems. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1999.

[6] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753{782, 1998.

[7] S. Arya and H. Kumar. A 2.5 approximation algorithm for the k-mst problem. *Information Processing Letter*, 65:117{118, 1998.

[8] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti=Spaccamela, and Marco Protasi. *Complexity and Approximation, Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.

[9] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 163{171, Montreal, Quebec, Canada, 23{25 May 1994.

[10] Avrim Blum, R. Ravi, and Santosh Vempala. A constant-factor approximation algorithm for the k-MST problem (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 442{448, Philadelphia, Pennsylvania, 22{24 May 1996.

[11] Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *37th Annual Symposium on Foundations of Computer Science*, pages 302{309, Burlington, Vermont, 14{16 October 1996. IEEE.

[12] M. X. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296{317, 1995.

[13] Michel Goemans and Jon Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 152{158, New York/Philadelphia, January 28{30 1996. ACM/SIAM.

[14] Elias Koutsoupias, Christos H. Papadimitriou, and Mihalis Yannakakis. Searching a xed graph. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium*, volume 1099 of *Lecture Notes in Computer Science*, pages 280{289, Paderborn, Germany, 8{12 July 1996. Springer-Verlag.

[15] Alberto Marchetti-Spaccamela and Leen Stougie, 1999. private communication.

Computing a Diameter-Constrained Minimum Spanning Tree in Parallel

Narsingh Deo and Ayman Abdalla

School of Computer Science
University of Central Florida, Orlando, FL 32816-2362, USA
{deo, abdalla}@cs.ucf.edu

Abstract. A minimum spanning tree (MST) with a small diameter is required in numerous practical situations. It is needed, for example, in distributed mutual exclusion algorithms in order to minimize the number of messages communicated among processors per critical section. The Diameter-Constrained MST (DCMST) problem can be stated as follows: given an undirected, edge-weighted graph *G* with *n* nodes and a positive integer *k*, find a spanning tree with the smallest weight among all spanning trees of *G* which contain no path with more than *k* edges. This problem is known to be NP-complete, for all values of *k*; $4 \le k \le (n-2)$. Therefore, one has to depend on heuristics and live with approximate solutions. In this paper, we explore two heuristics for the DCMST problem: First, we present a one-time-tree-construction algorithm that constructs a DCMST in a modified greedy fashion, employing a heuristic for selecting edges to be added to the tree at each stage of the tree construction. This algorithm is fast and easily parallelizable. It is particularly suited when the specified values for *k* are small—independent of *n*. The second algorithm starts with an unconstrained MST and iteratively refines it by replacing edges, one by one, in long paths until there is no path left with more than *k* edges. This heuristic was found to be better suited for larger values of *k*. We discuss convergence, relative merits, and parallel implementation of these heuristics on the MasPar MP-1 — a massively parallel SIMD machine with 8192 processors. Our extensive empirical study shows that the two heuristics produce good solutions for a wide variety of inputs.

1 Introduction

The Diameter-Constrained Minimum Spanning Tree (DCMST) problem can be stated as follows: given an undirected, edge-weighted graph *G* and a positive integer *k*, find a spanning tree with the smallest weight among all spanning trees of *G* which contain no path with more than *k* edges. The length of the longest path in the tree is called the diameter of the tree. Garey and Johnson [7] show that this problem is NP-complete by transformation from Exact Cover by 3-Sets. Let *n* denote the number of nodes in *G*. The problem can be solved in polynomial time for the following four special cases: $k = 2$, $k = 3$, $k = (n-1)$, or when all edge weights are identical. All other cases are NP-complete. In this paper, we consider graph *G* to be complete and with *n* nodes. An incomplete graph can be viewed as a complete graph in which the missing edges have infinite weights.

The DCMST problem arises in several applications in distributed mutual exclusion where message passing is used. For example, Raymond's algorithm [5,11] imposes a logical spanning tree structure on a network of processors. Messages are passed among processors requesting entrance to a critical section and processors granting the privilege to enter. The maximum number of messages generated per critical-section execution is $2d$, where d is the diameter of the spanning tree. Therefore, a small diameter is essential for the efficiency of the algorithm. Minimizing edge weights reduces the cost of the network.

Satyanarayanan and Muthukrishnan [12] modified Raymond's original algorithm to incorporate the "least executed" fairness criterion and to prevent starvation, also using no more than $2d$ messages per process. In a subsequent paper [13], they presented a distributed algorithm for the *readers and writers* problem, where multiple nodes need to access a shared, serially reusable resource. In this algorithm, the number of messages generated by a read operation and a write operation has an upper bound of $2d$ and $3d$, respectively.

In another paper on distributed mutual exclusion, Wang and Lang [14] presented a token-based algorithm for solving the *p-entry critical-section* problem, where a maximum of p processes are allowed to be in their critical section at the same time. If a node owns one of the p tokens of the system, it may enter its critical section; otherwise, it must broadcast a request to all the nodes that own tokens. Each request passes at most $2pd$ messages.

The DCMST problem also arises in Linear Lightwave Networks (LLNs), where multi-cast calls are sent from each source to multiple destinations. It is desirable to use a short spanning tree for each transmission to minimize interference in the network. An algorithm by Bala *et al* [4] decomposed an LLN into edge disjoint trees with at least one spanning tree. The algorithm builds trees of small diameter by computing trees whose maximum node-degree was less than a given parameter, rather than optimizing the diameter directly. Furthermore, the lines of the network were assumed to be identical. If the LLN has lines of different bandwidths, lines of higher bandwidth should be included in the spanning trees to be used more often and with more traffic. Employing an algorithm that solves the DCMST problem can help find a better tree decomposition for this type of network. The network would be modeled by an edge-weighted graph, where an edge of weight $1/x$ is used to represent a line of bandwidth x.

Three exact-solution algorithms the DCMST problem developed by Achuthan *et al* [3] used Branch-and-Bound methods to reduce the number of subproblems. The algorithms were implemented on a SUN SPARC II workstation operating at 28.5 MIPS. The algorithms were tested on complete graphs of different orders ($n \leq 40$), using 50 cases for each order, where edge-weights were randomly generated numbers between 1 and 1000. The best algorithm for $k = 4$ produced an exact solution for $n = 20$ in less than one second on average, but it took an average of 550 seconds for $n = 40$. Clearly, such exact-algorithms, with exponential time complexity, are not suitable for graphs with thousands of nodes.

For large graphs, Abdalla *et al* [2] presented a fast approximate algorithm. The algorithm first computed an unconstrained MST, then iteratively refined it by increasing the weights of ($\log n$) edges near the center of the tree and recomputing the MST until the diameter constraint was achieved. The algorithm was not always able to produce DCMST(k) for $k \leq 0.05n$ because sometimes it reproduced spanning trees already considered in earlier iterations, thus entering an infinite cycle.

In this paper, we first present a general method for evaluating the solutions to the DCMST problem in Section 2. Then, we present approximate algorithms for solving this problem employing two distinct strategies: One-Time Tree Construction (OTTC) and Iterative Refinement (IR). The OTTC algorithm, based on Prim's algorithm, is presented in Section 4. A special IR algorithm and a general one are presented in Sections 3 and 5, respectively.

2 Evaluating the Quality of a DCMST

Since the exact DCMST weights cannot be determined in a reasonable amount of time for large graphs, we use the ratio of the computed weight of the DCMST to that of the unconstrained MST as a rough measure of the quality of the solution.

To obtain a crude upper bound on the DCMST(k) weight (where k is the diameter constraint), observe that DCMST(2) and DCMST(3) are feasible (but often grossly suboptimal) solutions of DCMST(k) for all $k > 3$. Since there are polynomial-time exact algorithms for DCMST(2) and DCMST(3), these solutions can be used as upper bounds for the weight of an approximate DCMST(k). In addition, we develop a special approximate-heuristic for DCMST(4) and compare its weight to that of DCMST(3) to verify that it provides a tighter bound and produces a better solution for $k = 4$. We use these upper bounds, along with the ratio to the unconstrained MST weight, to evaluate the quality of DCMST(k) obtained.

3 Special IR Heuristic for DCMST(4)

The special algorithm to compute DCMST(k) starts with an optimal DCMST(3), then replaces higher-weight edges with smaller-weight edges, allowing the diameter to increase to 4.

3.1 An Exact DCMST(3) Computation

Clearly, in a DCMST(3) of graph G, every node must be of degree 1 except two nodes, call them u and v. Edge (u, v) is the *central edge* of such a spanning tree. To construct DCMST(3), we select an edge to be the central edge (u, v), then, for every node x in G, $x \notin \{u, v\}$, we include in the spanning tree the smaller of the two edges (x, u) and (x, v). To get an optimal DCMST(3), we compute all such spanning trees — with every edge in G as its central edge — and take the one with the smallest weight. Since we have m edges to choose from, we have to compute m different spanning trees. Each of these trees requires $(n - 2)$ comparisons to select (x, u) or (x, v). Therefore, the total number of comparisons required to obtain the optimal DCMST(3) is $(n - 2)m$.

3.2 An Approximate DCMST(4) Computation

To compute DCMST(4), we start with an optimal DCMST(3). Then, we relax the diameter constraint while reducing the spanning tree weight using edge replacement to get a smaller-weight DCMST(4). The refinement process starts by arbitrarily selecting one end node of the central edge (u, v), say u, to be the center of DCMST(4). Let $W(a, b)$ denote the weight of an edge (a, b). For every node x adjacent to v, we attempt to obtain another tree of smaller-weight by replacing edge (v, x) with edge (x, y), where $W(x, y) < W(x, v)$. Furthermore, the replacement (x, y) is an edge such that y is adjacent to u and for all nodes z adjacent to u and $z \neq v$, $W(x, y) \leq W(x, z)$. If no such edge exists, we keep edge (v, x) in the tree. We use the same method to compute a second DCMST(4), with v as its center. Finally, we accept the DCMST(4) with the smaller weight as the solution.

Suppose there are p nodes adjacent to u in DCMST(3). Then, there are $(n - p - 2)$ nodes adjacent to v. Therefore, we make $2p(n - p - 2)$ comparisons to get DCMST(4). It can be shown that employing this procedure for a complete graph, the expected number of comparisons required to obtain an approximate DCMST(4) from an exact DCMST(3) is $(n^2 - 8n - 12)/2$.

4 One-Time Tree Construction

In the One-Time Tree Construction (OTTC) strategy, a modification of Prim's algorithm is used to compute an approximate DCMST in one pass. Prim’s algorithm has been experimentally shown to be the fastest for computing an MST for large dense graphs[8].

The OTTC algorithm grows a spanning tree by connecting the nearest neighbor that does not violate the diameter constraint. Since such an approach keeps the tree connected in every iteration, it is easy to keep track of the increase in tree-diameter. This Modified Prim algorithm is formally described in Figure 1, where we maintain the following information for each node u:

- *near*(u) is the node in the tree nearest to the non-tree node u.
- *wnear*(u) is the weight of edge (u, *near*(u)).
- *dist*(u, 1..n) is the distance (unweighted path length) from u to every other node in the tree if u is in the tree, and is set to -1 if u is not yet in the tree.
- *ecc*(u) is the eccentricity of node u, (the distance in the tree from u to the farthest node) if u is in the tree, and is set to -1 if u is not yet in the tree.

To update *near*(u) and *wnear*(u), we determine the edges that connect u to partially-formed tree T without increasing the diameter (as the first criterion) and among all such edges we want the one with minimum weight. We do this efficiently, without having to recompute the tree diameter for each edge addition.

In Code Segment 1 of the OTTC algorithm, we set the *dist*(v) and *ecc*(v) values for node v by copying from its parent node *near*(v). In Code Segment 2, we update the values of *dist* and *ecc* for the parent node in n steps. In Code Segment 3, we update the values of *dist* and *ecc* for other nodes. We make use of the *dist* and *ecc* arrays, as described above, to simplify the OTTC computation.

```
procedure ModifiedPrim
INPUT:Graph $G$, Diameter bound $k$
OUTPUT: Spanning Tree $T = (V_T, E_T)$
initialize $V_T := \Phi$ and $E_T, := \Phi$
select a root node $v_0$ to be included in $V_T$
initialize $near(u) := v_0$ and $wnear(u) := w_{uv0}$, for every $u \notin V_T$
compute a next-nearest-node $v$ such that:
     $wnear(v) = MIN_{u \notin VT}\{wnear(u)\}$
while ($|E_T| < (n-1)$)
   select the node $v$ with the smallest value of $wnear(v)$
   set $V_T := V_T \cup \{v\}$ and $E_T := E_T \cup \{(v, near(v))\}$
   {1. set $dist(v,u)$ and $ecc(v)$}
   for $u$ = 1 to $n$
     if $dist(near(v),u) > -1$ then
       $dist(v,u) := 1 + dist(near(v),u)$
   $dist(v, v) := 0$
   $ecc(v) := 1 + ecc(near(v))$
   {2. update $dist(near(v)$,u) and $ecc(near(v))$}
   $dist(near(v),v) = 1$
   if $ecc(near(v)) < 1$ then
     $ecc(near(v)) = 1$
   {3. update other nodes' values of $dist$ and $ecc$}
   for each tree node $u$ other than $v$ or $near(v)$
     $dist(u,v) = 1 + dist(u,near(v))$
     $ecc(u) = MAX\{ecc(u),dist(u,v)\}$
   {4. update the $near$ and $wnear$ values for other nodes in $G$}
   for each node $u$ not in the tree
   if $1 + ecc(near(u)) > u$ then
     examine all nodes in $T$ to determine $near(u)$ and $wnear(u)$
   else
     compare $wnear(u)$ to the weight of $(u,v)$.
```

Fig. 1. OTTC Modified Prim algorithm

Code Segment 4 is the least intuitive. Here, we update the *near* and *wnear* values for every node not yet in the tree by selecting an edge which does not increase the tree diameter beyond the specified constraint and has the minimum weight among all such edges. Now, adding *v* to the tree may or may not increase the diameter. If the tree diameter increases, and *near*(*u*) lies along a longest path in the tree, then adding *u* to the tree by connecting it to *near*(*u*) may violate the constraint. In this case, we must reexamine all nodes of the tree to find a new value for *near*(*u*) that does not violate the diameter constraint. This can be achieved by examining *ecc*(*t*) for nodes *t* in the tree; i.e., we need not recompute the tree diameter. This computation includes adding a new node to the tree.

On the other hand, if (u, *near*(u)) is still a feasible edge, then *near*(u) is the best choice for u among all nodes in the tree except possibly v, the newly added node. In this case, we need only determine whether edge (u, v) would increase the tree diameter beyond the constraint, and if not, whether the weight of (u, v) is less than *wnear*(u).

The complexity of Code Segment 4 is $O(n^2)$ when the diameter constraint k is small, since it requires looking at each node in the tree once for every node not in the tree. This makes the time complexity of this algorithm higher than that of Prim's algorithm. The *while* loop requires $(n - 1)$ iterations. Each iteration requires at most $O(n^2)$ steps, which makes the worst case time complexity of the algorithm $O(n^3)$.

This algorithm does not *always* find a DCMST. Furthermore, the algorithm is sensitive to the node chosen for starting the spanning tree. In both the sequential and parallel implementations, we compute n such trees, one for each starting node. Then, we output the spanning tree with the largest weight.

To reduce the time needed to compute the DCMST further, we develop a heuristic that selects a small set of starting nodes as follows. Select the q nodes (q is independent of n) with the smallest sum of weights of the edges emanating from each node. Since this is the defining criterion for spanning trees with diameter $k = 2$ in complete graphs, it is polynomially computable. The algorithm now produces q spanning trees instead of n, reducing the overall time complexity by a factor $O(n)$ when we choose a constant value for q.

5 The General Iterative Refinement Algorithm

This IR algorithm does not recompute the spanning tree in every iteration; rather, a new spanning tree is computed by modifying the previously computed one. The modification performed never produces a previously generated spanning tree and, thus it guarantees the algorithm will terminate. Unlike the algorithm in [2], this algorithm removes one edge at a time and prevents cycling by moving away from the center of the tree whenever cycling becomes imminent.

This new algorithm starts by computing the unconstrained MST for the input graph G = (V, E). Then, in each iteration, it removes one edge that breaks a longest path in the spanning tree and replaces it by a non-tree edge without increasing the diameter. The algorithm requires computing eccentricity values for all nodes in the spanning tree in every iteration.

The initial MST can be computed using Prim's algorithm. The initial eccentricity values for all nodes in the MST can be computed using a preorder tree traversal where each node visit consists of computing the distances from that node to all other nodes in the spanning tree. This requires a total of $O(n^2)$ computations. As the spanning tree changes, we only recompute the eccentricity values that change. After computing the MST and the initial eccentricity values, the algorithm identifies one edge to remove from the tree and replaces it by another edge from G until the diameter constraint is met or the algorithm fails. When implemented and executed on a variety of inputs, we found that this process required no more than $(n + 20)$ iterations. Each iteration consists of two parts. In the first part, described in Subsection 5.1, we find an edge whose removal can contribute to reducing the diameter, and in the second part, described in Subsection 5.2, we find a good replacement edge. The IR algorithm is shown in Figure 2, and its two different edge-replacement subprocedures are shown in Figures 3 and 4. We use $ecc_T(u)$ to denote the eccentricity of node *u* with respect to spanning tree *T*, the maximum distance from *u* to any other node in *T*. The diameter of a spanning tree *T* is given by $MAX\{ecc_T(u)\}$ over all nodes *u* in *T*.

5.1 Selecting Edges for Removal

To reduce the diameter, the edge removed must break a longest path in the tree and should be near the center of the tree. The center of spanning tree *T* can be found by identifying the nodes *u* in *T* with $ecc_T(u) = \lceil \text{diameter}/2 \rceil$, the node (or two nodes) with minimum eccentricity.

Since we may have more than one edge candidate for removal, we keep a sorted list of candidate edges. This list, which we call *MID*, is implemented as a max-heap sorted according to edge weights, so that the highest-weight candidate edge is at the root.

Removing an edge from a tree does not guarantee breaking all longest paths in the tree. The end nodes of a longest path in *T* have maximum eccentricity, which is equal to the diameter of *T*. Therefore, we must verify that removing an edge splits the tree *T* into two subtrees, *subtree*1 and *subtree*2, such that each of the two subtrees contains a node *v* with $ecc_T(v)$ equal to the diameter of the tree *T*. If the highest-weight edge from list *MID* does not satisfy this condition, we remove it from *MID* and consider the next highest. This process continues until we either find an edge that breaks a longest path in *T* or the list *MID* becomes empty.

If we go through the entire list, *MID*, without finding an edge to remove, we must consider edges farther from the center. This is done by identifying the nodes *u* with $ecc_T(u) = \lceil \text{diameter}/2 \rceil$ + bias, where bias is initialized to zero, and incremented by 1 every time we go through *MID* without finding an edge to remove. Then, we recompute *MID* as all the edges incident to set of nodes *u*. Every time we succeed in finding an edge to remove, we reset the bias to zero.

This method of examining edges helps prevent cycling since we consider a different edge every time until an edge that can be removed is found. But to guarantee the prevention of cycling, we always select a replacement edge that reduces the length of a path in *T*. This will guarantee that the refinement process will terminate, since we will either reduce the diameter below the bound *k*, or bias will become so large that we try to remove the edges incident to the end-points of the longest paths in the tree.

```
procedure IterativeRefinement
INPUT: Graph $G = (V,E)$, diameter bound $k$
OUTPUT: Spanning tree $T$ with diameter $\leq k$
compute MST and $ecc_T(v)$ for all $v$ in $V$
$MID$ := $\Phi$
$move$ := false
```

```
repeat
    diameter := $MAX_{v \in V}\{ecc_T(v)\}$
    if MID = Φ then
       if move = true then
          move := false
          MID := edges (u,v) that are one edge farther from
               the center of T than in the previous iteration
       else
          MID := edges (u,v) at the center of T
    repeat
       (x,y) := highest weight edge in MID
       {This splits T into two trees: subtree1 and subtree2}
    until MID = Φ
          OR $MAX_{u \in subtree1}\{ecc_T(u)\} = MAX_{v \in subtree2}\{ecc_T(v)\}$
    if MID = Φ then        {no good edge to remove was found}
       move := true
    else
       remove (x,y) from T
       get a replacement edge and add it to T
    recompute $ecc_T$ values
until diameter ≤ k OR we are removing the edges farthest from
      the center of T
```

Fig. 2. The general IR algorithm

In the worst case, computing list *MID* requires examining many edges in *T*, requiring $O(n)$ comparisons. In addition, sorting *MID* will take $O(n \log n)$ time. A replacement edge is found in $O(n^2)$ time since we must recompute eccentricity values for all nodes to find the replacement that helps reduce the diameter. Therefore, the iterative process, which removes and replaces edges for *n* iterations, will take $O(n^3)$ time in the worst case. Since list *MID* has to be sorted every time it is computed, the execution time can be reduced by a constant factor if we prevent *MID* from becoming too large. This is achieved by an edge-replacement method that keeps the tree *T* fairly uniform so that it has a small number of edges near the center, as we will show in the next subsection. Since *MID* is constructed from edges near the center of *T*, this will keep *MID* small.

5.2 Selecting a Replacement Edge

When we remove an edge from a tree *T*, we split *T* into two subtrees: *subtree1* and *subtree2*. Then, we select a non-tree edge to connect the two subtrees in a way that reduces the length of at least one longest path in *T* without increasing the diameter. The diameter of *T* will be reduced when all longest paths have been so broken. We develop two methods, ERM1 and ERM2, to find such replacement edges.

5.2.1 Edge-Replacement Method ERM1

This method, shown in Figure 3, selects a minimum-weight edge (*a*, *b*) in *G* connecting a central node *a* in *subtree*1 to a central node *b* in *subtree*2. Among all edges that can connect *subtree*1 to *subtree*2, no other edge (*c*, *d*) will produce a tree such that the diameter of *subtree*1 ∪ *subtree*2 ∪ {(*c*, *d*)} is smaller than the diameter of *subtree*1 ∪ *subtree*2 ∪ {(*a*, *b*)}. However, such an edge (*a*, *b*) is not guaranteed to exist in incomplete graphs.

```
procedure ERM1
Recompute $ecc_{subtree1}$ and $ecc_{subtree2}$ for each subtree by itself
$m_1$ := $MIN_{u \in subtree1}\{ecc_{subtree1}(u)\}$
$m_2$ := $MIN_{u \in subtree2}\{ecc_{subtree2}(u)\}$
($a$,$b$) := minimum-weight edge in $G$ that has:
     $a \in subtree1$ AND $b \in subtree2$ AND
     $ecc_{subtree1}(a) = m_1$ AND $ecc_{subtree2}(b) = m_2$
Add edge ($a$,$b$) to $T$
if $MID = \Phi$ OR $bias = 0$ then
   $move$ := true
   $MID$ := $\Phi$
```

Fig. 3. Edge-replacement method ERM1

Since there can be at most two central nodes in each subtree, there are at most four edges to select from. The central nodes in the subtrees can be found by computing $ecc_{subtree1}$ and $ecc_{subtree2}$ in each subtree, then taking the nodes *v* with $ecc_{subtree}(v) = MIN\{ecc_{subtree}(u)\}$ over all nodes *u* in the subtree that contains *v*. This selection can be done in $O(n^2)$ time.

Finally, we set the boolean variable *move* to *true* every time we remove an edge incident to the center of the tree. This causes the removal of edges farther from the center of the tree in the next iteration of the algorithm, which prevents removing the edge (*a*, *b*) which has just been added.

This edge-replacement method seems fast at the first look, because it selects one out of four edges. However, in the early iterations of the algorithm, this method creates nodes of high degree near the center of the tree, which causes *MID* to be very large. This, as we have shown in the previous section, causes the time complexity of the algorithm to increase by a constant factor. Furthermore, having at most four edges from which to select a replacement often causes the tree weight to increase significantly.

5.2.2 Edge-Replacement Method ERM2

This method, shown in Figure 5, computes $ecc_{subtree1}$ and $ecc_{subtree2}$ values for each subtree individually, as in ERM1. Then, the two subtrees are joined as follows. Let the removed edge (*x*, *y*) have $x \in$ *subtree1* and $y \in$ *subtree2*. The replacement edge will be the smallest-weight edge (*a*, *b*) which (1) guarantees that the new edge does not increase the diameter, and (2) guarantees reducing the length of a longest path in the tree at least by one. We enforce condition (1) by:

$$ecc_{\text{subtree1}}(a) \leq ecc_{\text{subtree1}}(x) \text{ AND } ecc_{\text{subtree2}}(b) \leq ecc_{\text{subtree2}}(y) \,, \quad \textbf{(1)}$$

and condition (2) by:

$$ecc_{\text{subtree1}}(a) < ecc_{\text{subtree1}}(x) \text{ OR } ecc_{\text{subtree2}}(b) < ecc_{\text{subtree2}}(y) \,. \quad \textbf{(2)}$$

If no such edge (*a, b*) is found, we must remove an edge farther from the center of the tree, instead.

```
procedure ERM2
recompute $ecc_{\text{subtree1}}$ and $ecc_{\text{subtree2}}$ for each subtree by itself
$m_1$ := $ecc_{\text{subtree1}}(x)$
$m_2$ := $ecc_{\text{subtree2}}(y)$
($a$,$b$) := minimum-weight edge in $G$ that has:
      $a \in subtree1$ AND $b \in subtree2$ AND $ecc_{\text{subtree1}}(a) \leq m_1$
      AND $ecc_{\text{subtree2}}(b) \leq m_2$ AND
      ($ecc_{\text{subtree1}}(a) < m_1$ OR $ecc_{\text{subtree2}}(b) < m_2$)
if such an edge ($a$,$b$) is found then
   add edge ($a$,$b$) to $T$
else
   add the removed edge ($x$,$y$) back to $T$
   $move$ := true
```

Fig. 4. Edge-replacement method ERM2

Since ERM2 is not restricted to the centers of the two subtrees, it works better than ERM1 on incomplete graphs. In addition, it can produce DCMSTs with smaller weights because it selects a replacement from a large set of edges, instead of 4 or fewer edges as in ERM1. The larger number of edges increases the total time complexity of the IR algorithm by a constant factor over ERM1. Furthermore, this method does not create nodes of high degree near the center of the tree as in ERM1. This helps keep the size of list *MID* small in the early iterations, reducing the time complexity of the IR algorithm by a constant factor.

6 Implementation

In this section, we present empirical results obtained by implementing the OTTC and IR algorithms on the MasPar MP-1, a massively-parallel SIMD machine of 8192 processors. The processors are arranged in a mesh where each processor is connected to its eight neighbors.

Complete graphs K_n, represented by their ($n \times n$) weight matrices, were used as input. Since the MST of a randomly generated graph has a small diameter, $O(\log n)$ [2], they are not suited for studying the performance of DCMST(k) algorithms. Therefore, we generated graphs in which the minimum spanning trees are forced to have diameter of ($n - 1$).

6.1 One-Time Tree Construction

We parallelized the OTTC algorithm and implemented it on the MasPar MP-1 for graphs of up to 1000 nodes. The DCMST generated from one start node for a graph of 1000 nodes took roughly 71 seconds, which means it would take it about 20 hours to run with *n* start nodes. We address this issue by running the algorithm for a carefully selected small set of start nodes.

We used two different methods to choose the set of start nodes. SNM1 selects the center nodes of the *q* smallest stars in *G* as start nodes. SNM2 selects *q* nodes from *G* at random. As seen in Figure 5, the quality of DCMST obtained from these two heuristics, where we chose $q = 5$, is similar. The execution times of these two heuristics were also almost identical.

The results from running the OTTC algorithm with *n* start nodes were obtained for graphs of 50 nodes and compared with the results obtained with 5 start nodes for the same graphs; for $k = 4$, 5, and 10. The results compare the average value of the smallest weight found from SNM1 and SNM2 to the average weight found from the OTTC algorithm that runs for *n* iterations. The quotient of these values is reported. For $k = 4$, the DCMST obtained using SNM1 had weight of 1.077 times the weight from the *n*-iteration OTTC algorithm. The cost of SNM2-tree was 1.2 times that of the *n*-iteration tree. For $k = 5$, SNM1 weight-ratio was 1.081 while SNM2 weight-ratio was 1.15. For $k = 10$, SNM1 weight-ratio was 1.053 while SNM2 weight-ratio was 1.085. In these cases, SNM1 outperforms SNM2 in terms of the quality of solutions, in some cases by as much as 12%. The results obtained confirm the theoretical analysis that predicted an improvement of $O(n)$ in execution time, as described in Section 4. The execution time for both SNM1 and SNM2 is approximately the same. This time is significantly less than the time taken by the *n*-iteration algorithm as expected. Therefore, SNM1 is a viable alternative to the *n*-iteration algorithm.

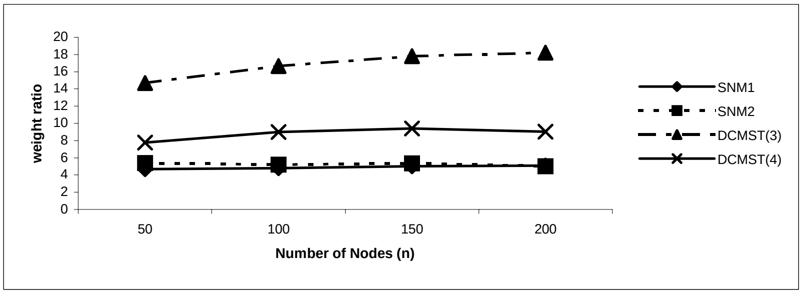

Fig. 5. Weight of DCMST(5), obtained using two different node-search heuristics, as a multiple of MST weight. Initial diameter = $n - 1$

6.2 The Iterative Refinement Algorithms

The heuristic for DCMST(4) was also parallelized and implemented on the MasPar MP-1. It produced DCMST(4) with weight approximately half that of DCMST(3), as we see in Figures 5, 6, and 8. The time to refine DCMST(4) took about 1% of the time to calculate DCMST(3).

We also parallelized and implemented the general IR algorithm on the MasPar MP-1. As expected, the algorithm did not enter an infinite loop, and it always terminated within (n + 20) iterations. The algorithm was unable to find a DCMST with diameter less than 12 in some cases for graphs with more than 300 nodes. In graphs of 400, 500, 1000, and 1500 nodes, our empirical results show a failure rate of less than 20%. The algorithm was 100% successful in finding a DCMST with k = 15 for graphs of up to 1500 nodes. This shows that the failure rate of the algorithm does not depend on what fraction of n the value of k is. Rather, it depends on how small the constant k is.

To see this, we must take a close look at the way we move away from the center of the tree when we select edges for removal. Note that the algorithm will fail only when we try to remove edges incident to the end-points of the longest paths in the spanning tree. Also note that we move away from the center of the tree every time we go through the entire set *MID* without finding a good replacement edge, and we return to the center of the spanning tree every time we succeed.

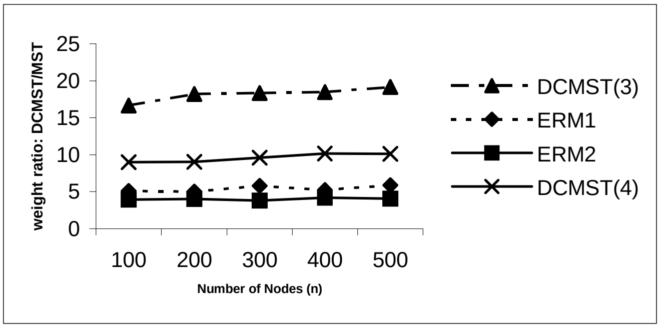

Fig. 6. Quality of DCMST(10) obtained using two different edge-replacement methods. Initial diameter = $n - 1$

Thus, the only way the algorithm fails is when it is unable to find a good replacement edge in $\lceil$diameter/2$\rceil$ *consecutive* attempts, each of which includes going through a different set of *MID*. Empirical results show that it is unlikely that the algorithm will fail for 8 consecutive times, which makes it suitable for finding DCMST where the value of k is a constant greater than or equal to 15. The algorithm still performs fairly well with k =10, and we did use that data in our analysis, where we excluded the few cases in which the algorithm did not achieve diameter 10. This exclusion should not affect the analysis, since the excluded cases all achieved diameter less than 15 with approximately the same speed as the successful attempts.

The quality of DCMST(10) obtained by the IR technique using the two different edge replacement methods, ERM1 and ERM2, is shown in Figure 6. The diagram shows the weight of the computed DCMST(10) as a multiple of the weight of the unconstrained MST. The time taken by the algorithm using ERM1 and ERM2 to obtain DCMST(10) is shown in Figure 7. As expected, ERM2 out-performs ERM1 in time and quality. In addition, ERM1 uses more memory than ERM2, because the size of *MID* when we use ERM1 is significantly larger than its size when ERM2 is used.

This is caused by the creation of high-degree nodes by ERM1, as explained in Subsection 4.2.

We tested the general IR algorithm, using ERM2, on random graphs. The quality of the DCMSTs obtained are charted in Figure 8. Comparing this figure with those obtained for the randomly-generated graphs forced to have unconstrained MST with diameter (n – 1), it can be seen that the quality of DCMST(10) in the graphs starting with MSTs of (n – 1) diameter is better than that in unrestricted random graphs. This is because the IR algorithm keeps removing edges close to the center of the constrained spanning tree, which contain more low-weight edges in unrestricted random graphs, coming from the unconstrained MST. But when the unconstrained MST has diameter (n –1), there are more heavy-weight edges near the center that were added in some earlier iterations of the algorithm. Therefore, the DCMST for this type of graphs will lose less low-weight edges than in unrestricted random graphs.

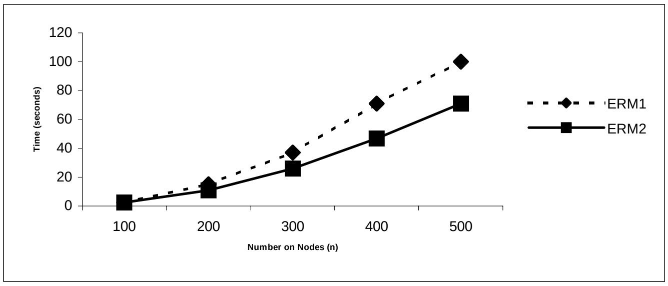

Fig. 7. Time to reduce diameter from *n*–1 to 10 using two different edge-replacement methods

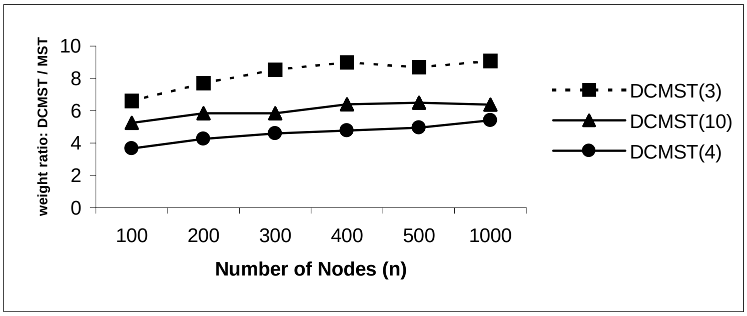

Fig. 8. Quality of DCMST(4) and DCMST(10) for unrestricted random graphs

Furthermore, the weight of DCMST(4) was lower than that of DCMST(10) in unrestricted random graphs. Note that the DCMST(4) heuristic approaches the diameter optimization from above, rather than from below. When the diameter constraint is small, it becomes more difficult for the general IR algorithm to find a solution and allows large increases in tree-weight in order to achieve the required diameter. The approach from the upper bound, however, guarantees the tree weight will not increase during the refinement process. The performance of the DCMST(4) algorithm did not change much in unrestricted random graphs. Rather, the quality of DCMST(10) deteriorated, exceeding the upper bound. Clearly, DCMST(4) algorithm provides a better solution for this type of graphs.

7 Conclusions

We have presented three algorithms that produce approximate solutions to the DCMST problem, even when the diameter constraint is a small constant. One is a modification of Prim’s algorithm, combined with a heuristic that reduces the execution time by a factor of *n* (by selecting a small constant number of nodes as the start nodes in the OTTC algorithm) at a cost of a small increase in the weight of the DCMST.

The second is an IR algorithm to find an approximate DCMST. This algorithm is guaranteed to terminate, and it succeeds in finding a reasonable solution when the diameter constraint is a constant, about 15. The third is a special IR algorithm to compute DCMST(4). This algorithm was found to be especially effective for random graphs with uniformly distributed edge weights, as it outperformed the other two in speed and quality of solution. This algorithm provides a tighter upper bound on DCMST quality than the one provided by the DCMST(3) solution. We implemented these algorithms on an 8192-processor, the MasPar MP-1, for various types of graphs. The empirical results from this implementation support the theoretical conclusions obtained.

References

1. Abdalla, A., Deo, N., Fraceschini, R.: Parallel heuristics for the diameter-constrained MST problem. Congressus Numerantium, (1999) (*to appear*)
2. Abdalla, A., Deo, N., Kumar, N., Terry, T.: Parallel computation of a diameter-constrained MST and related problems. Congressus Numerantium, Vol. 126. (1997) 131−155
3. Achuthan, N.R., Caccetta, L., Caccetta, P., Geelen, J.F: Algorithms for the minimum weight spanning tree with bounded diameter problem. Optimization: Techniques and Applications, (1992) 297–304
4. Bala, K., Petropoulos, K., Stern,T.E.: Multicasting in a Linear Lightwave Network. IEEE INFOCOM ’93, Vol. 3. (1993) 1350–1358
5. Chow, R., Johnson, T.: Distributed Operating Systems and Algorithms. Addison-Wesley, Reading, MA (1997)
6. Deo N., Kumar, K.: Constrained Spanning Tree Problems: Approximate Methods and Parallel Computation. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 40. (1998) 191–217
7. Garey, M.R., Johnson D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco (1979)
8. Moret, B.M.E., Shapiro, H.D.: An empirical analysis of algorithms for constructing a minimum spanning tree. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 15. (1994) 99–117
9. P.W. Paddock.: Bounded Diameter Minimum Spanning Tree Problem, M.S. Thesis, George Mason University, Fairfax, VA (1984)

10. Palmer., E.M.: Graph Evolution: An Introduction to the Theory of Random Graphs, John - Wiley & Sons, Inc., New York (1985)
11. Raymond, K.: A tree-based algorithm for distributed mutual exclusion. ACM Transactions on Computer Systems, Vol. 7. No. 1. (1989) 61–77
12. Satyanarayanan, R., Muthukrishnan, D.R.: A note on Raymond's tree-based algorithm for distributed mutual exclusion. Information Processing Letters, Vol. 43. (1992) 249–255
13. Satyanarayanan, R., Muthukrishnan, D.R.: A static-tree-based algorithm for the distributed readers and writers problem. Computer Science and Informatics, Vol. 24. No.2. (1994) 21–32
14. Wang, S, Lang, S.D.: A tree-based distributed algorithm for the k-entry critical section problem. In: Proceedings of the 1994 International Conference on Parallel and Distributed Systems, (1994) 592–597

Algorithms for a Simple Point Placement Problem

Joshua Redstone and Walter L. Ruzzo

Department of Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
fredstone,ruzzog@cs.washington.edu

Abstract. We consider algorithms for a simple one-dimensional point placement problem: given N points on a line, and noisy measurements of the distances between many pairs of them, estimate the relative positions of the points. Problems of this flavor arise in a variety of contexts. The particular motivating example that inspired this work comes from molecular biology; the points are markers on a chromosome and the goal is to map their positions. The problem is NP-hard under reasonable assumptions. We present two algorithms for computing least squares estimates of the ordering and positions of the markers: a branch and bound algorithm and a highly effective heuristic search algorithm. The branch and bound algorithm is able to solve to optimality problems of 18 markers in about an hour, visiting about 10^6 nodes out of a search space of 10^{16} nodes. The local search algorithm usually was able to find the global minimum of problems of similar size in about one second, and should comfortably handle much larger problem instances.

1 Introduction

The problem of mapping genetic information has been the subject of extensive research since experimenters started breeding fruit flies for physical characteristics. Due to the small scale of chromosomes, it has been difficult to obtain accurate information on their structure. Many techniques relying on statistical inference of indirect data have been applied to deduce this information; see [1] for some examples.

More recently, researchers have developed many techniques for estimating of relative positions various genetic features by more direct physical means. We are interested in one called fluorescent *in situ* hybridization (FISH). In this technique, pairs of fluorescently labeled probes are hybridized (attached) to specific sites on a chromosome. The 2-d projection of the distance between the probes is measured under a microscope. Despite the highly folded state of DNA *in vivo* and the resulting high variance of individual measurements, [10] shows that the genomic distance can be estimated if the experiment is repeated in many cells.

Not surprisingly, if more pairs of probes are measured, and the measurement between each pair is repeated many times, the accuracy of the answer increases. Unfortunately, so does the cost. Hence, the resulting computational problem is the following:

Problem: Given N probes on a line, and an incomplete set of noisy pairwise measurements between probes, determine the best estimate of the ordering and position of the probes.

If the measurements were complete and accurate, the problem would be easy—the farthest pair obviously are the extreme ends, and the intervening points can be placed by sorting their distances to the extremes. However, with partial, noisy data, the problem is known to be NP-hard. (See [6, 5] for a particularly simple proof.)

1.1 Previous Work

Brian Pinkerton previously investigated solving this problem using the seriation algorithm of [3], and a branch and bound algorithm (personal communication, 6/96). The seriation algorithm, which is a local search algorithm, was only moderately effective. The branch and bound algorithm, using a simple bounding function, was able to solve problems involving up to about 16 probes.

There has been extensive work on other algorithms to solve DNA mapping problems, but they are based on distance estimates from techniques other than FISH, and are tailored to the particular statistical properties of the distance measurements. Two among many examples are the distance geometry algorithm of [7], based on recombination frequency data, and [2], which investigated branch and bound, simulated annealing, and maximum likelihood algorithms based on data from radiation hybrid mapping.

1.2 Outline

We present two algorithms for finding least-squares solutions to the probe placement problem. One is a branch and bound algorithm that can find provably optimal solutions to problems of moderate, but practically useful, size. The second is a heuristic search algorithm, fundamentally a “hill-climbing” or greedy algorithm, that is orders of magnitude faster than the branch and bound algorithm, and although it is incapable of giving certifiably optimal solutions, it appears to be highly effective on this data.

In the next section we sketch some of the more difficult aspects of the problem. Section 3 develops a cost function to evaluate solutions. Section 4 describes the heuristic search algorithm. Section 5 outlines the branch and bound algorithm. We then present the results of simulations of the two algorithms in Section 6.

2 Introduction to the Solution Space

Before explaining the development of the algorithms, it is helpful to gain some intuition about the solution space. Given that the data is both noisy and incomplete, the problem can be under-constrained and/or over-constrained. In this domain, a “constraint” refers to a measurement between two probes (since it constrains the placement of the probes).

An under-constrained problem instance is one in which a probe might not have enough measurements to other probes to uniquely determine its position. In the example of four probes in Figure 1, probe B has only one measurement to probe A, and so a

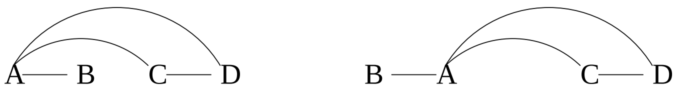

Fig. 1. An example of an under-constrained ordering (Probe B can be placed on either side of probe A). A line between two probes indicates a measurement between the probes.

location on either side of probe A is consistent with the data. It is also important to note that in all solutions, left/right orientation is arbitrary as is the absolute probe position.

In a more extreme example, a set of probes could have no measurements to another set. In Figure 2, probes A and B have no measurements to probes C and D, and placement anywhere relative to probes C and D is consistent with the data.

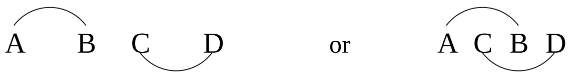

Fig. 2. Another example of an under-constrained ordering

In the examples of Figures 1 and 2, not only are the positions not uniquely determined, but different orderings are possible. When developing search algorithms, we have to be careful to recognize and treat such cases correctly. It appears that in real data such as from [Trask, personal communication, 1996], there are no degrees of freedom in the relative positioning of probes due to the careful choice of pairs of probes to measure. However, under-constrained instances do arise in the branch and bound algorithm described in Section 5 and in any algorithm that solves the problem by examining instances with a reduced set of constraints.

Due to the noise in the data, parts of a problem instance will be over-constrained. For example, as shown in Figure 3, if we examine three probes with pairwise measurements between them and there isn't an ordering such that the sum of two pairwise measurements equals the third pairwise measurement, there will be no way to place the three probes on a line. In this case, the distances between the probes in any linear placement will unavoidably be different from the measured distances.

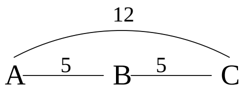

Fig. 3. There is no way to linearly place these probes on a line and respect all the measurements.

Given the existence of over and under-constrained problems, it is necessary to develop a method of evaluating how well a solution conforms to the data. This is covered in Section 3. Once we define how to evaluate a solution, we will develop algorithms to search for the best solution.

3 How to Evaluate a Probe Placement

We construct a cost function to evaluate the "goodness" of a solution, then solve the problem by finding the answer that has the least cost. Let N be the number of probes, x_i be the assigned position of probe i, d_{ij} be the measured distance between probe i and probe j ($d_{ij} = d_{ji}$), and let $w_{ij} = w_{ji}$ be a nonnegative weight associated with the measured distance between probes i and j. We define the *cost* of this placement to be the weighted sum of squares of the differences between the measured distance between two probes and the distance between the probes in the given linear placement of the probes:

$$Cost(x_1, \ldots, x_N) = \sum_{\substack{i<j \\ d_{ij} \text{ measured}}} w_{ij}(|x_i - x_j| - d_{ij})^2. \tag{1}$$

Many subsequent formulae will be simplified by assuming $w_{ij} = 0$ if $i = j$ or if the distance d_{ij} has not been measured. For example, we could have omitted the qualifier "d_{ij} measured" from Equation 1 under this assumption.

Intuitively, the weight w_{ij} reflects the relative confidence we have in measurement d_{ij}. For example, if the measurement errors were independent normal random variables, then we should choose the weight w_{ij} to be proportional to $1/\sigma_{ij}^2$, where σ_{ij}^2 is the variance of d_{ij}. Least squares solutions under these assumptions have several desirable properties, like being unbiased maximum likelihood estimators. Even though the error distribution in our motivating problem violated these assumptions, choosing weights inversely proportional to the variances substantially improved the solution quality (and speed) of our algorithms; see [9].

3.1 Finding Least Squares Solutions for a Fixed Ordering

One standard approach to solving a least squares problem is to take the partial derivatives of the cost with respect to each of the x_i's, set them equal to 0, and solve. Unfortunately, our cost function is not differentiable due to the absolute value terms. However, for a given *fixed* ordering of the probes we can bypass this difficulty, allowing us to find the placement which minimizes cost for the given ordering. Without loss of generality, assume

$$x_1 < x_2 < \cdots < x_N. \tag{2}$$

Then for a given probe k:

$$\frac{\partial}{\partial x_k}\left(\sum_{i<j} w_{ij}(|x_i - x_j| - d_{ij})^2\right) = \sum_{1\le i\le k-1} 2w_{ik}(x_k - x_i - d_{ik}) - \sum_{k+1\le i\le N} 2w_{ki}(x_i - x_k - d_{ki}). \quad (3)$$

Separating the terms and setting equal to 0, we get for $\frac{\partial}{\partial x_k}$:

$$x_k \sum_{1\le i\le N}(w_{ik}) + \sum_{1\le i\le N}(-w_{ik}x_i) = \sum_{1\le i\le k-1}(w_{ik}d_{ik}) - \sum_{k+1\le i\le N}(w_{ki}d_{ki}). \quad (4)$$

Equation 4 is of the form

$$\mathrm{M}\mathrm{x} = \mathrm{r} \quad (5)$$

where x is the vector of x_i's, M is the matrix defined as:

$$M_{ij} = \begin{cases} -w_{ij} & i \ne j; \\ \sum_{1\le p\le N}(w_{ip}) & i = j; \end{cases} \quad (6)$$

and r is the vector whose k^{th} component r_k is given by the right hand side of Equation 4. Thus, in matrix form, Equation 4 can be written as:

$$\begin{pmatrix} & \cdots & \\ -w_{k1} \cdots & M_{kk} & \cdots \; -w_{kN} \\ & \cdots & \\ & \cdots & \\ & \cdots & \end{pmatrix} \begin{pmatrix} \cdots \\ x_k \\ \cdots \\ \cdots \\ \cdots \end{pmatrix} = \begin{pmatrix} \cdots \\ \sum_{1\le i\le k-1}(w_{ik}d_{ik}) - \sum_{k+1\le i\le N}(w_{ki}d_{ki}) \\ \cdots \\ \cdots \\ \cdots \end{pmatrix}$$

where M_{kk}, the summation term in Equation 6, is the sum of the weights of the measurements from probe k to other probes.

A critical point is that there is no guarantee that the ordering of the probes in the solution of $\mathrm{M}\mathrm{x} = \mathrm{r}$ will respect the ordering (2) used to construct this linear system. However, the solution to this linear system provides useful information in either case.

– If the solution *does* respect the ordering, then it provides the optimal (in the least-squares sense) positioning of the probes with respect to the given ordering, and is a local minimum of the cost function.
– If the solution does *not* respect the ordering, then it gives a lower bound on the cost of the best placement with that ordering. This is true since solution to $\mathrm{M}\mathrm{x} = \mathrm{r}$ gives the minimum of $\sum_{i<j} w_{ij}(x_j - x_i - d_{ij})^2$ over all x, which is certainly no greater than the minimum over the region $\{\mathrm{x} \mid x_1 < x_2 < \cdots < x_N\}$. Furthermore, in this case the given ordering is not the optimal one, since the solution to $\mathrm{M}\mathrm{x} = \mathrm{r}$ gives

an ordering having a lower cost. This holds since for each pair $i < j$ for which $x_i > x_j$, we have

$$(j\, x_i - x_j\, j - d_{ij})^2 = (x_i - x_j - d_{ij})^2 < (x_j - x_i - d_{ij})^2:$$

In other words, at the point x solving $\mathsf{M}\mathsf{x} = \mathsf{r}$, each term in the true cost function is less that or equal to the corresponding term in the restricted cost function built assuming the ordering $x_1 < x_2 < \quad < x_N$, and so that ordering cannot be optimal.

These are the key observations on which our algorithms are built. The problem has been reduced from a continuous optimization problem to a discrete one—that of computing the matrix solution over all probe orderings and choosing the ordering with the lowest cost. Our branch and bound algorithm searches over all possible probe orderings, using an extension of the method above to bound the cost of large sets of possible orderings, provably finding the one(s) of minimum cost. The branch and bound algorithm is described more fully in Section 5. Our heuristic search algorithm is even simpler. Starting from many random orderings, it merely iterates the process described above until it reaches a local minimum. Empirically, this is highly effective at finding the global minimum quickly. This is described more fully in the next section.

4 Heuristic Search

As outlined in the previous section, solution to the linear system constructed for any fixed order of the probes either gives the optimal placement for probes in that order, which is a local minimum of the cost function, or gives a placement with another ordering ^{0} at which the cost function is lower than it is at any placement respecting . Our heuristic search algorithm is simply “iterated linear solve”:

1. choose a random ordering ;
2. set up the linear system corresponding to that ordering;
3. solve it;
4. if the resulting order ^{0} is equal to , record that as a potential minimum;
5. if $^0 \neq$, replace by ^{0} and return to step 2.

Finally, we repeat this entire process for many random initial orderings, and report the lowest cost solution found. In different tests, we either did a fixed number of random starts, usually 300, or repeated until the known optimal solution was found.

One nice feature of the matrix formulation is that M is independent of the ordering of the probes. When solving this system by LU decomposition (as in [8]), this means that once we perform an initial $O(N^3)$ operation on M, we can find a solution in $O(N^2)$ time per ordering, the time required to generate the (order-dependent) vector r and backsolve.

5 Branch and Bound

Our branch and bound algorithm constructs a search tree over probe orderings. The leaves will be complete orderings and the interior nodes will be partially specified orderings. There are two basic approaches to structuring the search tree. In the first approach, shown in Figure 4, the children of a node P in the tree will be the ordering of probes at node P augmented with a new probe in all possible positions among the probes ordered at P.

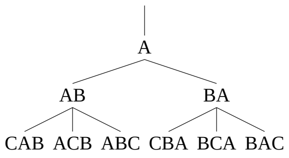

Fig. 4. At a node, the children are orderings in which an additional probe is placed in all possible positions with respect to the ordered probes.

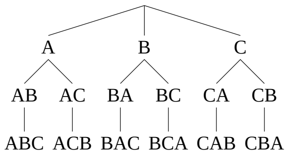

Fig. 5. At a node, the children are orderings in which each of the unordered probes has been placed to the right of the rightmost ordered probe.

For the second approach, in Figure 5, the ordering of a child of an interior node P will be the ordering of P augmented by a probe placed adjacent to the rightmost ordered probe in P.

In either approach, as is typical in branch and bound algorithms, little of the ordering is specified at higher levels of the search tree, hence the bounds computed there will be weak and pruning will be rare. Given this, the first approach has the advantage that the branching factor is much lower near the root of the tree compared to the second approach, e.g. 3 versus $N - 3$ on the third level. On the other hand, the second approach has the advantage that more information is known about the partially specified ordering at an interior node P, namely that all unordered probes lie to the right of the rightmost specified probe in every node of the subtree rooted at P. We can exploit this to give a strengthened bound at internal nodes compared to approach one. In our experiments [11], approach two outperformed approach one by nearly a factor of two both in run time and in number of tree nodes visited. Throughout the remainder of this paper, we will only consider approach two.

Our branch and bound algorithm searches through nodes in a tree, pruning a node if its cost is greater than the lowest cost found in a leaf node so far. At a leaf node in the tree, we compute the cost of the ordering as described in Section 3.1. At an interior node, the cost function must be a lower bound on the cost of all nodes in the subtree to allow us to possibly prune the subtree. In this section, we describe a simple cost function based on least squares.

Consider an interior node such as that in Figure 6. In this picture of an interior node, the circles represent probes, and the edges represent the existence of a measurement between two probes. Probes A, B, C, and D have been ordered (in that order). Probes E

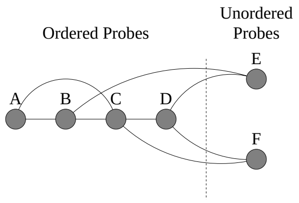

Fig. 6. An Interior Node

and F are unordered with respect to each other, but both will appear to the right of probe D. One way of computing a lower bound on the cost for this node is to consider only the measurements between the ordered probes. In this case, we compute the cost function by computing the matrix solution (as described in Section 3.1) using a matrix built from only measurements between the ordered probes. This is done by simply pretending the other measurements do not exist, i.e., the terms in M of Equation 5 for measurements that we are not considering are 0, and there is no contribution from them in the r vector.

We note that the cost function described here is ineffective at high levels in the tree (where nodes will reflect probe orderings with few constraints). In particular, the cost function described evaluates to zero for the first and second level in the tree (when only one or two probes are ordered). However, consider the measurements in Figure 6 between ordered probes $C; D$, and unordered probe F. Even though the position of F is undetermined with respect to E, we know that F will be to the right of D. This allows us to remove the absolute value sign in the sum of squares terms of Equation 1 for the measurements between F and $C; D$ and include these terms in the cost function computation. Thus, for an interior node, as well as considering all edges between ordered nodes, we can consider edges between ordered nodes and unordered nodes when constructing the cost function for the node. This improvement potentially allows us to compute a non-zero cost function for nodes as high as the second level in the tree (when only two probes are ordered). With this improvement, the only constraints we are not considering at a node are those between unordered probes. The bound function described here is the one we use in the simulations reported in Section 6, Results.

The cost of an interior node P computed in this way will be a lower bound on the cost of all nodes in the subtree rooted at P, since nodes in the subtree impose additional constraints on the ordering, never remove constraints, and each additional constraint adds additional non-negative terms to the cost function.

An additional issue which has a strong effect on the performance of our branch and bound algorithm is initialization of the bound. Starting the algorithm with a conservative default value for the bound (like $+1$) results in very poor pruning until a reasonably good solution is encountered. Instead we first run the local search algorithm from a few random starting orderings. Empirically, this will quickly locate a good solution, facilitating good pruning from the beginning. In our experiments, branch and bound removes 100-1000 times as many nodes as a result [9].

There is one remaining detail to be specified—we need to modify the construction of the M of Equation 5. As it stands, the linear system $\mathsf{M}\mathsf{x} = \mathsf{r}$ of Section 3.1 is under-constrained (the rank of the null-space of M is non-zero). Because the system is constructed from relative orderings between the probes, there is one degree of freedom: the absolute position of the probes. This is remedied by modifying the system to arbitrarily place probe 1 at location $x_1 = 0$. There may be additional degrees of freedom in the solutions. In particular, at high levels in the tree the small set of ordered probes may be partitioned into several disconnected components whose relative positions are unconstrained. These situations are handled similarly; see [9] for details.

Finally, we remark that the cost computed by the techniques outlined above is a lower bound, but not necessarily an attainable bound, on the cost of any ordering consistent with that specified at a search tree node. In particular, in the case where the solution to the linear system $\mathsf{M}\mathsf{x} = \mathsf{r}$ exhibits a different ordering than the one from which the system was constructed, we know that the bound is not attainable by the desired ordering. It is still valid to use this bound to prune the search tree, since we know the bound is attainable by some (other) ordering. However, pruning could be improved if a higher lower bound could be computed in these cases. One possible approach to doing so would be to use quadratic programming—minimization of the quadratic objective function in Equation 1 subject to the linear constraints in Equation 2 is a convex quadratic optimization problem, for which polynomial time algorithms are known; see, for example, [4]. However, it is not clear whether the increased pruning efficiency would offset the extra computational cost of using the more elaborate quadratic programming algorithm. Preliminary experiments have been inconclusive [11]

We now present the results of experiments performed on the heuristic search and branch and bound algorithms.

6 Results

We ran multiple simulations to assess the performance of the two algorithms and also to gauge the sensitivity of the algorithms to different parameters. We summarize the main results here; see [9, 11] for further details.

The experiments described below were all run on synthetic data generated in accord with the motivating problem presented in Section 1. Probes were placed uniformly at random, except that adjacent probes were separated by minimum distance of approximately 3% of the average spacing. Approximately 50% of the probe pairs were “measured,” were measurement consisted of drawing a random sample from a certain distribution whose mean was the actual distance between the probes. Data sets having more than one connected component or certain other anomalies were filtered out. The results do not seem to be overly sensitive to any of these parameters.

As a measure of the quality of the solution found by the algorithms, we used RMS error—the square root of the mean squared difference between the true and calculated positions of the probes. While this quantity varied from run to run, the median value was 10%–15% of the average interprobe distance, which is reasonably good considering the variance of the “measurements.”

We present the total time for the branch and bound algorithm using weighted least-squares in Figure 7, and the total time for heuristic search in Figure 8. Each point in the graph is a problem instance.

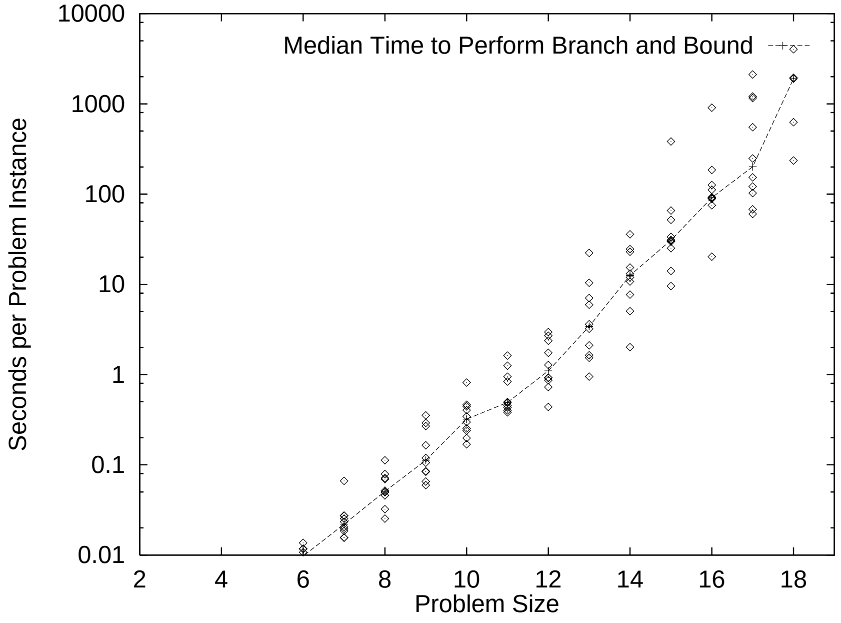

Fig. 7. Time for Branch and Bound (Seconds).

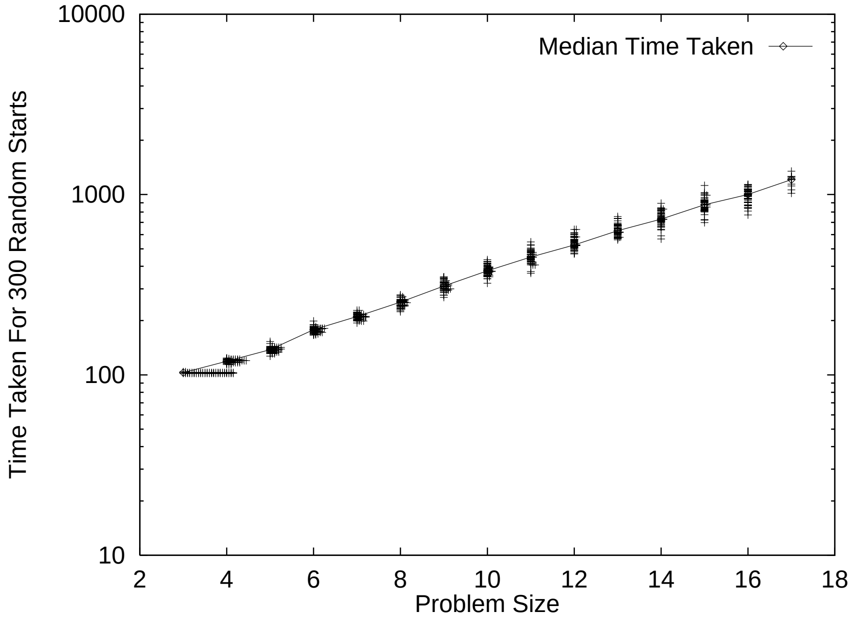

Fig. 8. Time for Heuristic Search. Trials showing identical times spread horizontally for clarity. (300 random starts; Milliseconds).

We can see that the running time of the branch and bound algorithm is exponential, as is expected, with time increasing roughly as 2.8^N. Note that at the far right of the graph, the most time taken to solve a problem of 18 probes was about 70 minutes. Since the number of nodes in a search tree of a problem that size is around 10^{16}, we can see that the pruning heuristic is quite effective; in fact it visited on the order of 10^6 nodes.

The performance of heuristic search is in some ways more difficult to assess. For a problem size of 18, the 300 random starts of heuristic search took about 1 second. The surprisingly stable growth rate also appears to be exponential, but grows much more slowly, roughly as 1.2^N. At this rate, problems of size 30 would be solvable in a few minutes and problems of size 50 in under an hour. However, note that 300 random starts is a very arbitrary choice. In most trials ($> 90\%$), the method finds the globally optimal solution within 10 random starts. In a few "hard core" cases, however, it can take several thousand starts to find the global. Unfortunately, of course, using heuristic search alone, one cannot tell when the globally optimal solution has been reached. (We compared to the provably optimal results from branch and bound.) Nevertheless, the method seems to be a powerful one and worth further study.

Timing experiments where performed on a 100 MHz DEC AlphaStation 200 4/100 with 96MB of memory. The C code was not optimized beyond the optimizations described here (and in [9, 11]). In particular, the LU decomposition routine was copied without modification from [8]. Since the process size for these algorithms was around 3 MB, and since the simulation code is CPU intensive, the time due to non-CPU activities (such as paging) does not significantly affect the results shown.

7 Conclusions

We have presented two search algorithms, a branch and bound algorithm and a heuristic local search algorithm, both of which attempt to minimize a weighted least-squares cost function to solve a one dimensional point placement problem.

Due to the exponential nature of the branch and bound algorithm, it is unlikely that it will scale to larger problem sizes. However, it does provide good performance on problems of 18-20 probes, large enough to be of practical use. Since it finds the global minimum, it is also useful as a benchmark against which to compare other algorithms.

The local search algorithm performed surprisingly well, finding optimal solutions in seconds and appears capable of handling much larger problem instances.

8 Acknowledgments

We would like to thank Brian Pinkerton, Barb Trask, Ger van den Engh, Harry Yeung, and the Statistics Consulting Group for their thoughtful comments and generous assistance.

References

[1] Timothy Bishop. Linkage analysis: Progress and problems. *Phil. Trans. R. Soc. Lond.*, 344:337–343, 1994.

[2] Michael Boehnke, Kenneth Lange, and David Cox. Statistical methods for multipoint radiation hybrid mapping. *Am. J. Hum. Genet.*, 49:1174–1188, 1991.

[3] Kenneth H. Buetow and Aravinda Chakravarti. Multipoint gene mapping using seriation. I. General methods. *Am. J. Hum. Genet.*, 41:180–188, 1987.

[4] Donald Goldfarb and Shucheng Liu. An $O(n^3 L)$ primal-dual potential reduction algorithm for solving convex quadratic programs. *Mathematical Programming*, 61:161–170, 1993.

[5] Brendan Marshall Mumey. A fast heuristic algorithm for a probe mapping problem. In *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pages 191–197, 1997.

[6] Brendan Marshall Mumey. *Some Computational Problems from Genomic Mapping*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1997.

[7] William R. Newell, Richard Mott, S. Beck, and Hans Lehrach. Construction of genetic maps using distance geometry. *Genomics*, 30:59–70, 1995.

[8] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[9] Joshua Redstone and Walter L. Ruzzo. Algorithms for ordering DNA probes on chromosomes. Technical Report UW-CSE-98-12-04, Department of Computer Science and Engineering, University of Washington, December 1998.

[10] Ger van den Engh, Ranier Sachs, and Barbara J. Trask. Estimating genomic distance from DNA sequence location in cell nuclei by a random walk model. *Science*, 257:1410–1412, 4 September 1992.

[11] Harry Yeung and Walter L. Ruzzo. Algorithms for determining DNA sequence on chromosomes. Unpublished, March 1997.

Duality in ATM Layout Problems

Shmuel Zaks

Department of Computer Science, Technion, Haifa, Israel.
zaks@cs.technion.ac.il, http://www.cs.technion.ac.il/ zaks

Abstract. We present certain dualities occuring in problems of design of virtual path layouts in ATM networks. We concentrate on the one-to-many problem on a chain network, in which one constructs a set of paths, that enable connecting one vertex with all others in the network. We consider the parameters of load (the maximum number of paths that go through any single edge) and hop count (the maximum number of paths traversed by any single message). Optimal results are known for the cases where the routes are shortest paths and for the general case of unrestricted paths. These solutions are symmetric with respect to the two parameters of load and hop count, and thus suggest duality between these two.

We discuss these dualities from various points of view. The trivial one follows from corresponding recurrence relations. We then present various one-to-one correspondences. In the case of shortest paths layouts we use binary trees and lattice paths (that use horizontal and vertical steps). In the general case we use ternary trees, lattice paths (that use horizontal, vertical and diagonal steps), and high dimensional spheres. These correspondences shed light on the structure of the optimal solutions, and simplify some of the proofs, especially for the optimal average case designs.

1 Introduction

In this paper we study path layouts in ATM networks, in which pairs of nodes exchange messages along pre-de ned paths in the network, termed *virtual paths*. Given a physical network, the problem is to design these paths optimally. Each such design forms a layout of paths in the network, and each connection between two nodes must consist of a concatenation of such virtual paths. The smallest number of these paths between two nodes is termed the *hop count* for these nodes, and the *load* (or *congestion*) of a layout is the maximum number of virtual paths that go through any (physical) communication line. The two principal parameters that determine the optimality of the layout are the maximum congestion of any communication line and the maximum hop count between any two nodes. The hop count corresponds to the time to set up a connection between the two nodes, and the congestion measures the load of the routing tables at the nodes.

Two problems that have been recently studied are the *one-to-all* (or *broadcast*) problem (e.g., [CGZ94, GWZ95, G95, DFZ97]), and the *all-to-all* problem (see, e.g., [CGZ94, G95, KKP95, SV96, ABCRS99, DFZ97]), in which one wishes to measure the hop count from one specified node (or all nodes) in the network to all other nodes. In what follows we always consider the one-to-all problem.

Given bounds on the load L and the hop count H between a given node termed *root* and all the other nodes in a layout, we look for the maximum number of nodes for which such a solution exists, satisfying these bounds. Considering a chain network, where the leftmost vertex has to be the root, and where each path traversed by a message must be a shortest path, a family of ordered trees $T_{short}(L,H)$ was presented in [GWZ95], within which an optimal solution can be found, for a chain of length N, with N bounded by $\binom{L+H}{L}$. This number, which is symmetric in H and L, is also equal to the number of lattice paths from $(0,0)$ to (L,H), that use horizontal and vertical steps. Optimal bounds for this shortest path case were also derived for the average case , which also turned out to be symmetric in H and L.

Considering the same problem but without the shortest path restriction, termed the general path case, a family of tree layouts $T(L,H)$ was introduced in [DFZ97], for a chain of length N, not assuming that the root is located at its leftmost vertex, and with N bounded by $\sum_{i=0}^{\min\{L,H\}} 2^i \binom{L}{i}\binom{H}{i}$ [GW70]. This number, which is also symmetric in H and L, is equal to the number of lattice points within an L-dimensional l_1-Sphere or radius H, and is also equal to the number of lattice paths from $(0,0)$ to (L,H), that use horizontal, vertical or (up-)diagonal steps.

As a consequence, the trees $T(L,H)$ and $T(H,L)$ have the same number of nodes, and so do the trees $T_{short}(L,H)$ and $T_{short}(H,L)$. In this paper we use one-to-one correspondences, using binary and ternary trees, in order to combinatorially explain the duality between these two measures of hop count and load, as reflected by these above symmetries. These correspondences shed more light into the structure of these two families of trees, allowing to find for any optimal layout with N nodes, load L and minimal (or minimal average) hop count H, its *dual* layout, having N nodes, maximal hop count L and minimal (or minimal average) load H, and vice-versa. Moreover, they give one proof for both measures, whereas in the above-mentioned papers these symmetries were only derived as a consequence of the final result; we note that the average-case results were derived by a seemingly-different formulas, whereas the worst-case results were derived by symmetric arguments. In addition, these correspondences also provide a simple proof to a new result concerning the duality of these two parameters in the worst case and the average case analysis for the general path case layouts. Finally, it is shown that an optimal worst case solution for the shortest path and general cases, is also an optimal average case solution in both cases, allowing a simpler characterization of these optimal layouts.

This paper surveys results from various papers. In Section 2 the ATM model is presented, following [CGZ94]. In Section 3 we discuss the optimal solutions; the optimal design for the shortest path case follows the discussion in [GWZ95], and the optimal design for the general case follows the discussion in [DFZ97, F98]. We encounter the duality of the parameters of load and hop count, which follows via recurrence relations. In Section 4 we describe the use of binary and ternary trees to shed more direct light on these duality results; this follows [DFZ97, F98]. Lattice paths and spheres are then used in Section 5 to supply additional points of view for these dualities, following ([DFZ97, F98]). We close with a discussion in Section 6.

2 The Model

We model the underlying communication network as an undirected graph $G = (V, E)$, where the set V of vertices corresponds to the set of switches, and the set E of edges corresponds to the physical links between them.

Definition 1. *A* rooted virtual path layout (*layout* for short) Ψ *is a collection of simple paths in G, termed* virtual paths *(VPs for short), and a vertex $r \in V$ termed the* root *of the layout (denoted $root(\Psi)$).*

Definition 2. *The* load $L(e)$ *of an edge $e \in E$ in a layout Ψ is the number of VPs $\in \Psi$ that include e.*

Definition 3. *The* load $L_{max}(\Psi)$ *of a layout Ψ is* $\max_{e \in E} L(e)$.

Definition 4. *The* hop count $H(v)$ *of a vertex $v \in V$ in a layout Ψ is the minimum number of VPs whose concatenation forms a path in G from v to $root(\Psi)$. If no such VPs exist, define $H(v) = \infty$.*

Definition 5. *The* maximal hop count *of Ψ is $H_{max}(\Psi) =$* $\max_{v \in V} \{H(v)\}$.

In the rest of this paper we assume that the underlying network is a *chain*. We consider two cases: the one in which only shortest paths are allowed, and the second one in which general paths are considered.

To minimize the load, one can use a layout Ψ which has a VP on each physical link, i.e., $L_{max}(\Psi) = 1$, however such a layout has a hop count of $N - 1$. The other extreme is connecting a direct VP from the root to each other vertex, yielding $H_{max} = 1$, but then $L_{max} = N - 1$. For the intermediate cases we need the following definitions.

Definition 6. *$H_{opt}(N, L)$ denotes the* optimal hop count *of any layout Ψ on a chain of N vertices such that $L_{max}(\Psi) \le L$, i.e., $H_{opt}(N, L) \equiv$* $\min_{\Psi} \{H_{max}(\Psi) : L_{max}(\Psi) \le L\}.$

Definition 7. *$L_{opt}(N, H)$ denotes the* optimal load *of any layout Ψ on a chain of N vertices such that $H_{max}(\Psi) \le H$, i.e., $L_{opt}(N, H) \equiv$* $\min_{\Psi} \{L_{max}(\Psi) :$ $H_{max}(\Psi) \le H\}.$

Definition 8. *Two VPs constitute a* crossing *if their endpoints $l_1; l_2$ and $r_1; r_2$ satisfy $l_1 < l_2 < r_1 < r_2$. A layout is called* crossing-free *if no pair of VPs constitute a crossing.*

It is known ([GWZ95, ABCRS99]) that for each performance measure (L_{max}, H_{max}, L_{avg}, H_{avg}) there exists an optimal layout which is crossing-free. In the rest of the paper we restrict ourselves to layouts viewed as a planar (that is, crossing-free) embedding of a tree on the chain, also termed *tree layouts*. Therefore, when no confusion occurs, we refer to each VP in a given layout Ψ an *edge* of Ψ.

$N_{\text{short}}(L; H)$ denotes the length of a longest chain in which one node can broadcast to all others, with at most H hops and a load bounded by L, for the case of shortest paths. The similar measure for the general case is denoted by $N(L; H)$.

3 Optimal Solutions and Their Duality

In this section we present the optimal solutions for layouts, when messages have to travel either along shortest paths or general paths. We'll show the symmetric role played by the load and hop count, and explain it via the corresponding recurrence relations.

3.1 Optimal Virtual Path for the Shortest Path Case

Assuming that the leftmost node in the chain has to broadcast to each node to its right, it is clear that, for given H and L, the largest possible chain for which such a design exists is like the one shows in Fig. 1.

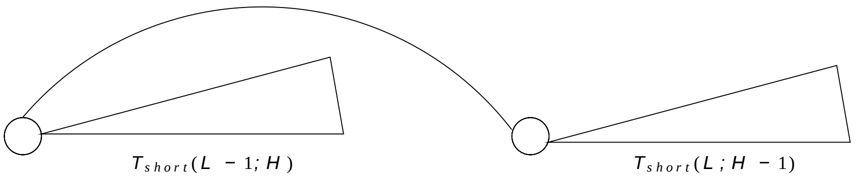

Fig. 1. The tree layout $T_{short}(L; H)$

Recall that $M_{\text{short}}(L, H)$ is the length of the longest chain in which a design exists, for a broadcast from the leftmost node to all others, for given parameters H and L. $M_{\text{short}}(L, H)$ clearly satisfies the following recurrence relation:

$$M_{\text{short}}(0; H) = M_{\text{short}}(L; 0) = 1 \quad \forall H; L \geq 0 \tag{1}$$
$$M_{\text{short}}(L; H) = M_{\text{short}}(L; H-1) + M_{\text{short}}(L-1; H) \quad \forall H; L > 0 .$$

It easily follows that

$$M_{\text{short}}(L;H) = \binom{L+H}{H} : \qquad (2)$$

This design is clearly symmetric in H and L, which establishes the rst result in which the load and hop count play symmetric roles.

Note that it is clear that the maximal number of nodes in a chain , $N_{\text{short}}(L;H)$, to which one node can broadcast using shortest paths, satis es

$$N_{\text{short}}(L;H) = 2\binom{L+H}{H} - 1 : \qquad (3)$$

The above discussion, and Fig. 1, clearly give rise to the trees $T_{short}(L;H)$ de ned as follows.

De nition 9. *The tree layout $T_{short}(L;H)$ is de ned recursively as follows. $T_{short}(L;0)$ and $T_{short}(0;H)$ are tree layouts with a unique node. Otherwise, the root of a tree layout $T_{short}(L;H)$ is the leftmost node of a $T_{short}(L-1;H)$ tree layout, and it is also the leftmost node of a tree layout $T_{short}(L;H-1)$*

Using these trees, it is easy to show that $L_{max}(T_{short}(L;H)) = L$ and $H_{max}(T_{short}(L;H)) = H$. The following two theorems follow:

Theorem 1. *Consider a chain of N vertices and a maximal load requirement L. Let H be such that*

$$\binom{L+H-1}{L} < N \binom{L+H}{L} :$$

Then $H_{opt}(N;L) = H$.

Theorem 2. *Consider a chain of N vertices and a maximal hop requirement H. Let L be such that*

$$\binom{L+H-1}{H} < N \binom{L+H}{H} :$$

Then $L_{opt}(N;H) = L$.

Optimal bounds were also derived in [GWZ95, GWZ97] for the average case, using dynamic programming; the results use di erent recursive constructions, but end up in structures that are symmetric in H and L. These results are stated as follows:

Theorem 3. *Let n and H be given. Let L be the largest integer such that N $\binom{L+H}{L}$, and let $r = N - \binom{L+H}{L}$. Then*

$$L_{tot}(N;H) = H\binom{L+H}{L-1} + r(L+1):$$

Theorem 4. *Let N and L be given. Let H be the maximal such that $N \geq \binom{L+H}{H}$, and let $r = N - \binom{L+H}{H}$. Then*

$$H_{tot}(N; L) = L \binom{L+H}{H-1} + r(H+1).$$

3.2 Optimal Virtual Path for the General Case

In the case where not only shortest paths are traversed, a new family of optimal tree layouts $T(L; H)$ is now presented.

Definition 10. *The tree layout $T(L; H)$ is defined recursively as follows. $T_{right}(L; 0)$, $T_{right}(0; H)$, $T_{left}(L; 0)$ and $T_{left}(0; H)$ are tree layouts with a unique node. Otherwise, the root r is also the rightmost node of a tree layout $T_{right}(L; H)$ and the leftmost node of a tree layout $T_{left}(L; H)$, when the tree layouts $T_{left}(L; H)$ and $T_{right}(L; H)$ are also defined recursively as follows. The root of a tree layout $T_{left}(L; H)$ is the leftmost node of a $T_{left}(L-1; H)$ tree layout, and it is also connected to a node which is the root of a tree layout $T_{right}(L-1; H-1)$ and a tree layout $T_{left}(L; H-1)$ (see Fig. 2). Note that the root of $T_{left}(L; H)$ is its leftmost node. The tree layout $T_{right}(L; H)$ is defined as the mirror image of $T_{left}(L; H)$.*

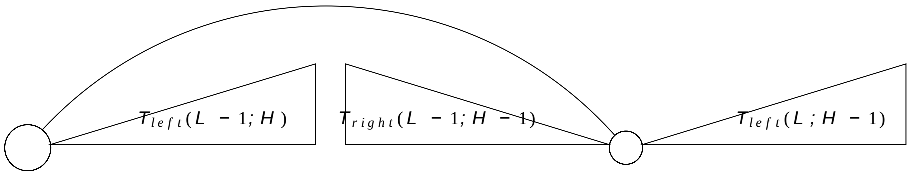

Fig. 2. $T_{left}(L; H)$ recursive definition

Denote by $N(L, H)$ the longest chain in which it is possible to connect one node to all others, with at most H hops and the load bounded by L. From the above, it is clear that this chain is constructed from two chains as above, glued at their root. $N(L, H)$ clearly satisfies the following recurrence relation:

$$N(0; H) = N(L; 0) = 1 \quad \forall H; L \geq 0 \tag{4}$$
$$N(L; H) = N(L; H-1) + N(L-1; H) + N(L-1; H-1) \quad \forall H; L > 0.$$

Again, the symmetric role of the hop count and the load are clear both from the definition of the corresponding trees and from the recurrence relations that compute their sizes.

It is known ([GW70]) that the solution to the recurrence relation (4) is given by

$$N = \sum_{i=0}^{\min\{L,H\}} 2^i \binom{L}{i}\binom{H}{i}. \tag{5}$$

4 Duality: Binary Trees and Ternary Trees

We saw in Section 3 that the layouts $T_{short}(L,H)$ and $T_{short}(H,L)$ and also $T(L,H)$ and $T(H,L)$ have the same number of vertices. We now turn to show that each pair within these virtual path layouts are, actually, quite strongly related. In Section 4.1 we deal with layouts that use shortest-length paths, and show their close relations to a certain class of binary trees, and in Section 4.2 we deal with the general layouts and show their close relations to a certain class of ternary trees.

4.1 $T_{short}(L,H)$ and Binary Trees

In this section we show how to transform any layout Ψ with hop count bounded by H and load bounded by L for layouts using only shortest paths, into a layout $\overline{\Psi}$ (its *dual*) with hop count bounded by L and load bounded by H. In particular, this mapping will transform $T_{short}(L,H)$ into $T_{short}(H,L)$.

To show this, we use transformation between any layout with x edges (VPs) and binary trees with x nodes (in a binary tree, each internal node has a left child and/or a right child). We'll derive our main correspondence between $T_{short}(H,L)$ and $T_{short}(L,H)$ for $x = N - 1$, where $N = \binom{L+H}{L}$. Our correspondence is done in three steps, as follows.

Step 1: Given a planar layout Ψ we transform it into a binary tree $T = b(\Psi)$, under which each edge e is mapped to a node $b(e)$, as follows. Let $e = (r,v)$ be the edge outgoing the root r to the rightmost vertex (to which there is a VP; we call this a *1-level* edge). This edge e is mapped to the root $b(r)$ of T. Remove e from Ψ. As a consequence, two layouts remain: Ψ_1 with root r and Ψ_2 with root v, when their roots are located at the leftmost vertices of both layouts. Recursively, the left child of node $b(e)$ will be $b(\Psi_1)$ and its right child will be $b(\Psi_2)$. If any of the layouts Ψ is empty, so is its image $b(\Psi)$ (in other words, we can stop when a Ψ that consists of a single edge is mapped to a binary tree that consists of a single vertex).

Step 2: Build a binary tree $\overline{T}$, which is a reflection of T (that is, we exchange the left child and the right child of each vertex).

Step 3: We transform back the binary tree $\overline{T}$ into the (unique) layout $\overline{\Psi}$ such that $b(\overline{\Psi}) = \overline{T}$

Example 1. In Fig. 3 the layouts for $L = 2, H = 3$ and $L = 3, H = 2$ are shown, together with the corresponding trees $T_{short}(2,3)$ and $T_{short}(3,2)$, and the corresponding binary trees constructed as explained above. The edge e in the layout $T_{short}(3,2)$ is assigned the vertex $b(e)$ in the corresponding tree $b(T_{short}(3,2))$.

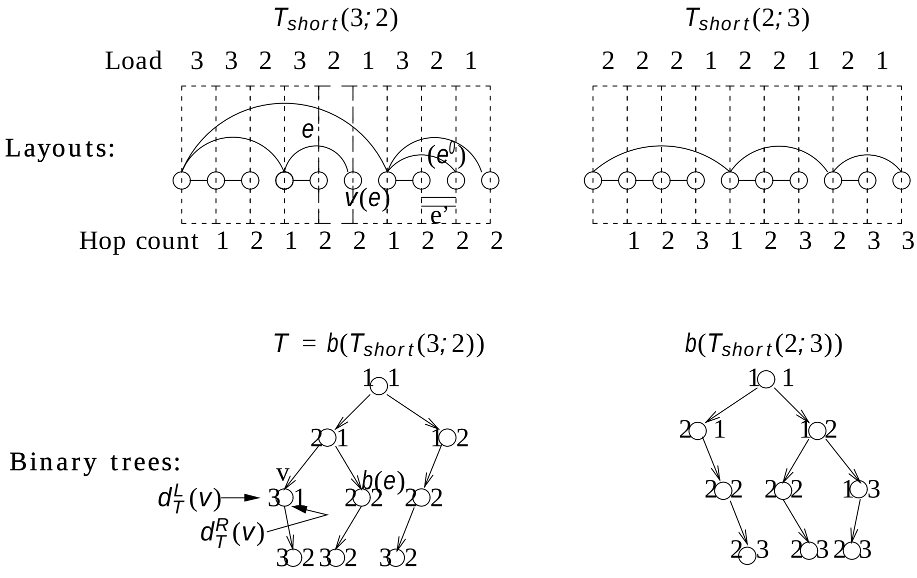

Fig. 3. An example of the transformation using binary trees

Given a non-crossing layout Ψ, we de ne the *level* of an edge e in Ψ, denoted $level_{\Psi}(e)$ (or $level(e)$ for short), to be one plus the number of edges above e in Ψ. In addition, to each edge e of the layout Ψ we assign its farthest end-point from the root, $v(e)$.

Example 2. In Fig. 3 the edge e in the layout $T_{short}(3,2)$ is assigned the vertex $v(e)$ in this layout, and its level $level(e)$ is 2.

One of our key observations is the following theorem:

Theorem 5. *For every H and L, the trees $b(T_{short}(L;H))$ and $b(T_{short}(H;L))$ are reflections of each other.*

This clearly establishes a one-to-one mapping between these trees, and thus establishes the required duality.

To further investigate the structure of these trees, we now turn to explore the properties of the binary trees that we have de ned above. We prove the following theorem:

Theorem 6. *Given a layout Ψ, let $T = b(\Psi)$ be the binary tree assigned to it by the transformation above. Let $d_T^L(v)$ ($d_T^R(v)$) be equal to one plus the number of left (right) steps in the path from the root r to v, for every node v in T. Then, for every edge e in the layout Ψ:*

1. $H_\Psi(v(e)) = d^R_T(b(e))$, *and*
2. $level(e) = d^L_T(b(e))$.

Given a non-crossing layout Ψ, for each physical link e^0 we assign an edge (e^0) in Ψ that includes it and is of highest level (such a path exists due to the connectivity and planarity of the layout; see edge e^0 and physical edge (e^0) in Fig. 3). It can be easily proved that:

Lemma 1. *Given a non-crossing tree layout Ψ, the mapping of a physical link e^0 to an edge (e^0) described above is one-to-one.*

Proposition 1. *Given a non-crossing tree layout Ψ over a physical network, let $T = b(\Psi)$ be the binary tree assigned to it. Then $L(e^0) = level((e^0))$ for every edge e^0 in the physical network.*

Given a layout Ψ over a chain network, if we consider the multiset $fd^R_T(v)jv\ 2\ b(\Psi)g$ we get exactly the multiset of hop counts of the vertices of this network (by Theorem 6), and if we consider the multiset $fd^L_T(v)jv\ 2\ b(\Psi)g$ we get exactly the multiset of loads of the physical links of this network (by Theorem 6 and Proposition 1). By using this and nding the dual layout $\overline{\Psi}$ with the multisets $fd^R_T(v)jv\ 2\ b(\overline{\Psi})g$ of hop counts of its vertices and $fd^L_T(v)jv\ 2\ b(\overline{\Psi})g$ of loads of its physical edges of $\overline{\Psi}$, we observe that the multiset of hop counts of $\overline{\Psi}$ is exactly the multiset of load of Ψ, and the multiset of loads of $\overline{\Psi}$ is also the multiset of hop counts of Ψ, thus deriving a complete combinatorial explanation for the symmetric results of Section 3.1 for either the worst case trees or average case trees:

Theorem 7. *Given an optimal layout Ψ with N nodes, load bounded by L and optimal hop count $H_{opt}(N; L)$, its dual layout $\overline{\Psi}$ has N nodes, hop count bounded by L and optimal load $H_{opt}(N; L)$.*

Theorem 8. *Given an optimal layout Ψ with N nodes, hop count bounded by H and optimal load $L_{opt}(N; H)$, its dual layout $\overline{\Psi}$ has N nodes, load bounded by H and optimal hop count $L_{opt}(N; H)$.*

Theorem 9. *Given an optimal layout Ψ with N nodes, load bounded by L and optimal average hop count, its dual layout $\overline{\Psi}$ has N nodes, hop count bounded by L and optimal average load.*

Theorem 10. *Given an optimal layout Ψ with N nodes, hop count bounded by H and optimal average load, its dual layout $\overline{\Psi}$ has N nodes, load bounded by H and optimal average hop count.*

4.2 $T(L;H)$ and Ternary Trees

We now extend the technique developed in Section 4.1 to general path case layouts; we show how to transform any layout Ψ with hop count bounded by H and load bounded by L into a layout $\overline{\Psi}$ (its *dual*) with hop count bounded by L and load bounded by H. In particular, this mapping will transform $T(L;H)$ into $T(H;L)$.

To show this, we use transformation between any layout with x edges (VP s) and ternary trees with x nodes (in a ternary tree, each internal node has a left child and/or a middle child and/or a right child). Our correspondence is done in three steps, as follows.
Step 1: Given a planar layout Ψ we transform it into a ternary tree $T = t(\Psi)$, under which each edge e is mapped to a node $t(e)$, as follows. Let $e = (r;v)$ be the edge outgoing the root r to the rightmost vertex (to which there is a VP; we call this a *1-level* edge). This edge e is mapped to the root $t(r)$ of T. Remove e from Ψ. As a consequence,three layouts remain: Ψ_1 with root r and and Ψ_3 with root v (when their roots are located at the leftmost vertices of both layouts) and Ψ_2 with root v (when v is its rightmost vertex). Recursively, the left child of node $t(e)$ will be $t(\Psi_1)$, its middle child will be $t(\Psi_2)$ and its right child will be $t(\Psi_3)$. If any of the layouts Ψ is empty, so is its image $t(\Psi)$ (in other words, we can stop when a Ψ that consists of a single edge is mapped to a ternary tree that consists of a single vertex).
Step 2: Build a ternary tree $\overline{T}$, which is a reflection of T (that is, we exchange the left child and the right child of each vertex; the middle child does not change).
Step 3: We transform back the ternary tree $\overline{T}$ into the (unique) layout $\overline{\Psi}$ such that $t(\overline{\Psi}) = \overline{T}$

See Fig. 4 for an example of this transformation.

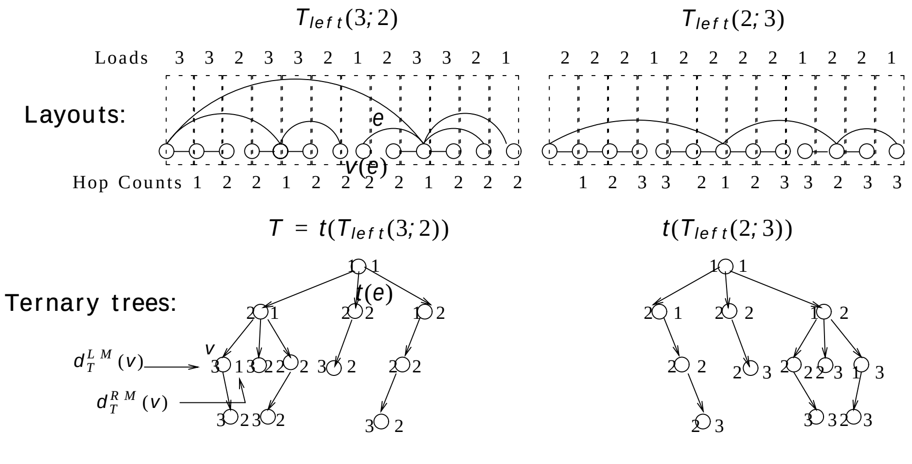

Fig. 4. An example of the transformation using ternary trees

One of our key observations is the following theorem:

Theorem 11. *For every H and L, the trees $t(T(L;H))$ and $t(T(H;L))$ are reflections of each other.*

This clearly establishes a one-to-one mapping between these trees, and thus establishes the required duality.

To further investigate the structure of these trees, we now turn to explore the properties of the ternary trees that we have de ned above. We prove the following theorem. Note that the de nition of $level$ (of an edge) and (of a physical link) remain exactly the same as in Section 4.1.

Theorem 12. *Given a layout Ψ, let $T = t(\Psi)$ be the ternary tree assigned to it by the transformation above. Let $d^{LM}_T(v)$ ($d^{RM}_T(v)$) be equal to one plus the number of left and middle (right and middle) steps in the path from the root r to v, for every node v in T. Then, for every edge e in the layout Ψ:*

1. $H_\Psi(v(e)) = d^{RM}_T(t(e))$, *and*
2. $level(e) = d^{LM}_T(t(e))$.

Proposition 2. *Given a non-crossing tree layout Ψ over a physical network, let $T = t(\Psi)$ be the ternary tree assigned to it. Then $L(e^0) = level((e^0))$ for every edge e^0 in the physical network.*

Given a layout Ψ over a chain network, if we consider the multiset $fd^{RM}_T(v)jv\ 2\ t(\Psi)g$ we get exactly the multiset of hop counts of the vertices of this network (by Theorem 12), and if we consider the multiset $fd^{LM}_T(v)jv\ 2\ t(\Psi)g$ we get exactly the multiset of loads of the physical links of this network (by Theorem 12 and Proposition 2). By using this and nding the dual layout $\overline{\Psi}$ with the multisets $fd^{RM}_T(v)jv\ 2\ t(\overline{\Psi})g$ of hop counts of its vertices and $fd^{LM}_T(v)jv\ 2\ t(\overline{\Psi})g$ of loads of its physical edges of $\overline{\Psi}$, we observe that the multiset of hop counts of $\overline{\Psi}$ is exactly the multiset of load of Ψ, and the multiset of loads of $\overline{\Psi}$ is also the multiset of hop counts of Ψ, thus deriving a complete combinatorial explanation for the symmetric results of either the worst-case trees or average-case trees in the general path case.

Following the above discussion, we obtain the exact four theorems (Theorems 7, 8, 9 and 10) extended to the general path case layouts.

5 Duality: Lattice Paths and High-Dimensional Spheres

5.1 Lattice Paths

The recurrence relation (1) clearly corresponds to the number of lattice paths from the point (0,0) to the point (L, H), that use only horizontal (right) and vertical (up) steps.

In Fig. 5 each lattice point is labeled with the number of lattice paths from (0,0) to it; the calculation is done by the recurrence relation 1. For the case $L =$

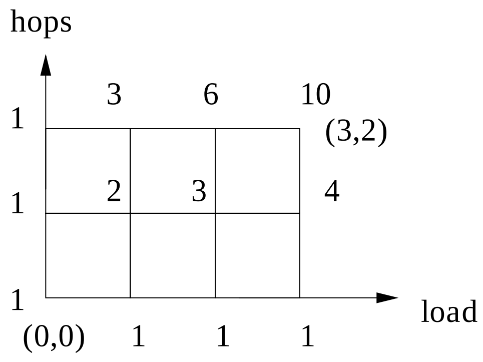

Fig. 5. Lattice paths with regular steps

3 and $H = 2$ one gets $\binom{3+2}{2} = 10$; this corresponds to the number of nodes in the tree $T_{short}(3;2)$ (see Fig. 3), and to the number of paths that go from (0,0) to (3,2).

The recurrence relation (4) clearly corresponds to the number of lattice paths from the point (0,0) to the point (L, H), that use horizontal (right), vertical (up), and diagonal (up-right) steps. In Fig. 6 each lattice point is labeled with the number of lattice paths from (0,0) to it. For the case $L = 3$ and $H = 2$ one gets 25 such paths. This corresponds to the number of nodes in the tree $T(3;2)$

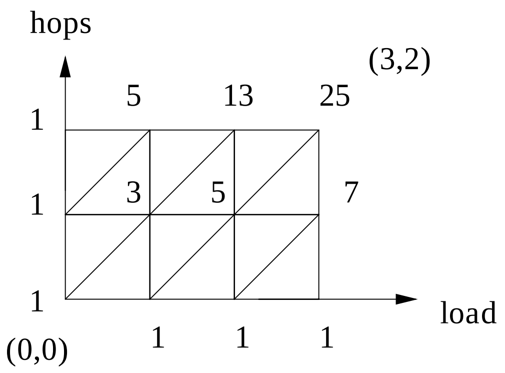

Fig. 6. Lattice paths with regular and diagonal steps

(see Fig. 7), that is constructed of two trees, glued at their roots, the one depicted in Fig. 3 (and containing 13 vertices), and its corresponding reverse tree.

We also refer to these lattice paths in Section 5.2.

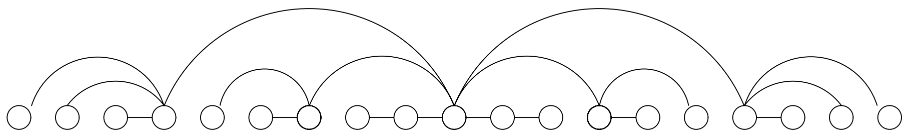

Fig. 7. The tree $T(3; 2)$

5.2 Spheres

Consider the set of lattice points (that is, points with integral coordinates) of an L-dimensional l_1-Sphere of radius H. The points in this sphere are L-dimensional vectors $v = (v_1; v_2; \ldots; v_L)$, where $|v_1| + |v_2| + \ldots + |v_L| \leq H$. Let $|Sp(L; H)|$ be the number of lattice points in this sphere. Let $|Rad(N; L)|$ be the radius of the smallest L-dimensional l_1-Sphere containing at least N internal lattice points.

It can be shown that

Theorem 13. *The tree $T(L; H)$ contains $|Sp(L; H)|$ vertices.*

The exact number of points in this sphere is given by equation (5). (This was studied, in conection with codewords, in [GW70].)

Moreover, we can show that

Theorem 14. *Consider a chain of N vertices and a maximal load requirement L. Then $H_{opt}(N; L) = |Rad(N; L)|$.*

This theorem is proved by showing a one-to-one mapping between the nodes of any layout with hop count bounded by H and load bounded by L into the L-dimensional spheres of radius H. This mapping turns out to be very useful in the analysis of this and related analytical results (see also Section 6).

Using the above correspondences and discussion, it is possible to show that, for either the shortest paths case or the general case, any optimal layout Ψ with N nodes, load bounded by L and optimal hop count, has also optimal average hop count regarding layouts with load bounded by L, and that any optimal layout Ψ with N nodes, hop count bounded by H and optimal load, has also optimal average load regarding layouts with hop count bounded by H.

We now sketch a one-to-one mapping between the set of lattice points of the L-dimensional sphere of radius H and the set of lattice paths from $(0; 0)$ to $(L; H)$ that use horizontal, vertical or (up-)diagonal steps. We first describe a function which maps every vector $v = (v_1; \ldots; v_L)$ in $SP(L; H)$ into such a lattice path. Starting from $(0; 0)$ make $|v_1|$ vertical steps and one horizontal step, make $|v_2|$ vertical steps and one horizontal step,..., make $|v_L|$ vertical steps and one more horizontal step, ending with $H - \sum_{i=1}^{i=l} v_i$ horizontal steps. After that, for every negative v_i component of v, we replace the $|v_i|$th vertical step and the subsequent horizontal step done during the translation of this component by an (up-)diagonal step. A close look at the properties of these paths enables us to further explore the properties of these trees. Returning to the discussion of the layouts $T_{short}(L;H)$ that use only shortest paths, it is possible to nd a similar correspondence between the vertices of these trees and lattice paths from $(0;0)$ to $(L;H)$ that use only vertical and horizontal steps, and to view some properties of these trees using these lattice paths.

6 Discussion

We presented the dual role played by the parameters of load and hop count in optimal designs of virtual path layouts in ATM chain networks, for the cases of shortest paths routes and the general case. We discussed these dualities with the aid of recurrence relations, one-to-one correspondences with binary trees (for the shortest paths case) and ternary trees (for the general case), lattice paths (that use horizontal and vertical steps for the shortest path case, and that also use diagonal steps for the general case), and high dimensional spheres (in the general case). These dualities shed light on the structure the optimal solutions, and simplify some of the proofs.

It might be of interest to further explore such duality relations between these and corresponding parameters (such as load measured at vertices (e.g., [FNP97]) also for other topologies (such as trees ([CGZ94, G95]), meshes or planar graphs ([BG97, BBGG97, G95]); one might also consult the survey in [Z97] for a general discssion of these extensions.

Of special interest in the use of geometry, presented in Section 5.2. This approach provides a clear view for the structure of the solution, and enabled improving results for the all-to-all problem on a ring network (see [DFZ97] for details).

Acknowledgments: I would like to thank my coauthors (Marcelo Feighelstein, Ori Gerstel, Avishai Wool, Israel Cidon and Ye m Dinitz), and Renzo Sprugnoli, Donatella Merlini, Cecilia Verri, Ron Aharoni and Noga Alon for helpful discussions.

References

[ABCRS99] W. Aiello, S. Bhatt, F. Chung, A. Rosenberg, and R. Sitaraman, *Augmented rings networks*, Proceedings of the *6th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Lacanau-Ocean, France, 1999, pp. 1-16.

[BG97] L. Beccheti and C. Gaibisso, *Lower bounds for the virtual path layout problem in ATM networks*, *Proceedings of the 24th Seminar on Theory and Practice of Informatics (SOFSEM)*, Milovny, The Czech Republic, November 1997, pp. 375-382.

[BBGG97] L. Becchetti, P. Bertolazzi, C. Gaibisso and G. Gambosi, *On the design of e cient ATM routing schemes*, submitted, 1997.

[CGZ94] I. Cidon, O. Gerstel and S. Zaks, *A scalable approach to routing in ATM networks. 8th International Workshop on Distributed Algorithms (WDAG)*, Lecture Notes in Computer Science 857, Springer Verlag, Berlin, 1994, pp.209-222.

[DFZ97] Ye. Dinitz, M. Feighelstein and S. Zaks, *On optimal graph embedded into path and rings, with analysis using l_1-spheres*, *23th International Workshop on Graph-Theoretic Concepts in Computer Sciences (WG)*, Berlin, Germany, June 1997.

[F98] M. Feighelstein, *Virtual path layouts for ATM networks with unbounded stretch factor*, , M.Sc. Dissertation, Department of Computer Science, Technion, Haifa, Israel, May 1998.

[FNP97] M. Flammini, E. Nardelli and G. Proietti, *ATM layouts with bounded hop count and congestion*, *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG)*, Saarbrüecken, Germany ,September 1997, pp. 24-26.

[FZ98] M. Feighelstein and S. Zaks, *Duality in chain ATM virtual path layouts*, Proceedings of the *4th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Monte Verita, Ascona, Switzerland, July 24-26, 1997, pp. 228-239.

[G95] O.Gerstel, *Virtual Path Design in ATM Networks*, Ph.D. thesis, Department of Computer Science, Technion, Haifa, Israel, December 1995.

[GW70] S. W. Golomb and L. R. Welch, *Perfect Codes in the Lee Metric and the Packing of Polyominoes*. *SIAM Journal on Applied Math.*,vol.18,no.2, January, 1970, pp. 302-317.

[GWZ95] O. Gerstel, A. Wool and S. Zaks, *Optimal layouts on a chain ATM network*, *Discrete Applied Mathematics* , special issue on Network Communications, 83, 1998, pp. 157-178.

[GWZ97] O. Gerstel, A. Wool and S. Zaks, *Optimal Average-Case Layouts on Chain Networks*, Proceedings of the *Workshop on Algorithmic Aspects of Communication*, Bologna, Italy, July 11-12, 1997.

[KKP95] E. Kranakis, D. Krizanc and A. Pelc, *Hop-congestion tradeo s for ATM networks, 7th IEEE Symp. on Parallel and Distributed Processing*, pp. 662-668.

[SV96] L. Stacho and I. Vrt'o, *Virtual Path Layouts for Some Bounded Degree Networks. 3rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Siena, Italy, June 1996.

[Z97] S. Zaks, *Path Layout in ATM Networks*, *The DIMACS Workshop on Networks in Distributed Computing*, DIMACS Center, Rutgers University, October 26-29, 1997, pp. 145-160.

The Independence Number of Random Interval Graphs

W. Fernandez de la Vega

Laboratoire de Recherche en Informatique
Universite de Paris-Sud, bat 490
91405 Orsay Cedex
lalo@lri.fr

Abstract. It is proved, sharpening previous results of Scheinerman and by analysing an algorithm, that the independence number of the random interval graph, defined as the intersection graph of n intervals whose end points are chosen at random on [0,1], concentrates around $2\sqrt{\frac{n}{\pi}}$.

Key Words: Random Graphs, Analysis of Algorithms, Probabilistic Methods

1 Introduction

Scheinerman defined [1], a random interval graph on n vertices as the intersection graph of n intervals $[X_1, Y_1], \ldots, [X_i, Y_i], \ldots, [X_n, Y_n]$ whose end points are chosen at random on [0,1]. Hence here we start with $2n$ independent random variables $Z_1, Z_2, \ldots, Z_{2n}$ independently and uniformly distributed on $[0, 1]$ and we put, for $1 \leq i \leq n$, $X_i = \min\{Z_{2i-1}, Z_{2i}\}$ and $Y_i = \max\{Z_{2i-1}, Z_{2i}\}$. Scheinerman derived many interesting properties of these graphs. Here we answer one of the questions that Scheinerman left open, namely we derive an asymptotic equivalent for the independence number of these graphs. The main ingredient in our proof is the analysis of an algorithm. Recall that the independence number of a graph is the maximum cardinality of a subset of vertices which span no edge.

Theorem *Let G_n denote the random graph defined as the intersection graph of n intervals whose end points are chosen at random on [0,1]. The independence number $\alpha(G_n)$ of this graph satisfies*

$$\frac{\alpha(G_n)}{2\sqrt{\frac{n}{\pi}}} \to 1$$

in probability as $n \to \infty$.

2 Proof of the Theorem

The proof uses a greedy algorithm. Let again $[X_1, Y_1], \ldots, [X_i, Y_i], \ldots, [X_n, Y_n]$ denote our random intervals. We call X_i (resp. Y_i) the left (resp. the right) end of

$[X_i, Y_i]$. We put $I_i = [X_i, Y_i]$. We also denote by I_i the vertex of G_n corresponding to the interval I_i. We say that $[X_j, Y_j]$ lies at the right of $[X_i, Y_i]$ if we have $X_j > Y_i$. It is clear that we can assume that the leftmost interval representing a vertex of an independent set of maximum cardinality is the interval I_i with leftmost rigth end. This implies by induction that the independent set $\{J_1, \ldots, J_h\}$ of G_n given by following algorithm has maximum cardinality (here each J_k is equal to some I_l).

Algorithm Al n

1. De ne J_1 as the interval I_i with leftmost right extremity and set $k = 1$.

2. If there is no interval I_i lying at the right of J_k put $(G_n) = k$ and stop. Else de ne J_{k+1} as the interval I_i lying at the right of J_k which has the leftmost right extremity, set $k = k + 1$ and go to 2.

This concludes the description of our algorithm.

2.1 Some Preliminary Results

We begin by restating for ease of reference two well known inequalities concerning the tail of the Binomial distribution.

Fact 1(Hoe ding-Cherno bounds) Let $S_{n,p}$ denote the sum of n $\{0, 1\}$ valued independent random variables $X_1, \ldots, X_n$ with $P[X_i = 1] = p,\ 1 \quad i \quad n$. We have then, for $0 \qquad 1$,

$$P[S_{n,p} \quad (1 - \ )np] \quad e^{-\ ^2 np/3} \tag{1}$$

and,

$$P[S_{n,p} \quad (1 + \ )np] \quad e^{-\ ^2 np/2} \tag{2}$$

We will need the following easy results concerning the distribution of the I_j's.

Fact 2 Suppose that $I_1, \ldots, I_m$ are m random intervals contained in the segment $J = [0, l]$ and let x denote the largest distance between the right end of J and the right ends of the I_i's. We have

$$E(x) \quad \frac{l}{2}\sqrt{\frac{\ }{m}},$$

and,

$$E(x^2) \quad \frac{l^2}{m}.$$

Proof. The probability that the right end of I_1 lies at a distance greater than x of the left end of J is equal to $1 - (x/l)^2$. The probability that this is true for every interval I_i is thus equal to $[1 - (x/l)^2]^m$. Hence we have

$$E(x) = -\int_0^l t d[1 - (t/l)^2]^m = \int_0^l [1 - (t/l)^2]^m dt$$
$$\int_0^l \exp\{-\frac{mt^2}{l^2}\} dt \quad \frac{l}{2}\sqrt{\frac{\ }{m}}.$$

and

$$E(x^2) = -\int_0^l t^2 d[1-(t/l)^2]^m = 2\int_0^l [1-(t/l)^2]^m t\,dt$$
$$\le 2\int_0^l e^{-\frac{mt^2}{l^2}} t\,dt \le \frac{l^2}{m}.$$

$\square$

Fact 3 Let $m(t)$ denote the number of intervals I_i which lie completely in the interval $[1-t;1]$. We have, with probability 1-o(1),

$$nt^2 - 5t\sqrt{n\log n} \le m(t) \le nt^2 + 5\sqrt{n\log n};\ n^{-1/2} \le t \le 1.$$

Proof. Use the fact that $m(t)$ is for fixed t a binomial random variable with parameters n and $p = t^2$. $\square$

2.2 Analysis of the Algorithm Al n

We set $x_o = 1$ and for each $i \ge 1$ we denote by x_i the distance between the rightmost extremity of the interval J_i and the right extremity of $[0;1]$. We denote by n_i the number of intervals I_j which lie at the right of x_i.(Here x_i and n_i are random variables which are defined for each value of i which does not exceed the independence number). Let us first observe that, since the restriction of the uniform distribution on $[0;1]$ to any subinterval is again uniform, it follows that, conditionally on x_i and n_i, the n_i intervals I_j which are contained in $[x_i;1]$ are independently and uniformly distributed on this interval.

Let us denote by B_i the σ-field generated by the random variables $x_o;n_o;x_1;n_1;\ldots;x_i;n_i$. Let us define

$$i_o = \max\{j : m(x_i) \ge nx_i^2 - x_i\sqrt{n\log n}; 1 \le i \le j\}$$

and j_o as the last value (if any) of the index i for which the inequality $x_i \ge n^{-\frac14}\sqrt{7\log n}$ is satisfied. If there is no such a value we set $j_o = n$. Let $k_o = \min\{i_o;j_o\}$. We define a new process $(y_i;n_i)$ by putting $y_i = x_i$ if $i \le k_o$ and $y_i = x_{k_o}(= y_{k_o})$ if not. Obviously this new process is also measurable relatively to the family of σ-fields (B_i). Since the conditional expectation of the difference $\Delta_{i+1} = y_i - y_{i+1}$ is, for a given y_i, a decreasing function of n_i, we have, for $i \le i_o - 1$, using fact 2 with $l = y_i;\ m = ny_i^2 - y_i\sqrt{n\log n}$,

$$E_{B_i}(\Delta_{i+1}) \le \frac{y_i}{2}\sqrt{\frac{\pi}{ny_i^2 - y_i\sqrt{7n\log n}}} \le \frac12\sqrt{\frac{\pi}{n(1-n^{-1/4})}}$$

and this inequality is obviously also true for $i \ge k_o$ since then Δ_{i+1} vanishes. It implies that the sequence (z_i) defined by

$$z_i = y_i + \frac{i}{2}\sqrt{\frac{\pi}{n(1-n^{-1/4})}} \qquad (3)$$

is a supermartingale relatively to the familly of - elds (B_i). Let us put l = $2(1 - 4n^{-1=4} \log^{1=2} n)(\frac{n(1-n^{-1=4})}{})^{\frac{1}{2}}$ We have

$$Var\, z_l \quad \sum_{i=1} \quad Var\, y_i \quad Cn^{-\frac{1}{2}}$$

since, by facts 2 and 3, each of the terms in this sum is bounded above by $Cn^{-(1=2)}$ where C is an absolute constant. Observing that Ez_i 1 and using Kolmogorov's inequality for martingales, we get

$$P[z_i \quad 1 - n^{-1=4} \log^{1=2} n;\ 1 \quad i \quad l] \quad 1 - \frac{C^2}{\log n}:$$

Replacing i by l in 3 we get the inequality y_l $z_l - 1 + n^{-1=4} \log^{1=2} n$ which gives, with the preceding inequality,

$$P[y_l \quad 3n^{-\frac{1}{4}} \log^{1=2} n] = 1 - o(1);$$

that is, with probability $1 - o(1)$, we have l j_o. Since we have also l i_o with probability $1 - o(1)$ it follows that, again with probability $1 - o(1)$, the process (y_i) coincides with the process (x_i) up to time l. This means that the independence number of our interval graph is at least $l = 2(\frac{n}{})^{1=2}(1 - o(1))$ and concludes the rst part of the proof. For the second part, that is in order to prove that the independence is bounded above by $2(\frac{n}{})^{1=2}(1 + o(1))$, it su ces to repeat essentially the same arguments, using inequalities reverse to those we have used. The details are omitted.

3 Conclusion

By analysing an algorithm, we have obtained an asymptotic equivalent to the independence number of a random interval graph in the sense of Scheinerman. An open problem is to nd an asymptotic equivalent to the independence number of a genuine random interval graph, in the model where every possible interval graph on n vertices is equally likely.

References

1. Scheinerman, E.R., *Random Interval Graphs*, Combinatorica 8 (1988) 357-371.

Online Strategies for Backups

Peter Damaschke

FernUniversität, Theoretische Informatik II
58084 Hagen, Germany
Peter.Damaschke@fernuni-hagen.de

Abstract. We consider strategies for full backups from the viewpoint of competitive analysis of online problems. We concentrate upon the realistic case that faults are rare, i.e. the cost of work between two faults is typically large compared to the cost of one backup. Instead of the (worst-case) competitive ratio we use a re ned and more expressive quality measure, in terms of the average fault frequency. This is not standard in the online algorithm literature. The interesting matter is, roughly speaking, to adapt the backup frequency to the fault frequency, while future faults are unpredictable. We give an asymptotically optimal deterministic strategy and propose a randomized strategy whose expected cost beats the deterministic bound.

1 Introducing the Backup Problem

The main method to protect data from loss (due to power failure, physical destruction of storage media, deletion by accident etc.) is to save the current status of a le or project from time to time. Such a full backup incurs some cost, but loss of data is also very costly and annoying, and faults are unpredictable. So it is a natural question what competitve analysis of online problems can say about backup (or autosave) strategies.

We consider the following basic model. Some le (or le system, project etc.) is being edited, while faults can appear. The cost of work per time is assumed to be constant. Every backup incurs a xed cost as well. We may w.l.o.g. choose the time unit and cost unit in such a way that every unit of working time and every backup incurs cost 1. In case of a fault, all work done after the most recent backup is lost and must be repeated. Before this, we have to recover the last consistent status from the backup, which incurs cost R (a constant ratio of recovery and backup cost). The goal is to minimize the total cost of a piece of work, which is the sum of costs for working time (including repetitions), backups and recoveries.

This seems to be the simplest reasonable model and a good starting point for studying online backup strategies. Perhaps the main criticism is concerned with the constant backup cost. Usually they depend somehow on the amount of changings (such as incremental backups). However, the constant cost assumption is also suitable in some cases, e.g. if the system always saves the entire le though the changings are minor updates, or if the save operation has large constant setup cost whereas the amount of data makes no di erence.

Extended models may, of course, take more aspects into account: faults which have more fatal implications than just repeated work (such as loss of hardly retrievable data etc.), backups which may be faulty in turn; repeated work which is done faster than the first time { to mention a few.

It is usual in competitive analysis to compare the costs of an online strategy to the costs of a clearvoyant (offline) strategy which has prior knowledge of the problem instance (here: the times when faults appear). Their worst-case ratio is called the competitive ratio. We remark that, for our problem, an optimal offline strategy is easy to establish but not useful in developing online strategies, so we omit this subject.

Only a few online problems have been considered where a problem instance merely consists of a sequence of points in time: rent-to-buy (or spin-block), acknowledgement delay, and some special online scheduling problems fall into this category, cf. the references.

2 Rare Faults and a Reformulation

Consider a piece of work that requires p time units and is to be carried out nonstop. A time interval I of length p is earmarked for this job. Let n be the number of faults that will appear during I. The ratio $f = n/p$ is referred to as the average *fault frequency*, with respect to I. In most realistic scenarios f is quite small compared to 1, i.e. the time equivalent of the cost of one backup is much smaller than the average distance between faults, thus we will focus attention to this case of rare faults.

If the online player would know f in advance (but not the times the faults appear at), it were not a bad idea to make a backup every $1/\sqrt{f}$ time units. Namely, the backup cost per time unit is $\sqrt{f}$, and every fault destroys work to the value of at most $1/\sqrt{f}$, hence the average cost of work to be repeated is bounded by $\sqrt{f}$ per time unit. The $\sqrt{f}$ fraction of work which got lost must be fetched later, immediately after I. New faults can occur in this extra time interval, but this adds lower-order terms to the costs. Hence the cost per time is at most $1 + 2\sqrt{f} + Rf$, and the stretch (i.e. ratio of completion time and productive time) is $1 + \sqrt{f}$, subject to $O(f)$ terms.

An offline player can trivially succeed with $1 + (1 + R)f = 1 + O(f)$ average cost per time, making a backup immediately before each fault. (This is not necessarily optimal.) We shall see below that any online strategy incurs cost at least $1 + \sqrt{f}$ per time unit, in the worst case. Hence the competitve ratio still behaves as $1 + \Omega(\sqrt{f})$, for small f. This suggests to simply consider the cost per time incurred by an online strategy, rather than the slightly smaller competitive ratio. In particular, the constant R we have preliminarily introduced appears in $O(f)$ only, thus we will suppress it henceforth. We also omit the summand 1 for normal work and merely consider the additional cost for backups and lost (i.e. repeated) work per time. Throughout the paper, this is called the *excess rate*.

The next simple observation shows that the average fault frequency is intrinsic in the excess rate, as announced. (Similarly one can realize that no competitive online strategy exists if faults are unrestricted.)

Proposition 1. *For any fault frequency f, even if the online player knows f beforehand, no deterministic backup strategy can guarantee an excess rate below $\sqrt{f}$.*

Proof. An adversary partitions the time axis into phases of length $1/f$ and places one fault in each phase by the following rule: If there occurs a gap of $1/\sqrt{f}$ time units between backups then the adversary injects a fault there, hence the online player loses $1/\sqrt{f}$ time, so the ratio of lost time is $\sqrt{f}$. Otherwise, if the distance between consecutive backups were always smaller than $1/\sqrt{f}$ then more than $1/\sqrt{f}$ backups have been made, so the backup cost per time is at least $\sqrt{f}$. In this case the adversary injects a fault at the end of the phase, to keep the fault frequency f. □

Thus the excess rate is some $c\sqrt{f}$, and the main interesting matter is to adapt the backup frequency so as to achieve the best coefficient c, under the realistic assumption that the online player has no previous knowledge of the faults at all.

Some remarks suggest that other objectives would be less interesting: (1) One might also study the excess rate in terms of the smallest fault distance d, rather than the fault frequency f. However this is not very natural, since a pair of faults occuring close together may be an exceptional event, and d can only decrease in time, so using d as a parameter would yield too cautious strategies. Moreover note that, trivially, any online strategy with excess rate $c\sqrt{f}$ has the upper bound $c/\sqrt{d}$, too. (2) For a given strategy one may easily compute that f maximizing $(1+c\sqrt{f})/(1+(1+R)f)$, thus estimating the worst-case competitive ratio, but this number is less meaningful than c itself.

Clearly a $c\sqrt{f}$ upper bound can only hold in case $f > 0$, in other words, if at least one fault occurs. If $f = 0$ then already the first backup yields an infinite coefficient, but if the online player makes no backups at all, speculating on absence of faults, the adversary can foil him by a late fault, which also yields a coefficient not bounded by any constant.

An elegant viewpoint avoiding this $f = 0$ singularity is the following reformulation of the problem. Consider a stream of work whose length is not a priori bounded. Given n, what c can be achieved for the piece of work up to the n-th fault? (If the n-th fault appears after p time units, f is understood to be n/p.) We refer to the corresponding strategies as n-fault strategies. This version of our problem is also supported by

Proposition 2. *If we partition, in retrospect, the work time interval arbitrarily into phases each containing at least one fault, such that we have always achieved an excess rate $c\sqrt{f_i}$, where f_i is the fault frequency in the i-th phase, then the overall excess rate is bounded by $c\sqrt{f}$.*

Proof. Let the i-th phase have length p_i and contain $n_i > 0$ faults. The cost of i-th phase is by assumption $p_i c\sqrt{n_i/p_i} = c\sqrt{n_i p_i}$. Exploiting an elementary inequality, the total cost is bounded by

$$c\sum_i \sqrt{n_i p_i} \le c\sqrt{\sum_i n_i}\sqrt{\sum_i p_i} = c\sqrt{np} = pc\sqrt{f}.$$

□

Therefore, once we have an n-fault strategy with excess rate $c\sqrt{f}$, we may apply it repeatedly to phases of n faults each, thus keeping an overall excess rate $c\sqrt{f}$.

Let us summarize this section: Instead of the competitive ratio we use a nonstandard measure for online backup strategies in terms of an input parameter (fault frequency f), called the excess rate. It behaves as $c\sqrt{f}$ and is much more expressive than a single number for the worst case, as the extra costs heavily depend on the fault frequency. Moreover, we consider an arbitrarily long piece of work until the n-th fault appears, and we try to minimize c for given n.

In the following sections we develop concrete n-fault strategies. In the proofs, we first consider a sort of continuous analogue of the discrete problem. Doing so we can first ignore tedious technicalities and conveniently obtain a heuristic solution which is then discretized. The bound for the discrete strategy is rigorously verified afterwards.

3 Deterministic n-Fault Strategies

We first settle case $n = 1$.

Theorem 1. *There exists a deterministic* 1*-fault strategy with* $c = \sqrt{8}$*, and this is the optimal constant coefficient.*

Proof. Work begins w.l.o.g. at time 0. A backup strategy is specified by the integer-valued function $y(x)$ describing the number of backups made before time x^2. (This quadratic scale will prove convenient.) In order to get a heuristic solution, we admit differentiable real functions y instead of integer-valued ones. That is, we provisionally fix the asymptotic growth of backup numbers in time only, but not the particular backup times.

We have to assign suitable costs to such functions. Assume that the fault occurs between time $(x-1)^2$ and x^2. Then the cost of backups and lost work incurred so far is bounded by $y(x) + 2x/y'(x)$. Namely, at most $y(x)$ backups have been made, and, in the worst case, the fault appears immediately before a planned backup, hence the lost time may equal the distance of consecutive backups. This distance can be roughly estimated as $2x/y'(x)$, since $y'(x)$ is the backup density on the quadratic scale, and $x^2 - (x-1)^2 < 2x$.

Remember that the excess rate $c\sqrt{f}$ is the cost per time. Since $f = 1/x^2$, the coefficient c is the cost divided by x. So our y and c must satisfy $y(x)+2x/y'(x)$

cx for all x. We can assume equality, since every backup may be deferred until coefficient c is reached. The resulting differential equation $y + 2x/y' = cx$ with $y(0) = 0$ has the solution $y = ax$ with suitable constant a. Substitution yields $a + 2/a = c$. The optimal $c = \sqrt{8}$ is achieved with $a = \sqrt{2}$.

Translating this back, let us make the x-th backup at time $x^2/2$. Is is not hard to verify accurately that this strategy has excess rate bounded by $\sqrt{8}\sqrt{f}$: Let the fault appear at time u^2, with $x^2/2 < u^2 \le (x + 1)^2/2$. The coefficient of $\sqrt{f}$ at this moment is obviously

$$c = \frac{x + u^2 - x^2/2}{u}.$$

This term is monotone increasing in u within this interval, so we may consider $u^2 = (x + 1)^2/2$, implying

$$c = \sqrt{2}\frac{2x + 1/2}{x + 1} < 2\sqrt{2}.$$

□

Note that optimality holds only in an asymptotic sense, i.e. for $f \to 0$. The coefficient is $\sqrt{8}$ minus some term vanishing with f. It might be interesting to analyze this lower-order term, too.

Next we extend the idea to n faults. Here the coefficient improves upon the 1-fault optimum, if we combine n single-fault phases appropriately: Note that the inequality used in Proposition 2 is tight for equal-length phases only, so it should be possible to beat $\sqrt{8}$ by adapting the backup frequency. A more intuitive explanation of this effect is that the online player learns, with each fault, more about the parameter f which describes an average behaviour in time.

Lemma 1. *Any deterministic n-fault strategy with excess rate $c_n\sqrt{f}$ yields a deterministic $(n + 1)$-fault strategy with excess rate*

$$c_{n+1} = \frac{c_n^2 n + 2}{c_n\sqrt{n^2 + n}}\sqrt{f}.$$

Proof. Apply the given n-fault strategy up to the n-th fault which occurs, say, at time z^2. With $c := c_n$, the cost of backups and lost time until the n-th fault is $c\sqrt{f}z^2 = c\sqrt{n}z$. Let $y(x)$ denote the number of further backups until time $(z + x)^2$. Assuming that the $(n + 1)$-th fault appears at time $(z + x)^2$ and allowing for differentiable real functions y, the total cost up to this moment is bounded by

$$c\sqrt{n}z + y(x) + 2(z + x)/y'(x).$$

(The arguments are the same as in Theorem 1.) On the other hand, with $C := c_{n+1}$, the cost up to $(z + x)^2$ is

$$C\sqrt{n + 1}(z + x).$$

Together this yields the differential equation

$$c\sqrt{n}z + y(x) + 2(z + x)/y'(x) = C\sqrt{n + 1}(z + x)$$

with $y(0) = 0$. One solution is given by $y(x) = c\sqrt{n}x$ and C as claimed.

Once we have derived this solution heuristically, we can verify it exactly:

Using the backup function $y(x) = c\sqrt{n}x$ means to make the k-th backup after z^2 at time $(z + \frac{k}{c\sqrt{n}})^2$. Let the next fault appear at time $(z + u)^2$, with $(z + \frac{k}{c\sqrt{n}})^2 < (z + u)^2 \le (z + \frac{k+1}{c\sqrt{n}})^2$. Then we have

$$C = \frac{c\sqrt{n}z + k + (z + u)^2 - (z + \frac{k}{c\sqrt{n}})^2}{\sqrt{n + 1}(z + u)}.$$

Considering the derivative $\frac{dC}{du}$ we find that $C(u)$ can attain its maximum only at the endpoints of the interval. In case $u = \frac{k}{c\sqrt{n}}$ we get

$$C = c\sqrt{\frac{n}{n + 1}} < \frac{c^2n + 2}{c\sqrt{n^2 + n}}.$$

So it suffices to consider $u = \frac{k+1}{c\sqrt{n}}$. Obvious algebraic manipulation yields, in a few steps

$$C = \frac{c^2n + 2 + kc\sqrt{n}/z + (2k + 1)/(c\sqrt{n}z)}{c\sqrt{n^2 + n} + (k + 1)\sqrt{n + 1}/z}.$$

In the numerator, replace k with $k + 1$ and $2k + 1$ with $2k + 2$. Then we see

$$C < \frac{c^2n + 2}{c\sqrt{n^2 + n}}$$

also in this case. □

Note that the excess rate at any time after the n-th fault is smaller than $\sqrt{\frac{n+1}{n}}c_n\sqrt{f}$. For $n \to \infty$ we get:

Theorem 2. *There exists a deterministic backup strategy with excess rate $c_n\sqrt{f}$ after n faults, such that* $\lim c_n = 2$.

Proof. Consider the sequence c_n given by Lemma 1. With $s_n := c_n^2$ we get

$$s_{n+1} = \frac{(s_n n + 2)^2}{s_n(n^2 + n)}.$$

Further let be $s_n = 4 + r_n/n$. By easy manipulation we obtain

$$r_{n+1} = r_n + \frac{4}{4n + r_n}.$$

Thus $r_n = O(\ln n)$, $\lim s_n = 4$, and $\lim c_n = 2$, independently of the start value. □

4 A Randomized Backup Strategy

As e.g. in the rent-to-buy problem [4], suitable randomization significantly improves the expected cost against an oblivious adversary (who has no insight into the online player's random decisions).

Theorem 3. *There exists a randomized* 1*-fault strategy with expected excess rate* $c\sqrt{f}$ *such that* $\lim_{f\to 0} c = 2$.

Proof. (Sketch) We modify the deterministic strategy of Theorem 1 which had coefficient $c = 2\sqrt{2}$. The x-th backup is made at time $(x + r)^2/2$, where r is a fixed number, randomly chosen from interval $[0, 1]$. This randomized strategy makes the same number of backups as our deterministic strategy did, but it is quite clear that the expected loss of working time is about half the worst-case loss incurred by S (subject to some failure vanishing with f, i.e. with growing backup number). Furthermore remember that, in Theorem 1, both the backups and the worst-case loss of time contributed the same amount $a = \sqrt{2}$ to c. We conclude that our randomized strategy is only $3/4$ times as expensive as S, which gives $\lim c = 3/\sqrt{2}$.

We achieve the slightly better factor 2 if we make the x-th backup at time $(x + r)^2$ instead! Namely, this reduces the backup cost and the expected loss by the same factor $\sqrt{2}$. □

For this type of randomized backup strategy (a fixed backup pattern randomly shifted on the quadratic scale), the above result is optimal, by a similar argument as in Theorem 1. It remains open whether it is optimal at all. We hope that a suitable application of Yao's minimax principle will provide an answer. For n faults we have:

Theorem 4. *There exists a randomized backup strategy with expected excess rate* $c_n\sqrt{f}$ *after* n *faults, such that* $\lim_{f\to 0}\lim_{n\to\infty} c_n = \sqrt{2}$.

Proof. (Sketch) The method of Lemma 1 of extending an n-fault strategy to an $(n + 1)$-fault strategy is also applicable in the randomized case, i.e. if c is the expected coefficient: If we apply a scheme as in the weaker $3/\sqrt{2}$ version of Theorem 3, the number of backups $y(x)$ is deterministic (subject to ± 1 deviations), and $2(z + x)/y'(x)$ is replaced with the expected loss, i.e. multiplied by $1/2$. We therefore use the modified equation

$$c\sqrt{n}z + y(x) + (z + x)/y'(x) = C\sqrt{n + 1}(z + x)$$

to obtain C which is the expected c_{n+1}. One solution is given by $y(x) = c\sqrt{n}x$, for

$$C = \frac{c^2 n + 2}{c\sqrt{n^2 + n}}.$$

Let $s_n = c_n^2$. We get $\lim s_n = 2$ in a similar way as in Theorem 2. The straightforward calculations are omitted. □

5 Some Lower Bounds

The quite trivial Proposition 1 remains true for randomized strategies and an adaptive adversary. A stronger lower bound can be shown if the adversary has no obligation to meet some prescribed f.

Proposition 3. *No backup strategy can guarantee an excess rate below* $2\sqrt{f} - f$ *against an adaptive adversary.*

Proof. The adversary partitions the time axis into phases of some fixed length $t > 2$ and behaves as follows. If the online player did not make any backup in a phase then the adversary injects a fault at the end of this phase. Let x be the fraction of phases without backup, thus ending with a fault. Here the online player pays 1 per time for repeated work. In the remaining $1 - x$ fraction of phases he pays $1/t$ or more per time for backups. Hence the average cost per time is at least $x + (1 - x)/t$. Furthermore note that $f = x/t$. Thus the coefficient of $\sqrt{f}$ is

$$c = (x + \frac{1}{t} - \frac{x}{t})\frac{\sqrt{t}}{\sqrt{x}} = \sqrt{tx} + \frac{1}{\sqrt{tx}} - \frac{\sqrt{x}}{\sqrt{t}}.$$

The online player can minimize c choosing any strategy with $x = 1/(t - 1)$ which yields

$$c = \frac{\sqrt{t}}{\sqrt{t-1}} + \frac{\sqrt{t-1}}{\sqrt{t}} - \frac{1}{\sqrt{t(t-1)}}$$

and also means $f = 1/t(t - 1)$. Now the assertion follows easily.

Note that the coefficient can be made arbitrarily close to 2 with large enough t. □

This lower bound does not contradict Theorem 4 which refers to an oblivious adversary who must fix the fault times beforehand, whereas in Proposition 3, the adversary can permanently decide whether to inject a fault or not, depending on the online players behaviour. Thus he can also gain some information about the coin tosses in a randomized online strategy. (Of course, the oblivious adversary better reflects the real-world situation.)

In the deterministic case the adversaries all have the same power, hence it follows:

Corollary 1. *No deterministic backup strategy can guarantee an excess rate better than* $2\sqrt{f} - f$. □

In view of Theorem 2 this is a matching asymptotic lower bound.

For deterministic strategies, a stronger lower bound than in Proposition 1 can be proven also for prescribed f. We state one such result:

Proposition 4. *For any fault frequency* f*, even if the online player knows* f *beforehand, no deterministic backup strategy can guarantee an excess rate below* $\sqrt{2}\sqrt{f}$.

Proof. An adversary partitions the time axis into phases of length $1/f$ and places one fault in each phase by the following rule: Since the online player's strategy is deterministic, the adversary knows the sequence of backups made in the next phase until a fault. W.l.o.g. the phase starts at time 0, and the backup times are $t_1, \ldots, t_k$. Let $t_0 = 0$, and in case $t_k < 1/f$ further define $t_{k+1} = 1/f$. The adversary injects a fault immediately before t_{i+1} such that $i + t_{i+1} - t_i$ is maximized.

The best an online player can do against this adversary's strategy is to choose his t_i so as to minimize $\max_i(i + t_{i+1} - t_i)$. Obviously all these terms should be equal, thus $t_i = it_1 - i(i-1)/2$. In particular this yields $1/f \approx kt_1 - k^2/2$. Since the adversary may place his fault at the end of the phase, both t_1 and k are lower bounds for the additional cost the online player incurs in this phase. By elementary calculation, $\max\{t_1, k\}$ is minimized if $t_1 = k = \sqrt{2/f}$. □

References

1. A. Borodin, R. El-Yaniv: *Online Computation and Competitive Analysis*, Cambridge Univ. Press 1998
2. P. Damaschke: Multiple spin-block decisions, *10th ISAAC'99*,
3. D.R. Dooly, S.A. Goldman, S.D. Scott: TCP dynamic acknowledgment delay: theory and practice, *30th STOC'98*, 389-398
4. A.R. Karlin, M.S. Manasse, L.A. McGeoch, S. Owicki: Competitive randomized algorithms for non-uniform problems, *Algorithmica* 11 (1994), 542-571
5. P. Krishnan, P.M. Long, J.S. Vitter: Adaptive disk spindown via optimal rent-to-buy in probabilistic environments, *Algorithmica* 23 (1999), 31-56
6. R. Motwani, S. Phillips, E. Torng: Nonclairvoyant scheduling, *Theor. Comp. Sc.* 130 (1994), 17-47; extended abstact in: *4th SODA'93*, 422-431

Towards the Notion of Stability of Approximation for Hard Optimization Tasks and the Traveling Salesman Problem

(Extended Abstract)⋆

Hans-Joachim Böckenhauer, Juraj Hromkovic, Ralf Klasing, Sebastian Seibert, and Walter Unger

Dept. of Computer Science I (Algorithms and Complexity), RWTH Aachen,
Ahornstra e 55, 52056 Aachen, Germany
fhjb,jh,rak,seibert,quaxg@i1.informatik.rwth-aachen.de

Abstract. The investigation of the possibility to e ciently compute approximations of hard optimization problems is one of the central and most fruitful areas of current algorithm and complexity theory. The aim of this paper is twofold. First, we introduce the notion of stability of approximation algorithms. This notion is shown to be of practical as well as of theoretical importance, especially for the real understanding of the applicability of approximation algorithms and for the determination of the border between easy instances and hard instances of optimization problems that do not admit any polynomial-time approximation.
Secondly, we apply our concept to the study of the traveling salesman problem. We show how to modify the Christo des algorithm for -TSP to obtain e cient approximation algorithms with constant approximation ratio for every instance of TSP that violates the triangle inequality by a multiplicative constant factor. This improves the result of Andreae and Bandelt [AB95].

Keywords: Stability of approximation, Traveling Salesman Problem

1 Introduction

Immediately after introducing NP-hardness (completeness) [Co71] as a concept for proving intractability of computing problems, the following question has been posed: If an optimization problem does not admit an e ciently computable optimal solution, is there a possibility to e ciently compute at least an approximation of the optimal solution? Several researchers [Jo74], [Lo75], [Chr76], [IK75] provided already in the middle of the seventies a positive answer for some optimization problems. It is a fascinating e ect if one can jump from exponential

? Supported by DFG Project HR 14/5-1 \Zur Klassi zierung der Klasse praktisch lösbarer algorithmischer Aufgaben" complexity (a huge inevitable amount of physical work) to polynomial complexity (tractable amount of physical work) due to a small change in the requirements | instead of an exact optimal solution one demands a solution whose cost di ers from the cost of an optimal solution by at most ″% of the cost of an optimal solution for some ″ > 0. This e ect is very strong, especially, if one considers problems for which this approximation concept works for any relative di erence ″ (see the concept of approximation schemes in [IK75], [MPS98], [Pa94], [BC93]). This is also the reason why currently optimization problems are considered to be tractable if there exist randomized polynomial-time approximation algorithms that solve them with a reasonable approximation ratio. In what follows an -approximation algorithm for a minimization [maximization] problem is any algorithm that provides feasible solutions whose cost divided by the cost of optimal solutions is at most [is at least $\frac{1}{\ }$].

There is also another possibility to jump from NP to P. Namely, to consider the subset of inputs with a special, nice property instead of the whole set of inputs for which the problem is well-de ned. A nice example is the Traveling Salesman Problem (TSP). TSP is not only NP-hard, but also the search for an approximate solution for TSP is NP-hard for every constant approximation ratio.[1] But if one considers TSP for inputs satisfying the triangle inequality (so called -TSP), one can even design a polynomial-time $\frac{3}{2}$-approximation algorithm [Chr76].[2] The situation is even more interesting if one considers the Euclidean TSP, where the distances between the nodes correspond to the distances in the Euclidean metrics. The Euclidean TSP is NP-hard [Pa77], but for every > 1 one can design a polynomial-time -approximation algorithm [Ar98], [Mi96]. Moreover, if one allows randomization the resulting approximation algorithm works in n $(\log_2 n)^{O(1)}$ time [Ar97].[3] This is the reason why we propose again to revise the notion of tractability especially because of the standard de nition of complexity as the worst-case complexity: Our aim is to try to separate the easy instances from the hard instances of every computing problem considered to be intractable. In fact, by our concept, we want to attack the de nition of complexity as the worst-case complexity. The approximation ratio of an algorithm is also de ned in a worst-case manner. Our idea is to split the set of input instances of the given problem into possibly in nitely many subclasses according to the hardness of their approximability, and to have an e cient algorithm for deciding the membership of any problem instance to one of the subclasses considered. To achieve this goal we introduce the concept of approximation stability.

Informally, one can describe the idea of our concept by the following scenario. One has an optimization problem for two sets of inputs L_1 and L_2, $L_1 \subsetneq L_2$. For

[1] Even no $f(n)$-approximation algorithm exists for f polynomial in the input size n.

[2] Note that -TSP is APX-hard and we know even explicit lower bounds on its inapproximability [En99, BHKSU00].

[3] Obviously, there are many similar examples where with restricting the set of inputs one crosses the border between decidability and undecidability (Post Correspondence Problem) or the border between P and NP (SAT and 2-SAT, or vertex cover problem).

L_1 there exists a polynomial-time -approximation algorithm A for some > 1, but for L_2 there is no polynomial-time γ-approximation algorithm for any $\gamma > 1$ (if NP is not equal to P). We pose the following question: Is the use of algorithm A really restricted to inputs from L_1? Let us consider a distance measure d in L_2 determining the distance $d(x)$ between L_1 and any given input $x\ 2\ L_2 - L_1$. Now, one can consider an input $x\ 2\ L_2 - L_1$ with $d(x) \leqslant k$ for some positive real k. One can look for how \good" the algorithm A is for the input $x\ 2\ L_2 - L_1$. If for every $k > 0$ and every x with $d(x) \leqslant k$, A computes a $\gamma_{k;}$ -approximation of an optimal solution for x ($\gamma_{k;}$ is considered to be a constant depending on k and only), then one can say that A is \(approximation) stable" according to the distance measure d. Obviously, such a concept enables to show positive results extending the applicability of known approximation algorithms. On the other hand it can help to show the boundaries of the use of approximation algorithms and possibly even a new kind of hardness of optimization problems.

Observe that the idea of the concept of approximation stability is similar to that of stability of numerical algorithms. Instead of observing the size of the change of the output value according to a small change of the input value, one looks for the size of the change of the approximation ratio according to a small change in the speci cation of the set of consistent input instances.

To demonstrate the applicability of our new approach we consider TSP, -TSP, and, for every real > 1, -TSP containing all input instances with $cost(u;v) \leqslant \ (cost(u;x) + cost(x;v))$ for all vertices $u;v;x$. If an input is consistent for -TSP we say that its distance to -TSP is at most $- 1$. We will show that known approximation algorithms for -TSP are unstable according to this distance measure. But we will nd a way how to modify the Christo des algorithm in order to obtain approximation algorithms for -TSP that are stable according to this distance measure. So, this e ort results in a $(\frac{3}{2}\ ^2)$-approximation algorithm for -TSP.[4] This improves the result of Andreae and Bandelt [AB95] who presented a $(\frac{3}{2}\ ^2 + \frac{1}{2}\ )$-approximation algorithm for -TSP. Our approach essentially di ers from that of [AB95], because in order to design our $(\frac{3}{2}\ ^2)$-approximation algorithm we modify the Christo des algorithm while Andreae and Bandelt obtain their approximation ratio by modifying the original 2-approximation algorithm for -TSP.

Note that, after this paper was written, we got the information about the independent, unpublished result of Bender and Chekuri, accepted for WADS'99 [BC99]. They designed a 4 -approximation algorithm which can be seen as a modi cation of the 2-approximation algorithm for -TSP. Despite this nice result, there are three reasons to consider our algorithm. First, our algorithm provides a better approximation ratio for $< \frac{8}{3}$. Secondly, in the previous work [AB95], the authors claim that the Christo des algorithm cannot be modi ed

[4] Note that in this way we obtain an approximate solution to every problem instance of TSP, where the approximation ratio depends on the distance of this problem instance to -TSP. Following the discussion in [Ar98] about typical properties of real problem instances of TSP our approximation algorithm working in $O(n^3)$ time is of practical relevance.

in order to get a stable (in our terminology) algorithm for TSP, and our result disproves this conjecture. This is especially of practical importance, since for instances where the triangle inequality is violated only by a few edge costs, one can expect that the approximation ratio will be as in the underlying algorithm with a high probability. Finally, our algorithm is a practical $O(n^3)$-algorithm. This cannot be said about the 4β-approximation algorithm from [BC99]. The first part of the latter algorithm is a 2-approximation algorithm for finding minimal two-connected subgraphs with time complexity $O(n^4)$. For the second part, constructing a Hamiltonian tour in S^2 (if S was the two-connected subgraph), there exist only proofs saying that it can be implemented in polynomial time, but no low-degree polynomial upper bound on the time complexity of these procedures has been established.

This paper is organized as follows: In Section 2 we introduce our concept of approximation stability. In Section 3 we show how to apply our concept in the study of the TSP, and in Section 4 we discuss the potential applicability and usefulness of our concept.

2 Definition of the Stability of Approximation Algorithms

We assume that the reader is familiar with the basic concepts and notions of algorithmics and complexity theory as presented in standard textbooks like [BC93], [CLR90], [GJ79], [Ho96], [Pa94]. Next, we give a new definition of the notion of an optimization problem. The reason to do this is to obtain the possibility to study the influence of the input sets on the hardness of the problem considered. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of nonnegative integers, let $\mathbb{R}^+$ be the set of positive reals, and let $\mathbb{R}^{\geq a}$ be the set of all reals greater than or equal to a for some $a \in \mathbb{R}$.

Definition 1. *An* **optimization problem** U *is a 7-tuple* $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$, *where*

1. Σ_I *is an alphabet called* **input alphabet**,
2. Σ_O *is an alphabet called* **output alphabet**,
3. $L \subseteq \Sigma_I^*$ *is a language over* Σ_I *called the* **language of consistent inputs**,
4. $L_I \subseteq L$ *is a language over* Σ_I *called the* **language of actual inputs**,
5. M *is a function from* L *to* $2^{\Sigma_O^*}$, *where, for every* $x \in L$, $M(x)$ *is called the* **set of feasible solutions** *for the input* x,
6. $cost$ *is a function, called* **cost function**, *from* $\bigcup_{x \in L} M(x) \times L_I$ *to* $\mathbb{R}^{\geq 0}$,
7. $goal \in \{minimum, maximum\}$.

For every $x \in L$, *we define*

$$Output_U(x) = \{y \in M(x) \mid cost(y, x) = goal\{cost(z, x) \mid z \in M(x)\}\}$$

and

$$Opt_U(x) = cost(y, x) \quad \text{for some } y \in Output_U(x).$$

Clearly, the meaning for Σ_I, Σ_O, M, $cost$ and $goal$ is the usual one. L may be considered as the set of consistent inputs, i.e., the inputs for which the optimization problem is consistently defined. L_I is the set of inputs considered and only these inputs are taken into account when one determines the complexity of the optimization problem U. This kind of definition is useful for considering the complexity of optimization problems parameterized according to their languages of actual inputs. In what follows, $\mathit{Language}(U)$ denotes the language L_I of actual inputs of U. If the input x is fixed, we usually use $cost(y)$ instead of $cost(y, x)$ in what follows.

Definition 2. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ be an optimization problem. We say that an algorithm A is a* **consistent algorithm for U** *if, for every input $x \in L_I$, A computes an output $A(x) \in M(x)$. We say that A* **solves** *U if, for every $x \in L_I$, A computes an output $A(x)$ from $Output_U(x)$. The time complexity of A is defined as the function*

$$\mathit{Time}_A(n) = \max\{\mathit{Time}_A(x) \mid x \in L_I \cap \Sigma_I^n\}$$

from $\mathbb{N}$ to $\mathbb{N}$, where $\mathit{Time}_A(x)$ is the length of the computation of A on x.

Next, we give the definitions of standard notions in the area of approximation algorithms (see e.g. [CK98], [Ho96]).

Definition 3. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ be an optimization problem, and let A be a consistent algorithm for U. For every $x \in L_I$, the* **approximation ratio $R_A(x)$** *is defined as*

$$R_A(x) = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

For any $n \in \mathbb{N}$, we define the **approximation ratio of A** *as*

$$R_A(n) = \max\{R_A(x) \mid x \in L_I \cap \Sigma_I^n\}.$$

For any positive real $\delta > 1$, we say that A is a **δ-approximation algorithm for U** *if $R_A(x) \leqslant \delta$ for every $x \in L_I$.*

For every function $f : \mathbb{N} \to \mathbb{R}^{>1}$, we say that A is an **$f(n)$-approximation algorithm for U** *if $R_A(n) \leqslant f(n)$ for every $n \in \mathbb{N}$.*

In what follows, we consider the standard definitions of the classes NPO, PO, APX (see e.g. [Ho96],[MPS98]). In order to define the notion of stability of approximation algorithms we need to consider something like a distance between a language L and a word outside L.

Definition 4. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ and $\overline{U} = (\Sigma_I, \Sigma_O, L, L, M, cost, goal)$ be two optimization problems with $L_I \subsetneq L$. A* **distance function for U according to L_I** *is any function $h_L : L \to \mathbb{R}^{\geqslant 0}$ satisfying the properties*

1. $h_L(x) = 0$ *for every* $x\ 2\ L_I$, *and*
2. h_L *can be computed in polynomial time.*

Let h *be a distance function for* U *according to* L_I. *We de ne, for any* $r\ 2\ \mathrm{IR}^+$,

$$Ball_{r,h}(L_I) = fw\ 2\ L\,j\,h(w) \leqslant rg:$$

Let A *be a consistent algorithm for* $\overline{U}$, *and let* A *be an* $''$*-approximation algorithm for* U *for some* $''\ 2\ \mathrm{IR}^{>1}$. *Let* p *be a positive real. We say that* A *is* **p-stable according to h** *if, for every real* $0 < r \leqslant p$, *there exists a* $\ _{r;''}\ 2\ \mathrm{IR}^{>1}$ *such that* A *is a* $\ _{r;''}$*-approximation algorithm for* $U_r = (\ _I,\ \ _O, L, Ball_{r;h}(L_I), M, cost, goal)$.[5]

A *is* **stable according to h** *if* A *is* p*-stable according to* h *for every* $p\ 2\ \mathrm{IR}^+$. *We say that* A *is* **unstable according to h** *if* A *is not* p*-stable for any* $p\ 2\ \mathrm{IR}^+$.

For every positive integer r, *and every function* $f_r : \mathrm{IN}\ !\ \mathrm{IR}^{>1}$ *we say that* A *is* **$(r; f_r(n))$-quasistable according to h** *if* A *is an* $f_r(n)$*-approximation algorithm for* $U_r = (\ _I;\ \ _O, L, Ball_{r;h}(L_I), M, cost, goal)$.

A discussion about the potential usefulness of our concept is given in the last section. In the next section we show a transparent application of our concept for TSP.

3 Stability of Approximation Algorithms and TSP

We consider the well-known TSP problem (see e.g. [LLRS85]) that is in its general form very hard for approximation. But if one considers complete graphs in which the triangle inequality holds, then we have a $\frac{3}{2}$-approximation algorithm due to Christo des [Chr76]. So, this is a suitable starting point for the application of our approach based on approximation stability. First, we de ne two natural distance measures and show that the Christo des algorithm is stable according to one of them, but not according to the second one. This leads to the development of a new algorithm, PMCA, for -TSP. This algorithm is achieved by modifying Christo des algorithm in such a way that the resulting algorithm is stable according to the second distance measure, too. In this way, we obtain a $(\frac{3}{2}\ \ (1 + r)^2)$-approximation algorithm for every input instance of TSP with the distance at most r from $Language($ -TSP$)$, i.e. with $cost(u; v) \leqslant (1 + r)\ \ (cost(u; w) + cost(w; v))$ for every three nodes $u; v; w$. This improves the result of Andreae and Bandelt [AB95] who achieved approximation ratio $\frac{3}{2}(1 + r)^2 + \frac{1}{2}(1 + r)$.

To start our investigation, we concisely review two well-known algorithms for -TSP: the 2-approximative algorithm 2APPR and the $\frac{3}{2}$-approximative Christo des algorithm [Chr76], [Ho96].

[5] Note that $\ _{r;''}$ is a constant depending on r and $''$ only.

Algorithm 2APPR

Input: A complete graph $G = (V, E)$ with a cost function $cost : E \to \mathbb{R}^{\geqslant 0}$ satisfying the triangle inequality (for every $u, v, q \in V$, $cost(u, v) \leqslant cost(u, q) + cost(q, v)$).

Step 1a: Construct a minimal spanning tree T of G. (The cost of T is surely smaller than the cost of the optimal Hamiltonian tour.)

Step 1b: Construct an Eulerian tour D on T going twice via every edge of T. (The cost of D is exactly twice the cost of T.)

Step 2: Construct a Hamiltonian tour H from D by avoiding the repetition of nodes in the Eulerian tour. (In fact, H is the permutation of nodes of G, where the order of a node v is given by the first occurrence of v in D.)

Output: H.

Christofides Algorithm

Input: A complete graph $G = (V, E)$ with a cost function $cost : E \to \mathbb{R}^{\geqslant 0}$ satisfying the triangle inequality.

Step 1a: Construct a minimal spanning tree T of G and find a matching M with minimal cost (at most $\frac{1}{2}$ of the cost of the optimal Hamiltonian tour) on the nodes of T with odd degree.

Step 1b: Construct a Eulerian tour D on $G' = T \cup M$.

Step 2: Construct a Hamiltonian tour H from D by avoiding the repetition of nodes in the Eulerian tour.

Output: H.

Since the triangle inequality holds and Step 2 in both algorithms is realized by repeatedly shortening a path $x, u_1, \ldots, u_m, y$ by the edge (x, y) (because $u_1, \ldots, u_m$ have already occurred before in the prefix of D) the cost of H is at most the cost of D. Thus, the crucial point for the success of 2APPR and Christofides algorithm is the triangle inequality. A reasonable possibility to search for an extension of the application of these algorithms is to look for inputs that "almost" satisfy the triangle inequality. In what follows we do this in two different ways.

Let Δ-TSP $= (\Sigma_I, \Sigma_O, L, L_I, M, cost, minimum)$ be a representation of the TSP with triangle inequality. We may assume $\Sigma_I = \Sigma_O = \{0, 1, \#\}$, L contains codes of all cost functions for edges of complete graphs, and L_I contains codes of cost functions that satisfy the triangle inequality. Let, for every $x \in L$, $G_x = (V_x, E_x, cost_x)$ be the complete weighted graph coded by x. Obviously, the above algorithms are consistent for $(\Sigma_I, \Sigma_O, L, L, M, cost, minimum)$.

Let $\Delta_{1+r,d}$-TSP $= (\Sigma_I, \Sigma_O, L, Ball_{r,d}(L_I), M, cost, minimum)$ for any $r \in \mathbb{R}^+$ and for any distance function d for Δ-TSP. We define for every $x \in L$,

$$dist(x) = \max \left\{ 0, \max \left\{ \frac{cost(u,v)}{cost(u,p) + cost(p,v)} - 1 \,\middle|\, u, v, p \in V_x \right\} \right\},$$

and

$$distance(x) = \max \left\{ 0, \max \left\{ \frac{cost(u,v)}{\sum_{i=1}^{m} cost(p_i, p_{i+1})} - 1 \,\middle|\, u, v \in V_x, \right.\right.$$
$$\text{and } u = p_1, p_2, \ldots, p_{m+1} = v \text{ is a simple path between}$$
$$\left.\left. u \text{ and } v \text{ in } G_x \right\} \right\}.$$

Since the distance measure *dist* is the most important for us we will use the notation -TSP instead of $_{,dist}$-TSP. For simplicity we consider the size of x as the number of nodes of G_x instead of $|x|$. We observe that for every $\geqslant 1$ the inputs from $_{,dist}$-TSP have the property $cost(u,v) \leqslant (cost(u,x) + cost(x,v))$ for all u, v, x ($= 1 + r$). It is a simple exercise to prove the following lemma.

Lemma 1. *The 2APPR and Christofides algorithm are stable according to distance.* □

Now, one can ask for the approximation stability according to the distance measure *dist* that is the most interesting distance measure for us. Unfortunately, as shown in the next lemmas, the answer is not as positive as for *distance*.

Lemma 2. *For every $r \in \mathbb{R}^+$, Christofides algorithm is $\left(r, \frac{3}{2} \cdot (1+r)^{2\lceil \log_2 n \rceil}\right)$-quasistable for dist, and 2APPR is $(r, 2 \cdot (1+r)^{\lceil \log_2 n \rceil})$-quasistable for dist.* □

That the result of Lemma 2 cannot be essentially improved, is shown by presenting an input for which the Christofides algorithm as well as 2APPR provide a very poor approximation.

Lemma 3. *For every $r \in \mathbb{R}^+$, if the Christofides algorithm (or 2APPR) is $(r, f_r(n))$-quasistable for dist, then $f_r(n) \geqslant n^{\log_2(1+r)}/(2 \cdot (1+r))$.*

Proof. We construct a weighted complete graph from $Ball_{r,dist}(L_I)$ as follows. We start with the path $p_0, p_1, \ldots, p_n$ for $n = 2^k$, $k \in \mathbb{N}$, where every edge (p_i, p_{i+1}) has the cost 1. For all other edges we take maximal possible costs in such a way that the constructed input is in $Ball_{r,dist}(L_I)$. As a consequence, for every $m \in \{1, \ldots, \log_2 n\}$, we have $cost(p_i, p_{i+2^m}) = 2^m \cdot (1+r)^m$ for $i = 0, \ldots, n - 2^m$ (see Figure 1).

Let us have a look at the work of Christofides algorithm on this input. (Similar considerations can be made for 2APPR.) There is only one minimal spanning tree that corresponds to the path containing all edges of the cost 1. Since every path contains exactly two nodes of odd degree, the Eulerian graph constructed in Step 1 is the cycle $D = p_0, p_1, p_2, \ldots, p_n, p_0$ with the n edges of cost 1 and the edge of the maximal cost $n \cdot (1+r)^{\log_2 n} = n^{1+\log_2(1+r)}$. Since the Eulerian path is a Hamiltonian tour, the output of the Christofides algorithm is unambiguously

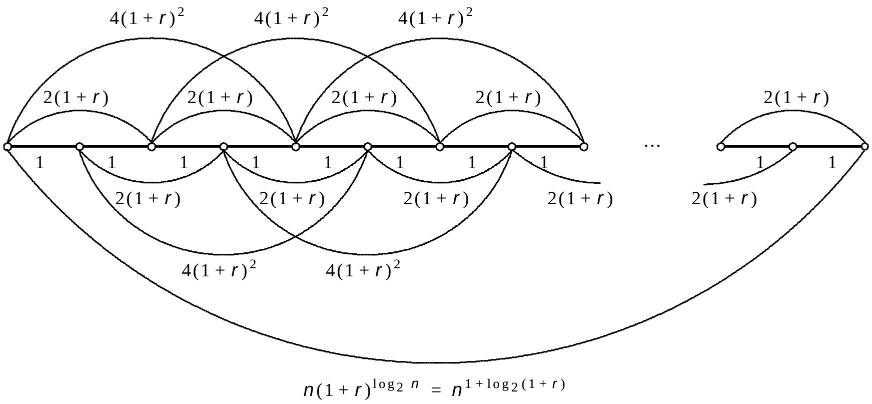

Fig. 1. A hard $_{;dist}$-TSP instance

the cycle $p_0; p_1; :::; p_n; p_0$ with the cost $n + n \ (1 + r)^{\log_2 n}$. But the optimal tour is

$$T = p_0; p_2; p_4; :::; p_{2i}; p_{2(i+1)}; :::; p_n; p_{n-1}; p_{n-3}; :::; p_{2i+1}; p_{2i-1}; :::; p_3; p_1; p_0:$$

This tour contains two edges $(p_0; p_1)$ and $(p_{n-1}; p_n)$ of the cost 1 and all $n - 2$ edges of the cost $2 \ (1 + r)$. Thus, $Opt = cost(T) = 2 + 2 \ (1 + r) \ (n - 2)$, and

$$\frac{cost(D)}{cost(T)} = \frac{n + n^{1+\log_2(1+r)}}{2 + 2 \ (1 + r) \ (n - 2)} \geqslant \frac{n^{1+\log_2(1+r)}}{2 \ n \ (1 + r)} = \frac{n^{\log_2(1+r)}}{2 \ (1 + r)}:$$

ut

Corollary 1. *2APPR and the Christo des algorithm are unstable for dist.*

The results above show that 2APPR and Christo des algorithm can be useful for a much larger set of inputs than the original input set. But the stability according to *dist* would provide approximation algorithms for a substantially larger class of input instances. So the key question is whether one can modify the above algorithms to get algorithms that are stable according to *dist*. In what follows, we give a positive answer on this question.

Theorem 1. *For every $\ 2 \ \mathbb{R}^{\geqslant 1}$, there is a $(\frac{3}{2} \ ^2)$-approximation algorithm PMCA for $_{;dist}$-TSP working in time $O(n^3)$.*

Proof sketch.. In the following, we will give a sketch of the proof of Theorem 1 by stating algorithm PMCA. The central ideas of PMCA are the following. First, we replace the minimum matching generated in the Christo des Algorithm by a \minimum path matching". That means to nd a pairing of the given vertices s.t. the vertices in a pair are connected by a path rather than a single edge, and the goal is to minimize the sum of the path costs. In this way, we obtain an Eulerian tour on the multi-graph consisting of spanning tree and path matching (in general not Hamiltonian). This Eulerian tour has a cost of at most 1:5 times of the cost of an optimal TSP tour.

The second new concept concerns the substitution of sequences of edges by single ones, when transforming the above mentioned tour to a Hamiltonian one. Here, we can guarantee that at most four consecutive edges will be eventually substituted by a single one. This may increase the cost of the tour by a factor of at most ^{2} for inputs from -TSP (remember that we deal in what follows only with distance function *dist*, and therefore drop the corresponding subscript from $_{;dist}$-TSP).

Before stating the algorithm in detail, we have to introduce its main tools rst. Let $G = (V;E)$ be a graph. A *path matching* for a set of vertices U V of even size is a collection of $jUj{=}2$ edge-disjoint paths having U as the set of endpoints.

Assume that $p = (u_0;u_1);(u_1;u_2);:::;(u_{k-1};u_k)$ is a path in $(V;E)$, not necessarily simple. A *bypass* for p is an edge $(u;v)$ from E, replacing a sub-path $(u_i;u_{i+1});(u_{i+1};u_{i+2});:::;(u_{j-1};u_j)$ of p from $u = u_i$ to $u_j = v$ $(0 \leqslant i < j \leqslant k)$. Its *size* is the number of replaced edges, i.e. $j - i$.[6] Also, we say that the vertices $u_{i+1};u_{i+2};:::;u_{j-1}$ *are bypassed*. Given some set of simple paths , a *conflict* according to is a vertex which occurs at least two times in the given set of paths.

Algorithm PMCA

```
Input: a complete graph $(V;E)$ with cost function $cost : E \; ! \; \mathbb{R}^{\geqslant 0}$
   (a -TSP instance for 1).
1. Construct a minimal spanning tree $T$ of $(V;E)$.
2. Let $U$ be the set of vertices of odd degree in $T$;
   construct a minimal (edge-disjoint) path matching for $U$.
3. Resolve conflicts according to , in order to
      obtain a vertex-disjoint path matching $^0$ with $cost(\;^0) \leqslant \; cost(\;)$
      (using bypasses of size 2 only).
4. Construct an Eulerian tour on $T$ and $^0$.
      ( can be considered as a sequence of paths $p_1;p_2;p_3;:::$
      such that $p_1;p_3;:::$ are paths in $T$, and $p_2;p_4;::: 2 \;^0$)
5. Resolve conflicts inside the paths $p_1;p_3;:::$ from $T$, such that $T$ is divided into
      a forest $T_f$ of trees of degree at most 3, using bypasses of size 2 only.
     (Call the resulting paths $p_1^0;p_3^0;:::$ and the modi ed tour $^0$ is $p_1^0;p_2;p_3^0;p_4:::$.)
6. Resolve every double occurrence of nodes in $^0$ such that the overall size
     of the bypasses is at most 4 (where ''overall'' means that a bypass constructed
      in Step 3 or 5 counts for two edges). Obtain tour $^{00}$.
Output: Tour $^{00}$.
```

In the following, we have to explain how to e ciently obtain a minimal path matching, and how to realize the conflict resolution in Steps 3, 5, and 6. The latter not only have to be e cient but must also result in substituting at most four edges by a single one after all.

[6] Obviously, we are not interested in bypasses of size 1.

How to construct an Eulerian cycle in Step 4 is a well-studied task. We only observe that since each vertex can be endpoint of at most one path from ^{0} by de nition the same holds for T: the endpoints of $p_1; p_3; : : :$ are the same as those of $p_2; p_4; : : :$.

We give in the following detailed descriptions of Steps 2, 3, 5, and 6, respectively.

Claim 1
One can construct in time $O(jV j^3)$ a minimum path matching for U that has the following properties:

Every two paths in are edge-disjoint. (1)

forms a forest. (2)

Proof sketch.. First, we will show how to construct a path matching within the given time. To construct the path matching, we rst compute all-pairs cheapest paths.[7] Then, we de ne $G^0 = (V; E^0)$ where $cost^0(v; v^0)$ is the cost of a cheapest path between v and v^0 in G. Next, we compute a minimum matching on G^0 (in the usual sense), and nally, we substitute the edges of G^0 in the matching by the corresponding cheapest paths in G. Clearly, this can be done in time $O(n^3)$ and results in a minimum path matching.

The claimed properties (1) and (2) are a consequence of the minimality. The technical details will be given in the full version of this paper. ⊓⊔

The essential property of a minimal path matching for our purposes is that it costs at most half of the cost of a minimal Hamiltonian tour. Now we show how Step 3 of the algorithm is performed.

Claim 2
Every path matching having properties (1) and (2) can be modi ed into a vertex-disjoint one by using bypasses of size 2 only. Moreover, on each of the new paths, there will be at most one bypass.

Proof sketch.. By Claim 1, every vertex used by two paths in a path matching belongs to some tree. We will show how to resolve a tree of by using bypasses of size 2 in such a way that only vertices of the tree are a ected. Then we are done by solving all trees independently.

Let $_T$ be a subset of , forming a tree. For simplicity, we address $_T$ itself as a tree. Every vertex of $_T$ being a conflict has at least three edges incident to it, since it cannot be endpoint of two paths in , and it is part of at least two edge-disjoint paths by de nition of a conflict.

We reduce the size of the problem at hand in that we eliminate paths from the tree by resolving conflicts.

[7] Since we associate a cost instead of a length to the edges, we speak about cheapest instead of shortest path.

Procedure 1

```
Input: A minimal path matching   for some vertex set $U$ on $(V; E)$.
For all trees  $_T$ of
  While there are conflicts in  $_T$ (i.e. there is more than one path in  $_T$)
    pick an arbitrary path $p$ 2  $_T$;
    if $p$ has only one conflict $v$, and $v$ is an endpoint of $p$,
      pick another path using $v$ as new $p$ instead;
    let $v_1; v_2; :::; v_k$ be (in this order) the conflicts in $p$;
    while $k > 1$
     consider two paths $p_1; p_k$ 2  $_T$ which use $v_1$ respectively $v_k$, commonly with $p$;
      pick as new $p$ one of $p_1; p_k$ which was formerly not picked;
      let $v_1; v_2; :::; v_k$ be (in this order) the conflicts in $p$;
    let $v$ be the only vertex of the  nally chosen path $p$ which is a conflict;
    if $v$ has two incident edges in $p$,
      replace those with a bypass,
    else ($v$ is an endpoint of $p$)
      replace the single edge incident to $v$ in $p$ together
      with one of the previously picked paths with a bypass.
Output: the modi ed conflict-free path matching  $^0$.
```

The proof of the correctness of Procedure 1 is moved to the full version of this paper. □

Now we describe the implementation of Step 5 of Algorithm PMCA. It divides the minimal spanning tree by resolving conflicts into several trees, whose crucial property is that they have vertices of degree at most 3.

Procedure 2 below is based on the following idea. First, a root of T is picked. Then, we consider a path p_i in T which, under the orientation w.r.t. this root, will go up and down. The two edges immediately before and after the turning point are bypassed. One possible view on this procedure is that the minimal spanning tree is divided into several trees, since each bypass building divides a tree into two trees.

Procedure 2

```
Input: $T$ and the paths $p_1; p_3; p_5; :::$ computed in Step 4 of Algorithm PMCA.
Choose a node $r$ as a root in $T$.
For each path
$p_i = (v_1; v_2); (v_2; v_3); :::; (v_{n_i-1}; v_{n_i})$ in $T$ do
  Let $v_j$ be the node in $p_i$ of minimal distance to $r$ in $T$.
  If $1 < j < n_i$ then
    bypass the node $v_j$ and call this new path $p_i^0$.
  else $p_i^0 = p_i$.
Output: The paths $p_1^0; p_3^0; p_5^0; :::$, building a forest $T_f$.
```

Now the following properties hold. Their proofs are given in the full version of this paper.

1. If a node v occurs in two di erent paths p_i^0 and p_j^0 of T_f, then v is an inner node in one path and a terminal node in the other path. I.e. the node degree of the forest spanned by $p_1^0; p_3^0; p_5^0; :::$ is at most three.
2. In T_f, every path has at most one bypass, and every bypass is of size two.
3. Vertices which are leaves in T_f are not conflicts in ^{0}.
4. In the cycle $p_1^0; p_2; p_3^0; p_4; p_5^0; p_6; :::$, between each two bypasses there is at least one vertex not being a conflict.

Below, we present Procedure 3 which consecutively resolves the remaining conflicts. Note that $s; t; u; v$, and their primed versions, denote *occurrences* of vertices on a path, rather than the vertices itself. In one step, Procedure 3 has to make a choice.

Procedure 3

```
Input: a cycle $^0$ on $(V; E)$ where every vertex of $V$ occurs once or twice.
Take an arbitrary conflict, i.e. a vertex  occurring twice as $u$ and $u^0$ in $^0$;
bypass one occurrence, say $u$ (with a bypass of size 2);
while there are conflicts remaining
   if occurrence $u$ has at least one unresolved conflict as neighbor
      let $v$ be one of them, chosen by the following rule:
          If between $u$ and another bypassed vertex occurrence $t$ on
            $^0$, there are only unresolved conflicts, choose $v$ to be the
          neighbor of $u$ towards $t$.
      (($v; u$) or ($u; v$) is an edge of $^0$ and there is another occurrence $v^0$ of
         the same vertex as $v$)
      resolve that conflict by bypassing $v^0$
   else
      resolve an arbitrary conflict;
   let $u$ be the bypassed vertex.
Output: the modi ed cycle $^{00}$.
```

The proofs of correctness of Procedure 3 and of the approximation ratio of PMCA are given in the full version of this paper. ⧄

Theorem 1 improves the approximation ratio achieved in [AB95]. Note that this cannot be done by modifying the approach of Andreae and Bandelt. The crucial point of our improvement is based on the presented modi cation of Christo des algorithm while Andreae and Bandelt conjectured in [AB95] that Christo des algorithm cannot be modi ed in order to get an approximation algorithm for $_{;dist}$-TSP.

Note that Theorem 1 can also be formulated in a general form by substituting the parameter by a function (n), where n is the number of nodes of the graph considered.

4 Conclusion and Discussion

In the previous sections we have introduced the concept of stability of approximations and we have applied it to TSP. Here we discuss the potential applicability and usefulness of this concept. Applying it, one can establish positive results of the following types:

1. An approximation algorithm or a PTAS can be successfully used for a larger set of inputs than the set usually considered (see Lemma 1).
2. We are not able to successfully apply a given approximation algorithm A (a PTAS) for additional inputs, but one can modify A to get a new approximation algorithm (a new PTAS) working for a larger set of inputs than the set of inputs of A (see Theorem 1 and [AB95, BC99]).
3. To learn that an approximation algorithm is unstable for a distance measure could lead to the development of completely new approximation algorithms that would be stable according to the considered distance measure.

The following types of negative results may be achieved:

4. The fact that an approximation algorithm is unstable according to all \reasonable" distance measures and so that its use is really restricted to the original input set.
5. Let $Q = (\ _I, \ _O, L, L_I, M, cost, goal)$ 2 NPO be well approximable. If, for a distance measure d and a constant r, one proves the nonexistence of any approximation algorithm for $Q_{r;d} = (\ _I; \ _O; L; Ball_{r;d}(L_I), M, cost, goal)$ under the assumption $P \not= NP$, then this means that the problem Q is \unstable" according to d.

Thus, using the notion of stability one can search for a spectrum of the hardness of a problem according to the set of input instances, which is the main aim of our concept. This has been achieved for TSP now. Collecting results of Theorem 1 and of [BC99], we have $\min f\frac{3}{2} \ ^2; 4 \ g$-approximation algorithms for $_{;dist}$-TSP, and following [BC99], $_{;dist}$-TSP is not approximable within a factor 1 + " for some " < 1. While TSP does not seem to be tractable from the previous point of view of approximation algorithms, using the concept of approximation stability, it may look tractable for many speci c applications.

References

[Ar97] S. Arora: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 38th IEEE FOCS*, 1997, pp. 554{563.

[Ar98] S. Arora: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. In: *Journal of the ACM* 45, No. 5 (Sep. 1998), pp. 753{782.

[AB95] T. Andreae, H.-J. Bandelt: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM J. Discr. Math.* 8 (1995), pp. 1{16.

[BC93] D. P. Bovet, P. Crescenzi: *Introduction to the Theory of Complexity,* Prentice-Hall 1993.

[BC99] M. A. Bender, C. Chekuri: Performance guarantees for the TSP with a parameterized triangle inequality. In: *Proc. WADS'99, LNCS*, to appear.

[BHKSU00] H.-J. Böckenhauer, J. Hromkovic, R. Klasing, S. Seibert, W. Unger: An Improved Lower Bound on the Approximability of Metric TSP and Approximation Algorithms for the TSP with Sharpened Triangle Inequality (Extended Abstract). In: *Proc. STACS'00, LNCS*, to appear.

[Chr76] N. Christo des: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[Co71] S. A. Cook: The complexity of theorem proving procedures. In: *Proc. 3rd ACM STOC*, ACM 1971, pp. 151{158.

[CK98] P. Crescenzi, V. Kann: *A compendium of NP optimization problems.* `http://www.nada.kth.se/theory/compendium/`

[CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest: *Introduction to algorithms*. MIT Press, 1990.

[En99] L. Engebretsen: An explicit lower bound for TSP with distances one and two. Extended abstract in: *Proc. STACS'99, LNCS 1563*, Springer 1999, pp. 373{382. Full version in: *Electronic Colloquium on Computational Complexity*, Report TR98-046 (1999).

[GJ79] M. R. Garey, D. S. Johnson: *Computers and vIntractability. A Guide to the Theory on NP-Completeness.* W. H. Freeman and Company, 1979.

[Ha97] J. Hastad: Some optimal inapproximability results. Extended abstract in: *Proc. 29th ACM STOC*, ACM 1997, pp. 1{10. Full version in: *Electronic Colloquium on Computational Complexity,* Report TR97-037, (1999).

[Ho96] D. S. Hochbaum (Ed.): *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company 1996.

[Hr98] J. Hromkovic: Stability of approximation algorithms and the knapsack problem. Unpublished manuscript, RWTH Aachen, 1998.

[IK75] O. H. Ibarra, C. E. Kim: Fast approximation algorithms for the knapsack and sum of subsets problem. *J. of the ACM* 22 (1975), pp. 463{468.

[Jo74] D. S. Johnson: Approximation algorithms for combinatorial problems. *JCSS* 9 (1974), pp. 256{278.

[Lo75] L. Lovasz: On the ratio of the optimal integral and functional covers. *Discrete Mathematics* 13 (1975), pp. 383{390.

[LLRS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys (Eds.): *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[Mi96] I. S. B. Mitchell: Guillotine subdivisions approximate polygonal subdivisions: Part II | a simple polynomial-time approximation scheme for geometric *k*-MST, TSP and related problems. Technical Report, Dept. of Applied Mathematics and Statistics, Stony Brook 1996.

[MPS98] E. W. Mayr, H. J. Prömel, A. Steger (Eds.): *Lectures on Proof Veri cation and Approximation Algorithms. LNCS* 1967, Springer 1998.

[Pa77] Ch. Papadimitriou: The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science* 4 (1977), pp. 237{244.

[Pa94] Ch. Papadimitriou: *Computational Complexity,* Addison-Wesley 1994.

[PY93] Ch. Papadimitriou, M. Yannakakis: The traveling salesman problem with distances one and two. *Mathematics of Operations Research* 18 (1993), 1{11.

Semantical Counting Circuits[⋆]

Fabrice Noilhan[1] and Miklos Santha[2]

[1] Universite Paris-Sud, LRI, Bât. 490, 91405 Orsay, France
`Fabrice.Noilhan@lri.fr`
[2] CNRS, URA 410, Universite Paris-Sud, LRI, Bât. 490, 91405 Orsay, France
`Miklos.Santha@lri.fr`

Abstract. Counting functions can be de ned syntactically or semantically depending on whether they count the number of witnesses in a non-deterministic or in a deterministic computation on the input. In the Turing machine based model, these two ways of de ning counting were proven to be equivalent for many important complexity classes. In the circuit based model, it was done for #P and #L, but for low-level complexity classes such as $\#AC^0$ and $\#NC^1$ only the syntactical de - nitions were considered. We give appropriate semantical de nitions for these two classes and prove them to be equivalent to the syntactical ones. This enables us to show that $\#AC^0$ is included in the family of counting functions computed by polynomial size and constant width counting branching programs, therefore completing a result of Caussinus et al [CMTV98]. We also consider semantically de ned probabilistic complexity classes corresponding to AC^0 and NC^1 and prove that in the case of unbounded error, they are identical to their syntactical counterparts.

1 Introduction

Counting is one of the basic questions considered in complexity theory. It is a natural generalization of non-determinism: computing the number of solutions for a problem is certainly not easier than just deciding if there is a solution at all. Counting has been extensively investigated both in the machine based and in the circuit based models of computation.

Historically, the rst counting classes were de ned in Turing machine based complexity theory. Let us call a non-deterministic Turing machine an NP-machine if it works in polynomial time, and an NL-machine if it works in logarithmic space. In the case of a non-deterministic machine, an accepting path in its computation tree on a string x certi es that x is accepted. We will call such a path a *witness* for x. The very rst, and still the most famous, counting class called #P was introduced by Valiant [Val79] as the set of counting functions that map a string x to the number of witnesses for x of some NP-machine. An analogous de nition was later made by Alvarez and Jenner [AJ93] for the class #L: it contains the set of counting functions that map x to the number of witnesses for x of

[?] This research was supported by the ESPRIT Working Group RAND2 No. 21726 and by the French-Hungarian bilateral project Balaton, Grant 99013 some NL-machine. These classes contain several natural complete problems: for example computing the permanent of a matrix is complete in #P, whereas computing the number of paths in a directed graph between two speci ed vertices is complete in #L.

The so-called \Gap" classes were de ned subsequently to include functions taking also negative values into the above model. GapP was introduced by Fenner, Fortnow and Kurtz [FFK94] as the di erence of two functions in #P. The analogous de nition for GapL was made independently by Vinay [Vin91], Toda [Tod91], Damm [Dam91] and Valiant [Val92]. This later class has received considerable attention, mostly because it characterizes the complexity of computing the determinant of a matrix [AO96, ST98, MV97].

Still in the Turing machine model, there is an alternative way of de ning the classes #P and #L, based on the computation of deterministic machines. In the following discussion let us consider deterministic Turing machines acting on pairs of strings $(x;y)$ where for some polynomial $p(n)$, the length of y is $p(jxj)$. We will say that the string y is a *witness* for x when the machine accepts $(x;y)$, otherwise y is a *non-witness*. We will call a deterministic Turing machine a P-machine if it works in polynomial time, and an L-machine if it works in logarithmic space and it has only one-way access to y. Then #P (respectively #L) can be de ned as the set of functions f for which there exists a P-machine (respectively L-machine) such that $f(x)$ is the number of witnesses for x. The equivalence between these de nitions can be established if we interpret the above deterministic Turing machines as a normal form, with simple witness structure, for the corresponding non-deterministic machines, where the string y describes the sequence of choices made during the computation on x. Nonetheless this latter way of looking at counting has at least two advantages over the previous one.

The rst advantage is that this de nition is more robust in the following sense. Two non-deterministic machines, even if they compute the same relation $R(x)$, might de ne di erent counting functions depending on their syntactical properties. On the other hand, if the de nition is based on deterministic machines, only the relation they compute is playing a role. Indeed, two deterministic machines computing the same relation $R(x;y)$ will necessarily de ne the same counting function independently from the syntactical properties of their computation. Therefore, from now on, we will refer to the non-deterministic machine based de nition of counting as *syntactical*, and to the deterministic machine based de nition as *semantical*.

The second advantage of the semantical de nition of counting is that probabilistic complexity classes can be de ned more naturally in that setting. For example PP(respectively PL) is just the set of languages for which there exists a P-machine (respectively L-machine) such that a string x is in the language exactly when there are more witnesses for x than non-witnesses. In the case of the syntactical de nition of counting the corresponding probabilistic classes usually are de ned via the Gap classes.

The above duality in the de nition of counting exists of course in other models where determinism and non-determinism are meaningful concepts. This is the case of the circuit based model of computation. Still, in this model syntactical counting has received considerably more attention than semantical counting. Before we discuss the reason for that, let us make clear what do we mean here by these notions.

The syntactical notion of a witness for a string x in a circuit family was de ned by Venkateswaran [Ven92] as an accepting subtree of the corresponding circuit on x, which is a smallest sub-circuit certifying that the circuit's output is 1 on x. It is easy to show that the number of such witnesses is equal to the value of the arithmetized version of the circuit on x. Let us stress again that this number, and therefore the counting function de ned by a circuit, depends heavily on the speci c structure of the circuit and not only on the function computed by it. For example if we consider circuit C_1 which is just the variable x, and circuit C_2 which consists of an OR gate whose both inputs are the same variable x, then clearly these two circuits compute the same function. On the other hand, on input $x = 1$, the counting function de ned by C_1 will take the value 1, whereas the counting function de ned by the circuit C_2 will take the value 2.

For the semantical notion of a witness we consider again families whose inputs are pairs of strings of polynomially related lengths. As in the case of Turing machines, y is a witness for x if the corresponding circuit outputs 1 on $(x; y)$.

Venkateswaran was able to give a characterization of #P and #L in the circuit model based on the syntactical de nition of counting. His results rely on a circuit based characterization of NP and NL. He has shown that #P is equal to the set of counting functions computed by uniform semi-unbounded circuits of exponential size and of polynomial algebraic degree; and #L is equal to the set of counting functions computed by uniform skew-symmetric circuits of polynomial size. Semantically #P can be characterized as the set of counting functions computed by uniform polynomial size circuits.

In recent years several low level counting classes were de ned in the circuit based model, all in the syntactical setting. Caussinus et al.[CMTV98] have de ned $\#\mathrm{NC}^1$, and Agrawal et al. [AAD97] have de ned $\#\mathrm{AC}^0$ as the set of functions counting the number of accepting subtrees in the respective circuit families. In subsequent works, many important properties of these classes were established [ABL98, AAB$^+$99]. Although some attempts were made [Yam96], no satisfactory characterization of these classes was obtained in the semantical setting. The main reason for that is that by simply adding \counting" bits to AC^0 or NC^1 circuits, we fall to the all too powerful counting class #P [SST95, VW96], and it is not at all clear what type of restrictions should be made in order to obtain $\#\mathrm{AC}^0$ and $\#\mathrm{NC}^1$.

The main result of this paper is such a semantical characterization of these two counting classes. Indeed, we will de ne semantically the classes $\#\mathrm{AC}^0_{\mathrm{CO}}$ and $\#\mathrm{NC}^1_{\mathrm{CO}}$ by putting some relatively simple restrictions on the structure of AC^0 and NC^1 circuits involved in the de nition, and on the way they might contain counting variables. Our main result is that this de nition is equivalent to the syntactical de nition, that is we have

Theorem 1. $\#AC^0 = \#AC^0_{CO}$ *and* $\#NC^1 = \#NC^1_{CO}$.

Put it another way, if standard AC^0 and NC^1 are seen as \non-deterministic" circuit families in the syntactical de nition of the corresponding counting classes, we are able to characterize their \deterministic" counterparts which de ne the same counting classes semantically.

We also examine the relationship between #BR, the family of counting functions computed by polynomial size and constant width counting branching programs, and counting circuits. While Caussinus et al. [CMTV98] proved that $\#BR \quad \#NC^1$, we will show

Theorem 2. $\#AC^0 \quad \#BR$.

Semantically de ned counting classes give rise naturally to the corresponding probabilistic classes in the three usually considered cases: in the unbounded, in the bounded and in the one sided error model. Indeed, we will de ne the probabilistic classes PAC^0_{CO}, PNC^1_{CO}, $BPAC^0_{CO}$, $BPNC^1_{CO}$, RAC^0_{CO} and RNC^1_{CO}. PAC^0 and PNC^1 were already de ned syntactically via $\#AC^0$ and $\#NC^1$, and we will prove for this model that our de nitions coincide with previous ones:

Theorem 3. $PAC^0_{CO} = PAC^0$ *and* $PNC^1_{CO} = PNC^1$*:*

In the other two error models, previous de nitions were also semantical, but without any restrictions on the way the corresponding circuits could use the counting variables. We couldn't determine if they coincide with ours, and we think that this question is worth of further investigations. Nonetheless we argue that because of their close relationship with counting branching programs, the counting circuit based de nition might be the right one.

The paper is organized as follows: Section 2 contains the de nitions for semantical circuit based counting. Section 3 exhibits the mutual simulations of syntactical and semantical counting for the circuit classes AC^0 and NC^1. Theorem 1 is a direct consequence of Theorems 4 and 5 proven here. In section 4 we deal with counting branching programs, and Theorem 2 will follow from Theorem 6. Finally in section 5 we discuss the gap and random classes which are derived from semantical counting circuits. Theorem 8 relating gap classes and counting circuits will imply Theorem 3.

2 De nitions

In this chapter we de ne *counting circuit families* which will be used for the semantical de nition of a counting function. Counting circuits have two types of input variables: standard and counting ones. They are in fact restricted boolean circuits, where the restriction is put on the way the gates and the counting variables can be used in the circuits. First we will de ne the usual boolean circuit families and the way they are used to de ne (syntactically) counting functions, and then we do the same for counting circuit families. The names "circuit" versus "counting circuit" will be used systematically this way in the rest of the paper.

A *bounded fan-in circuit* with n input variables is a directed acyclic graph with vertices of in-degree 0 or 2. The vertices of in-degree 0 are called inputs, and they are labeled with an element of the set $\{0, 1, x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$. The vertices of in-degree 2 are labeled with the bounded AND or OR gate. There is a distinguished vertex of outdegree 0, this is the output of the circuit. An *unbounded fan-in circuit* is defined similarly with the only difference that non input vertices can have arbitrary in-degree, and they are labeled with unbounded AND or OR gate. A *circuit family* is a sequence $(C_n)_{n=1}^{\infty}$ of circuits where C_n has n input variables. It is *uniform* if its direct connection language is computed in DLOGTIME. An AC^0 circuit family is a uniform, unbounded fan-in circuit family of polynomial size and constant depth. An NC^1 circuit family is a uniform, bounded fan-in circuit family of polynomial size and logarithmic depth.

A circuit C is a *tree circuit* if all its vertices have out-degree 1. A *proof tree* in C on input x is a connected subtree which contains its output, has one edge into each OR gate, has all the edges into the AND gates, and which evaluates to 1 on x. The number of proof trees in C on x will be denoted by $\#\mathrm{PT}_C(x)$. A boolean tree circuit family $(C_n)_{n=1}^{\infty}$ computes a function $f : \{0,1\}^* \to \mathbb{N}$ if for every x, we have $f(x) = \#\mathrm{PT}_{C_{|x|}}(x)$. We denote by #AC^0 (respectively by #NC^1 the class of functions computed by a uniform AC^0 (respectively NC^1) tree circuit family.

In order to introduce counting variables into counting circuits and to carry out the syntactical restrictions, we use two new gates, SELECT and PADAND gates. These are actually small circuits which will be built some specific way from AND and OR gates. The SELECT gates which use a counting variable to choose a branch of the circuit will actually replace OR gates which will be prohibited in their general form. The PADAND gates will function as AND gates, but they will allow again the introduction of counting variables. They will actually fix the value of these counting variables to the constant 1.

We now define formally these gates. In the following we will denote single boolean variables with a subscript such as v_0. Boolean vector variables will be denoted without a subscript, such as v. We will also identify an integer $0 \le s \le 2^k - 1$ with its binary representation $(s_0, \ldots, s_{k-1})$.

The bounded fan-in SELECT gate will have 3 arguments. It is defined by $\mathrm{SELECT}^1(x_0, x_1, u_0) = x_{u_0}$, and represented by $\mathrm{OR}(\mathrm{AND}(x_0, \overline{u_0}), \mathrm{AND}(x_1, u_0))$. For every k, the unbounded fan-in SELECT^k gate has $2^k + k$ arguments and is defined by $\mathrm{SELECT}^k(x_0, \ldots, x_{2^k-1}, u_0, \ldots, u_{k-1}) = x_u$. This gate is represented by the circuit $\mathrm{OR}_{i=0}^{2^k-1}(\mathrm{AND}(x_i, u = i))$ where $u = i$ stands for the circuit $\mathrm{AND}_{j=0}^{k-1}(\mathrm{OR}(\mathrm{AND}(\overline{u_j}, \overline{i_j}), \mathrm{AND}(u_j, i_j)))$. The last gate can easily be extended to $m + k$ arguments for $m < 2^k$ as $\mathrm{SELECT}^k(x_0, \ldots, x_{m-1}, u_0, \ldots, u_{k-1}) = \mathrm{SELECT}^k(x_0, \ldots, x_m, 0, \ldots, 0, u_0, \ldots, u_{k-1})$. Clearly, SELECT^k can be simulated by a circuit of depth $O(\log k)$ containing only SELECT^1 gates. The unbounded fan-in PADAND gate has at least two arguments and is defined by

PADAND$(x_0, u_0, \ldots, u_l) = \text{AND}(x_0, u_0, \ldots, u_l)$. Its bounded fan-in equivalent PADAND^b can also have an arbitrary number of arguments, and in case of m arguments is represented by a circuit of depth $\lceil \log m \rceil$ consisting of a balanced binary tree of bounded AND gates. It will always be clear from the context if we are dealing with the bounded or the unbounded PADAND gate.

We will define recursively unbounded fan-in counting circuits. There will be two types of input variables: "standard" and "counting" ones.

Definition 1 (Counting circuit).

– *If C is a boolean tree circuit, then C is a counting circuit. All its variables are standard.*
– *If $C_0, \ldots, C_{2^k-1}$ are counting circuits and $u_0, \ldots, u_{k-1}$ are input variables which are not appearing in them, then* SELECT$(C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1})$ *is a counting circuit. The variables $u_0, \ldots, u_{k-1}$ are counting variables.*
– *If $C_0, \ldots, C_k$ are counting circuits and they do not have any common counting variables, then* AND$(C_0, \ldots, C_k)$ *is a counting circuit.*
– *If C is a counting circuit and $u_0, \ldots, u_l$ are input variables, then* PADAND$(C, u_0, \ldots, u_l)$ *is a counting circuit. The variables $u_0, \ldots, u_l$ are counting variables.*

Moreover, we require that no input variable can be counting and standard at the same time.

Bounded counting circuits are defined analogously, with $k = 1$ in all the construction steps.

The set of all standard (respectively counting) variables of a circuit C will be denoted $\mathrm{SV}(C)$ (respectively $\mathrm{CV}(C)$). Let C be a counting circuit with n standard variables. The *counting function* $\#\text{CO}_C : \{0,1\}^n \mapsto \mathbb{N}$ associated with C is defined as:

$$\#\text{CO}_C(x) = \begin{cases} C(x) & \text{if } \mathrm{CV}(C) = \emptyset, \\ \#\{u \mid C(x, u) = 1\} & \text{if } \mathrm{CV}(C) \neq \emptyset. \end{cases}$$

A sequence $(C_n)_{n=1}^{\infty}$ of counting circuits is a *counting family* if there exists a polynomial p such that for all n, C_n has n standard variables and at most $p(n)$ counting variables. A family is *uniform* if its direct connection language is computed in DLOGTIME. The *counting function* computed by a circuit family is defined as $\#\text{CO}_{C_{|x|}}(x)$. Finally, the *semantical counting classes* are defined as follows: $\#\text{AC}^0_{\text{CO}}$ (respectively $\#\text{NC}^1_{\text{CO}}$) is the set of functions computed by a uniform AC^0 (NC^1) family of counting circuits.

3 Circuits and Counting Circuits

3.1 Simulating Circuits by Counting Circuits

We will use a step-by-step simulation. We will define a function which maps circuits into counting circuits by structural recursion on the output gate G of the circuit. The de nition will be done for the unbounded case from which the bounded case can be obtained by replacing the parameter k with 1 in all the construction steps, and unbounded gates by bounded ones.

De nition 2 (the function). *If G is a literal, then $(G) = G$ and the corresponding variable is standard. If G is an* AND *gate whose entries are the circuits $C_0, \dots, C_k$, then let the circuits C'_i be obtained from (C_i) by renaming counting variables so that $\forall i \neq j$, $\mathrm{CV}(C'_i) \cap \mathrm{CV}(C'_j) = \mathrm{CV}(C'_i) \cap \mathrm{SV}(C'_j) = \emptyset$. Then $(C) =$* AND*$(C'_0, \dots, C'_k)$. If G is an* OR *gate whose entries are the circuits $C_0, \dots, C_{2^k-1}$, then the circuits C'_i are obtained from (C_i) by renaming counting variables so that $\forall i \neq j$, $\mathrm{CV}(C'_i) \cap \mathrm{CV}(C'_j) = \mathrm{CV}(C'_i) \cap \mathrm{SV}(C'_j) = \emptyset$. Let $V = \mathrm{CV}(C'_0) \cup \dots \cup \mathrm{CV}(C'_{2^k-1})$ and $V_i = V - \mathrm{CV}(C'_i)$. Let C''_i be de ned as* PADAND*(C'_i, V_i), and let $u_0, \dots, u_{k-1}$ be counting variables such that $\{u_0, \dots, u_{k-1}\} \cap V = \emptyset$. Then $(C) =$* SELECT*$(C''_0, \dots, C''_{2^k-1}, u_0, \dots, u_{k-1})$.*

The next two lemmas will prove that the de nition of is correct and that the functions computed by the corresponding circuit families are equal.

Lemma 1. *If (C_n) is a uniform* AC^0 *(respectively* NC^1*) family of circuits, then $((C_n))$ is a uniform* AC^0 *(resp.* NC^1*) family of counting circuits.*

Proof. Throughout the construction, we assured that the entry circuits of an AND gate do not have common counting variables. Clearly, no input variable can be counting and standard at the same time.

Since there are a polynomial number of gates and for each gate, we introduced a polynomial number of counting variables, the number of counting variables is bounded by a polynomial. The uniformity of $((C_n))$ follows from the uniformity of (C_n). To nish the proof, we should consider the depth of the counting circuits.

In the unbounded case, $((C_n))$ is of constant depth since the SELECT gates which replace the OR gates of the original circuit are of constant depth.

In the bounded case, let k be such that there are at most n^k variables in C_n. The depth of C_n is $O(\log n)$. Let us de ne $d_i = \max\{\mathrm{depth}((D))\}$ where D is a subcircuit of C_n of depth i. Then we have

$$d_{i+1} \quad 3 + \max(d_i, k \quad \log n)$$

since the depth increases only when the output gate is an OR. Therefore, $((C_n))$ is of logarithmic depth. ⊓⊔

Lemma 2. *For every circuit C, $\#\mathrm{PT}_C(x) = \#\mathrm{CO}_{(C)}(x)$.*

Proof. We will prove this by structural recursion on the output gate G of C. If G is a literal, then by de nition, circuits and counting circuits de ne the same counting function. If G is an AND gate then, since for $i = 0, \dots, k$ the variables in $\mathrm{CV}(C'_i)$ are distinct, $\#\mathrm{CO}_{(C)}(x) = \prod \#\mathrm{CO}_{C'_i}(x)$, which is the same as $\prod \#\mathrm{CO}_{(C_i)}(x)$ because C'_i was obtained from (C_i) by renaming the variables. By the inductive hypothesis and the de nition of the proof tree model, this is equal to $\#\mathrm{PT}_C(x)$. If G is an OR gate then since the counting variables $u_0,\ldots,u_{k-1}$ are distinct from the counting variables of the subcircuits, $\#\mathrm{CO}_{(C)}(x) = \#\mathrm{CO}_{C_i''}(x)$. For every i, $\#\mathrm{CO}_{C_i''}(x) = \#\mathrm{CO}_{C_i'}(x)$ since the PADAND gate fixes all the counting variables outside V_i. This is the same value as $\#\mathrm{CO}_{(C_i)}(x)$ since C_i' was obtained from (C_i) by renaming the variables. The statement follows from the inductive hypothesis.

The two lemmas imply

Theorem 4. $\#\mathrm{AC}^0$ $\#\mathrm{AC}^0_{\mathrm{CO}}$ *and* $\#\mathrm{NC}^1$ $\#\mathrm{NC}^1_{\mathrm{CO}}$.

3.2 Simulating Counting Circuits by Circuits

Let us remark first that any counting circuit C can be easily transformed to another counting circuit C' computing the same counting function such that if $\mathrm{PADAND}(D',u_0,\ldots,u_l)$ is a subcircuit of C', then $\{u_0,\ldots,u_l\} \cap \mathrm{CV}(D') = \emptyset$. This is indeed true since if $\mathrm{PADAND}(D,u_0,\ldots,u_l)$ is a subcircuit of C and if for example $u_0 \in \mathrm{CV}(D)$ then we can rewrite D with respect to $u_0 = 1$ by modifying SELECT and PADAND gates accordingly. For the rest of the paper, we will suppose that counting circuits have been transformed this way.

We will use in the construction circuits computing fixed integers which are powers of 2. For l 0 the circuit A_{2^l} computing the integer 2^l is defined as follows. A_1 is the constant 1 and A_2 is $\mathrm{OR}(1,1)$. For l 2, in the unbounded case, A_{2^l} has a topmost unbounded AND gate with l subcircuits A_2. In the bounded case, we replace the unbounded AND gate by its standard bounded simulation consisting of a balanced binary tree of bounded AND gates. Clearly, the depth of A_{2^l} in the bounded case is $\lceil \log l \rceil + 1$.

We now define a function which maps counting circuits into circuits by structural recursion on the output gate G of the counting circuit. Again, the definition will be done for the unbounded case, from which the bounded case can be obtained by replacing the parameter k with 1.

Definition 3 (the function). *If G is a literal, then $(G) = G$. If G is an* AND *gate whose entries are $C_0,\ldots,C_k$, then $(C) = \mathrm{AND}((C_0),\ldots,(C_k))$. If G is a* PADAND *whose entries are C_0 and $u_0,\ldots,u_l$, then $(C) = (C_0)$. If G is a* SELECT *gate whose entries are $C_0,\ldots,C_{2^k-1},u_0,\ldots,u_{k-1}$ then set $V = \mathrm{CV}(C_0)\cup\ldots\cup\mathrm{CV}(C_{2^k-1})$ and $V_i = V - \mathrm{CV}(C_i)$. We let $C_i' = \mathrm{AND}((C_i), A_{2^{|V_i|}})$ and $(C) = \mathrm{OR}(C_0',\ldots,C_{2^k-1}')$.*

Again, we proceed with two lemmas to prove the correctness of the simulation.

Lemma 3. *If (C_n) is a uniform* AC^0 *(respectively* NC^1*) family of counting circuits, then $((C_n))$ is a uniform* AC^0 *(resp.* NC^1*) family of circuits.*

Proof. In the construction, we get rid of PADAND and SELECT gates and of the counting variables. We do not modify the number of standard variables. The only step where we increase the size or the depth of the circuit is when SELECT gates are replaced. Each replacement introduces at most a polynomial number of gates. Therefore the size remains polynomial. Uniformity of ((C_n)) follows from the uniformity of (C_n).

In the unbounded case, the depth remains constant since the circuits A_{2^l} have constant depth. In the bounded case, we claim that every replacement of a SELECT gate increases the depth by a constant. This follows from the fact that depth$(A_{2^{|V_i|}})$ depth$(C_{1-i}) + 1$ for $i = 0; 1$ since a bounded counting circuit with m counting variables has depth at least $d\log(m + 1)e$. Therefore the whole circuit remains of logarithmic depth. ut

Lemma 4. *For every counting circuit C, $\#\mathrm{PT}_{(C)}(x) = \#\mathrm{CO}_C(x)$.*

Proof. We will prove this by structural recursion on the output gate G of the counting circuit. In the proof, we will use the notation of de nition 3. If G is a literal, then by de nition, C and (C) de ne the same counting function. If G is an AND gate then by de nition $\#\mathrm{PT}_{(C)}(x) = \mathrm{Q}\ \#\mathrm{PT}_{(C_i)}(x)$. Since for $i = 0; : : : ; k$ the subcircuits C_i do not share common counting variables, using the inductive hypothesis this is equal to $\#\mathrm{CO}_C(x)$. If G is a PADAND gate then from the de nition and the inductive hypothesis, we have $\#\mathrm{PT}_{(C)}(x) = \#\mathrm{CO}_{C_0}(x)$. Since for all i, v_i 2 CV(C_0), we have $\#\mathrm{CO}_{C_0}(x) = \#\mathrm{CO}_C(x)$. If G is a SELECT gate then $\#\mathrm{PT}_{(C)}(x) = \mathrm{P}\ 2^{jV_ij}\ \#\mathrm{PT}_{(C_i)}(x)$. Also, $\#\mathrm{CO}_C(x) = \mathrm{P}\ 2^{jV_ij}\ \#\mathrm{CO}_{C_i}(x)$ since the value of variables in V_i do not influence the value of the circuit C_i. The result follows from the inductive hypothesis. ut

Theorem 5. $\#\mathrm{AC}^0_{\mathrm{CO}}$ $\#\mathrm{AC}^0$ *and* $\#\mathrm{NC}^1_{\mathrm{CO}}$ $\#\mathrm{NC}^1$.

4 Branching Programs and Counting Circuits

Branching problems constitute another model for de ning low level counting classes, and this possibility was rst explored by Caussinus et al. [CMTV98]. Let us recall that a *counting branching program* is an ordinary branching program [Bar89] with two types of (standard and counting) variables with the restriction that every counting variable appears at most once on each computation path. Given a branching program $B(x; u)$ whose standard variables are x, the counting function computed by it is

$$\#B(x) = \begin{cases} B(x) & \text{if } B \text{ does not have counting variables,} \\ \# f u \mid B(x; u) = 1 g & \text{otherwise.} \end{cases}$$

Finally #BR is the family of counting functions computed by DLOGTIME-uniform polynomial size and constant width counting branching programs. Caussinus et al. [CMTV98] could prove that #BR $\#\mathrm{NC}^1$, but they left open the question if the inclusion was proper. Our results about counting circuits enable us to show that $\#\mathrm{AC}^0$ #BR. Our proof will proceed in two steps. First we show that $\#\mathrm{AC}^0_{\mathrm{CO}}$ is included in #BR and then we use theorem 5 to conclude.

Theorem 6. $\#\mathrm{AC}^0_{\mathrm{CO}}$ $\#\mathrm{BR}$:

Proof. We de ne a function which maps counting circuits into synchronous counting branching programs such that for every counting circuit C, $\#\mathrm{CO}_C$ = # (C). The de nition will be done by structural recursion on the output gate G of the counting circuit, and the proof of this statement can be done without di culty also by recursion.

If C is a boolean tree circuit, then (C) is a width-5 polynomial size branching program provided by Barrington's result which recognizes the language accepted by C. If C_0 and C_1 are counting circuits and u is a new counting variable then (SELECT$(C_0; C_1; u)$) is the branching program whose source is labelled by u, and whose two successors are the source nodes of respectively (C_0) and (C_1). If C_0 and C_1 are counting circuits without common variables, then by increasing the width of (C_0) by 1, we ensure that it has a unique 1-sink. Then (AND$(C_0; C_1)$) is the branching program whose source is the source of (C_0) whose 1-sink is identi ed with the source of (C_1). If C is a counting circuit and $u_0; \ldots; u_k$ are counting variables, then (PADAND$(C; u_0; \ldots; u_k)$) = (C') where C' is obtained from C by setting $u_0; \ldots; u_k$ to 1.

If $(C_n)_{n=1}^{\infty}$ is an AC^0 family of counting circuits, then ((C_n)) has constant width since we double the width only a constant number of times for the SELECT gates. Also, the uniformity of the family follows from the uniformity of (C_n). ⊓⊔

5 Gap and Random Classes via Semantical Counting

In this section, we will point out another similarity between semantical counting circuits and deterministic Turing machine based counting: we will de ne probabilistic classes by counting the fraction of assignments for the counting variables which make the circuit accept. We will prove that in the unbounded error case, our de nitions coincide with the syntactical de nition via Gap classes. For the proof, we will need the notion of an extended counting circuit which may contain OR and PADOR gates without changing the family of counting functions they can compute. In the bounded error and one sided error models we could not determine if our de nitions and the syntactical ones are identical.

5.1 Extented Counting Circuits

By de nition, the unbounded fan-in PADOR$(x_0; u_0; \ldots; u_l)$ gate has at least two arguments and is de ned as OR$(x_0; \overline{u_0}; \ldots; \overline{u_l})$. Its bounded fan-in equivalent is represented by a circuit of depth $\lceil \log(l + 2) \rceil$, consisting in the usual bounded expansion of the above circuit. Similarly to the case of the PADAND gate, from now on we will suppose without loss of generality that if PADOR$(D; u_0; \ldots; u_l)$ is a subcircuit of an extended counting circuit, then $\mathrm{CV}(D) \cap \{u_0; \ldots; u_l\} = \emptyset$.

An unbounded fan-in *extended counting circuit* is a counting circuit with the following additional construction steps to the De nition 1.

- if $C_0, \ldots, C_k$ are extended counting circuits and they do not have any common counting variable then $\mathrm{OR}(C_0, \ldots, C_k)$ is an extended counting circuit.
- if C is an extended counting circuit and $u_0, \ldots, u_l$ are input variables, then $\mathrm{PADOR}(C, u_0, \ldots, u_l)$ is an extended counting circuit. The variables $u_0, \ldots, u_l$ are counting variables.
- if C is a counting circuit, $\overline{C}$ is an extended counting circuit.

We obtain the de nition for the bounded fan-in case by taking again $k = 1$. We will denote by $\#\mathrm{AC}^0_{\mathrm{ECO}}$ (respectively $\#\mathrm{NC}^1_{\mathrm{ECO}}$) the set of functions computed by a uniform AC^0 (respectively NC^1) family of extended counting circuits. The following theorem whose proof will be given in the appendix shows that extended counting circuits are no more powerful than regular ones.

Theorem 7. $\#\mathrm{AC}^0_{\mathrm{ECO}} = \#\mathrm{AC}^0_{\mathrm{CO}}$ *and* $\#\mathrm{NC}^1_{\mathrm{ECO}} = \#\mathrm{AC}^0_{\mathrm{CO}}$.

Proof. Let C be an extended counting circuit. We will show that there exists a counting circuit computing $\#\mathrm{CO}_C$ whose size and depth is of the same order of magnitude. First observe that negation gates in C can be pushed down to the literals. For this, besides the standard de Morgan laws, one can use the following equalities whose veri cation is straightforward:

$$\overline{\mathrm{SELECT}(C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1})} = \mathrm{SELECT}(\overline{C_0}, \ldots, \overline{C_{2^k-1}}, u_0, \ldots, u_{k-1}),$$

$$\overline{\mathrm{PADAND}(C_0, (u_0, \ldots, u_l))} = \mathrm{PADOR}(\overline{C_0}, (u_0, \ldots, u_l)),$$

$$\overline{\mathrm{PADOR}(C_0, (u_0, \ldots, u_l))} = \mathrm{PADAND}(\overline{C_0}, (u_0, \ldots, u_l)).$$

Then one can get rid of the OR gates by recursively replacing $\mathrm{OR}(C_0, \ldots, C_{2^k-1})$ with $\mathrm{SELECT}(C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1})$ where $\{u_0, \ldots, u_{k-1}\} \cap \mathrm{CV}(C_0) \cup \ldots \cup \mathrm{CV}(C_{2^k-1}) = \,$. Since $C_0, \ldots, C_{2^k-1}$ do not have any common counting variables, this does not change the counting function computed by the counting circuit. Finally we show how to extend the function of De nition 2 to PADOR gates while keeping the same counting function as in Lemma 2. We de ne

$$(\mathrm{PADOR}(C_0, (u_0, \ldots, u_l))) = \mathrm{OR}(A_{2^{|\mathrm{CV}((C_0))|} (2^{l+1}-1)}, (C_0)).$$

Then, $\#\mathrm{PT}_{(C)}(x) = \#\mathrm{PT}_{(C_0)}(x) + (2^{|\mathrm{CV}((C_0))|} (2^{l+1} - 1))$ which by the inductive hypothesis is $\#\mathrm{CO}_{C_0}(x) + (2^{|\mathrm{CV}((C_0))|} (2^{l+1} - 1)).$ This in turn equals to $\#\mathrm{CO}_C(x)$ by de nition of the PADOR gate and since $u_0, \ldots, u_l$ are not counting variables of C_0.

Since all these transformations may increase the size or the depth only by a constant factor, the statement follows. ⊓⊔

5.2 Gap Classes

As usual, for any counting class #C, we de ne the associated "Gap" class GapC. A function f is in GapC i there are two functions f_1 and f_2 in #C such that $f = f_1 - f_2$. The following theorem will be useful for discussing probabilistic complexity classes.

Theorem 8. *Let f be a function in* GapAC^0 *(respectively* GapNC^1*) . Then there is an* AC^0 *(resp.* NC^1*) uniform family of counting circuits* (C_n) *such that* $2 \cdot f(x) = \#\mathrm{CO}_{C_{|x|}}(x) - \#\mathrm{CO}_{\overline{C_{|x|}}}(x)$.

Proof. In fact, we will construct a family of extended counting circuits with the required property. Then, the result will follow from theorem 7. Let #C be one of the classes #AC^0 or #NC^1, and let f be a function in GapC. Then there exist two functions in #C, f_1 and f_2 such that $f = f_1 - f_2$. Fix an entry x of length n. Let us take two uniform #C families of counting circuits which compute f_1 and f_2, and let respectively D_1 and D_2 be the counting circuits in these families which have n input variables.

Let $V_1 = \mathrm{CV}(D_1)$, $V_2 = \mathrm{CV}(D_2)$ and $m = |V_1| + |V_2|$. We will suppose without loss of generality that $V_1 \cap V_2 = \emptyset$ (by renaming the variables if necessary). We define $D_1' = \mathrm{PADAND}(D_1, V_2)$ and $D_2' = \mathrm{PADAND}(D_2, V_1)$. Let t be a counting variable such that $t \notin V_1 \cup V_2$ and define $C_n = \mathrm{SELECT}(D_1', \overline{D_2'}, t)$. The counting circuit family (C_n) is uniform and is in C since its depth and size are changed only up to a constant with respect to the families computing f_1 and f_2.

We first claim that the counting function associated with C_n, on entry x, computes $f_1(x) + (2^m - f_2(x))$. First, let us observe that $\#\mathrm{CO}_{\overline{D_2'}}(x) = 2^m - \#\mathrm{CO}_{D_2'}(x)$. Therefore, since t does not appear in D_1' and D_2', we have $\#\mathrm{CO}_{C_n}(x) = \#\mathrm{CO}_{D_1'}(x) + (2^m - \#\mathrm{CO}_{D_2'}(x))$. By the definition of the PADAND gate, the claim follows.

The number of variables in C_n is $m+1$. Therefore, $\#\mathrm{CO}_{C_n}(x) - \#\mathrm{CO}_{\overline{C_n}}(x) = 2 \cdot \#\mathrm{CO}_{C_n}(x) - 2^{m+1}$. By the previous claim, this is $2f(x)$. $\square$

5.3 Random Classes via Semantical Counting

Another advantage of semantical counting circuits is that probabilistic complexity classes can easily be defined in this model. The definition is analogous to the definition of probabilistic classes based on Turing machines' computation: for a given input, we will count the fraction of all settings for the counting variables which make the circuit accept. We will define now the usual types of probabilistic counting classes in our model and compare them to the existing definitions. For a counting circuit C we define $\mathrm{Pr}_{\mathrm{CO}}(C(x))$ by:

$$\mathrm{Pr}_{\mathrm{CO}}(C(x)) = \begin{cases} C(x) & \text{if } \mathrm{CV}(C) = \emptyset, \\ \#\{v \mid C(x,v) = 1\}/2^{|\mathrm{CV}(C)|} & \text{if } \mathrm{CV}(C) \neq \emptyset. \end{cases}$$

Let now C be one of the classes AC^0 or NC^1. Then PC$_{\mathrm{CO}}$ is the family of languages for which there exists a uniform C family of counting circuits (C_n) such that $x \in L$ iff $\mathrm{Pr}_{\mathrm{CO}}(C_{|x|}(x)) > 1/2$. Similarly BPC$_{\mathrm{CO}}$ is the family of languages for which there exists a uniform C family of counting circuits (C_n) such that $x \in L$ iff $\mathrm{Pr}_{\mathrm{CO}}(C_{|x|}(x)) > 1/2 + \epsilon$ for some constant $\epsilon > 0$. Finally, RC$_{\mathrm{CO}}$ is the family of languages for which there exists a uniform C family of counting circuits (C_n) such that if $x\ 2\ L$ then $\mathrm{Pr}_{\mathrm{CO}}(C_{jxj}(x))$ $1{=}2$ and if $x\ 62\ L$ then $\mathrm{Pr}_{\mathrm{CO}}(C_{jxj}(x)) = 0$.

Let us recall that PC was de ned [AAD97, CMTV98] as the family of languages L for which there exists a function f in GapC such that $x\ 2\ L$ i $f(x) > 0$. Theorem 8 implies that these de nitions coincide with ours.

Bounded error and one sided error circuit based probabilistic complexity classes were de ned in the literature for the classes in the AC and NC hierarchy [Weg87, Joh90, Coo85]. These are semantical de nitions in our terminology, but unlike in our case, no special restriction is put on the way counting variables are introduced. To be more precise, let a *probabilistic circuit family* (C_n) as a uniform family of circuits where the circuits have standard and probabilistic input variables and the number of probabilistic input variables is polynomially related to the number of input variables. For any input x, the probability that such a family accepts x is the fraction of assignments for the probabilistic variables which make the circuit C_{jxj} accept x. Then the usual de nition of BPC and RC is similar to that of BPC$_{\mathrm{CO}}$ and RC$_{\mathrm{CO}}$ except that probabilistic circuit families and not counting circuit families are used in the de nition.

The robustness of our de nitions is underlined by the fact that the bounded error (respectively one-sided error) probabilistic class de ned via constant depth and polynomial size branching programs lies between the classes BPAC$^0_{\mathrm{CO}}$ and BPNC$^1_{\mathrm{CO}}$ (respectively RAC$^0_{\mathrm{CO}}$ and RNC$^1_{\mathrm{CO}}$). This follows from the inclusions #AC$^0_{\mathrm{CO}}$ #BR #NC$^1_{\mathrm{CO}}$, and from the fact that counting branching programs are also de ned semantically.

As we mentioned already, it is known [SST95, VW96] that if PAC^0 is de- ned via probabilistic and not counting circuit families, then it is equal to PP. Therefore, it is natural to ask what happens in the other two error models: is BPC$_{\mathrm{CO}}$ = BPC and is RC$_{\mathrm{CO}}$ = RC? If not, then we think that since branching programs form a natural model for de ning low level probabilistic complexity classes, the above result indicates that counting circuits might constitute the basis of the \right" de nition.

6 Conclusion

Circuit based counting functions and probabilistic classes can be de ned semantically via counting circuits families. These circuits contain additional counting variables whose appearances are restricted. When these de nitions are equivalent to the syntactical ones, we can rightly consider the classes robust. In the opposite case, we think that this is a serious reason for reconsidering the de nitions.

Let us say a word about the restrictions we put on the counting variables. They are essentially twofold: rstly they can not appear in several subcircuits of an AND gate and secondly they can be introduced only via the SELECT and PADAND gates. Of these restrictions, the second one is purely technical. Indeed, any appearance of a counting variable u can be replaced by a subcircuit SELECT$(0; 1; u)$ without changing the counting function computed by the circuit. On the other hand, the rst one is essential: without it, #AC^0 would be equal to #P.

References

[AAB$^+$99] E. Allender, A. Ambainis, D.M. Barrington, S. Datta, and H. LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, pages 149{158, 1999.

[AAD97] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 134{148, 1997.

[ABL98] A. Ambainis, D.M. Barrington, and H. LêThanh. On counting AC^0 circuits with negated constants. In *Proceedings of the 23th ACM Symposium on Mathematical Foundations of Computer Science*, pages 409{417, 1998.

[AJ93] Alvarez and Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3{30, 1993.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO*, 30:1{21, 1996.

[Bar89] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150{164, 1989.

[CMTV98] H. Caussinus, P. McKenzie, D. Therien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200{212, 1998.

[Coo85] S.A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2{22, 1985.

[Dam91] C. Damm. $DET = L^{\#L}$. *Informatik-Preprint, Fachbereich Informatik der Humboldt-Universität zu Berlin 8*, 1991.

[FFK94] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-de nable counting classes. *Journal of Computer and System Sciences*, 48(1):116{148, 1994.

[Joh90] D.S. Johnson. A catalog of complexity classes. *Handbook of Theoretical Computer Science*, A:67{161, 1990.

[MV97] M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, December 1997.

[SST95] S. Saluja, K. V. Subrahmanyam, and M. N. Thakur. Descriptive complexity of #P functions. *Journal of Computer and System Sciences*, 50(3):493{505, 1995.

[ST98] M. Santha and S. Tan. Verifying the determinant in parallel. *Computational Complexity*, 7:128{151, 1998.

[Tod91] S. Toda. Counting problems computationally equivalent to computing the determinant. *Tech. Rep. CSIM 91-07*, 1991.

[Val79] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189{201, 1979.

[Val92] L.G. Valiant. Why is boolean complexity theory di cult? *London Mathematical Society Lecture Notes Series*, 16(9), 1992.

[Ven92] H. Venkateswaran. Circuit de nitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:665{670, 1992.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the 6th Annual IEEE Conference on Structure in Complexity Theory*, pages 270{284, 1991.

[VW96] H. Vollmer and K.W. Wagner. Recursion theoretic characterizations of complexity classes of counting functions. *Theoretical Computer Science*, 163(1{2):245{258, 1996.

[Weg87] I. Wegener. *The complexity of boolean functions*. Wiley - Teubner series in computer science, 1987.

[Yam96] T. Yamakami. Uniform AC^0 counting circuits. *Manuscript*, 1996.

The Hardness of Placing Street Names in a Manhattan Type Map[⋆]

Sebastian Seibert and Walter Unger

Dept. of Computer Science I (Algorithms and Complexity), RWTH Aachen,
Ahornstra e 55, 52056 Aachen, Germany
fseibert,quaxg@i1.informatik.rwth-aachen.de

Abstract. Map labeling is a classical key problem. The interest in this problem has grown over the last years, because of the need to churn out di erent types of maps from a growing and altering set of data. We will show that the problem of placing street names without conflicts in a rectangular grid of streets is NP-complete and APX-hard. This is the rst result of this type in this area. Further importance of this result arises from the fact that the considered problem is a simple one. Each row and column of the rectangular grid of streets contains just one street and the corresponding name may be placed anywhere in that line.

1 Introduction and De nitions

The street layout of some modern cities planned on a drawing table are often right angled. In a drawing of such a map the street names should be placed without conflict. Thus each name should be drawn within the rectangular area without splitting or conflicting with other names. See Figure 1 for an example. The names are indicated by simple lines.

A *map* consists of an $N_h \quad N_v$ grid with N_h columns and N_v rows. A *horizontal line*, i.e. a horizontal street name, is given by the pair $(i; l)$ where i $(1 \quad i \quad N_v)$ indicates the row of the street and l $(1 \quad l \quad N_h)$ the length of that name. The *vertical lines* are given in a similar way, by indicating their column and height.

A *placement* in a map assigns to every line (street name) a position to place the rst character in. If the rst character of a horizontal line $(i; l)$ is placed in column s $(1 \quad s \quad N_h - l + 1)$, then the name will occupy the following space in the grid: $f(j; i) \, j \, s \quad j \quad s + l - 1g$. Vertical lines are placed analogously. A *conflict* in a placement is a position occupied by both, a vertical and a horizontal line.

Given a map and a set of lines to be placed in the map, StrP denotes the problem to decide whether the given lines may be placed conflict-free. We will show that this problem is NP-complete. Max-StrP is the problem to nd a placement that maximizes the number of lines placed without conflict, which will

? Supported by DFG Project HR 14/5-1 \Zur Klassi zierung der Klasse praktisch lösbarer algorithmischer Aufgaben"

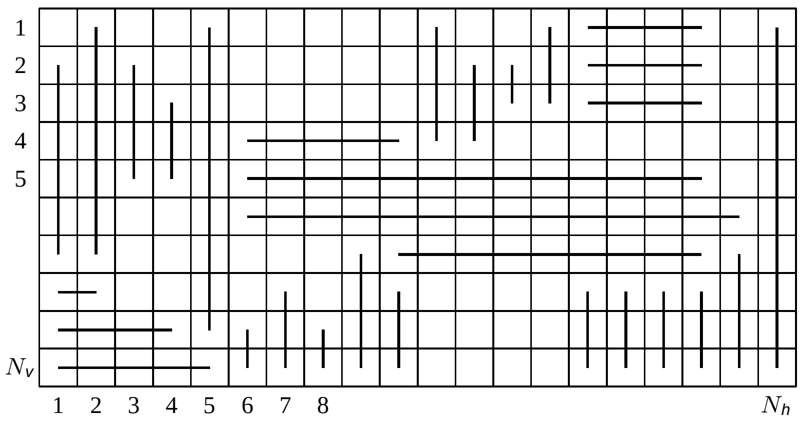

Fig. 1. A map with street names

be shown to be APX-hard below. An overview on approximation problems and hardness of approximation proofs can be found in [Ho96, MPS98].

Map labeling problems have been investigated intensively in the last twenty years. Most results are in the eld of heuristic and practical implementation [WS99]. The known complexity results are about restricted versions of the map labeling problem. In [KI88, FW91, MS91, KR92, IL97], NP-completeness results are presented for the case that each label has to be attached to a xed point. The type of label and alignment varies in these papers. For other models of the map labeling problem see [WS99].

Note that this type of map labeling was motivated by practical applications and introduced in [NW99]. There are algorithms given for solving StrP which run for some special cases in polynomial time and perform reasonable in practical applications. With this respect, the APX-hardness is even more surprising. For the harder problem to place lines on a cylindrical map, there is also a proof of NP-completeness in [NW99] which relies essentially on the cylindrical structure.

Note that our notation di ers from [NW99] where for instance a vector of street-lengths for all rows and columns is given as input. But for showing APX-hardness by a reduction from 3SAT, we need to describe e ciently a map of order n^2 n having only $O(n)$ lines to be placed, if n is the size of the formula.

Our reduction is based on the following result. Please note that in the formulae constructed in the proof of that theorem, every variable is used at least twice[1].

[1] This is not surprising at all: variables used at most once in a formula can be assigned a truth value trivially, hence their inclusion would rather make the problem more tractable, not harder.

Theorem 1 (J. Hastad [Ha97]).
For every small $\varepsilon > 0$, the problem to distinguish 3SAT *instances that can be satisfied from those where at most a fraction of* $7/8 + \varepsilon$ *of the clauses can be satisfied at the same time,* $(1, \frac{7}{8} + \varepsilon)-$3SAT *for short, is NP-hard.*

The construction and proof of NP-completeness of StrP will be presented in Section 2. Section 3 contains the proof for the APX -completeness of Max-StrP and Section 4 gives the conclusions.

2 Map Construction and NP-Hardness

In this section, we give a reduction from 3SAT to StrP which we use for showing NP-hardness as well as APX-hardness. The proof of NP-hardness comes immediately with the construction. For the proof of APX-hardness of StrP only a small construction detail has to be added. Thus we give in this section the full construction, and we prove the APX-hardness of StrP in the next section.

Assume we have a 3SAT formula Φ consisting of clauses $c_1, \ldots, c_l$ over variables $x_1, \ldots, x_m$. Each clause c_i is of the form $z_{i,1} \vee z_{i,2} \vee z_{i,3}$, the $z_{i,j}$ being from $\{x_1, \ldots, x_m\} \cup \{\overline{x_1}, \ldots, \overline{x_m}\}$. We assume w.l.o.g. that each variable occurs at least twice.

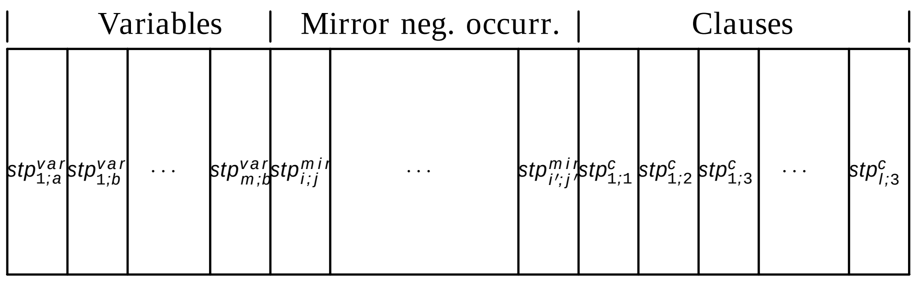

Fig. 2. The outline of the map

First we give an informal description of the proof idea. We want to construct a map M_Φ out of Φ. Let $n = 3l + m$. M_Φ will have height $N_v = 14n$ and width $N_h \approx 36n^2$. It will be split into several vertical stripes, which means that there are several vertical lines of full height N_v. For clear reference, we use names for the stripes. The general picture of M_Φ is shown in Figure 2.

It consists of three groups of vertical stripes. To the left, we have a pair of vertical stripes for each variable. Here, the placement of lines corresponds to assigning a certain value to the respective variable. The rightmost part contains for each clause a triple of vertical stripes such that a non-conflicting placement of all lines in these stripes can be made only if there is a clause satisfying setting of at least one of its variables represented in the leftmost part. The middle part, called \mirror negative occurrences" is necessary to connect the other two properly. It will be explained below.

To ease the description we will just de ne the width of these stripes and further lines added within the strips. Thus the position of the separating vertical lines will be implicitly de ned. The width of these stripes varies between $6n$ and $4n - l > 3n$. The position of vertical lines put within such a stripe will be given relative to the separating vertical lines. Let *stp* be a stripe of width w surrounded by vertical lines at position c and $c + w + 1$. If we say a vertical line *is put at position i (1 i w) in stripe stp,* this line will be put in column $c + i$ of the whole map.

To move the information around in M , our main tool are horizontal lines which t into the above mentioned stripes. These horizontal lines will be ordered (from top and bottom rows towards the middle) according to their length. A line of width $6n - j$ will be in row $n + j$ or $N_v - n - j$, where 0 $j < 3n$. The reason for leaving the top and bottom n rows unused will become clear in Section 3. We denote by $H(j)$ a horizontal line of width $6n - j$ put in row $n + j$ and by $\overline{H}(j)$ a horizontal line of width $6n - j$ put in row $N_v - n - j$.

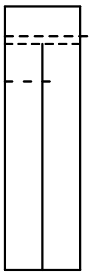

Fig. 3. A vertical stripe

In the middle of a typical vertical stripe of width $6n - j$, we will put a vertical line of height $N_v - 1 - (n + j)$. Consequently, either $H(j)$ or $\overline{H}(j)$ can be placed in this stripe, by placing the middle vertical line either at the lowest position, opening row $n + j$ for $H(j)$, or at the highest position, opening row $N_v - n - j + 1$ for $\overline{H}(j)$. Furthermore, no other horizontal line $H(j^0)$ or $\overline{H}(j^0)$ (0 $j^0 < 3n; j^0 \neq j$) can be placed in that stripe. Either it is wider than the whole stripe, or if it is smaller, it will be in a row which is already blocked by the middle vertical line, see Figure 3.

There will be some exception where we may place several horizontal lines in the same stripe. But as we will see, the overall principle will remain una ected.

The details of the construction are shown in Figure 4. Here we have solid lines depicting lines drawn in their de nite position or in one of their only two possibly non-conflicting positions. These positions are always limited to the respective vertical stripe. Dashed lines are used for the horizontal lines which are the essential tool to connect the di erent parts of the map. They have always two possibly non-conflicting positions in two di erent parts of the map. For some of them, you can see both positions in Figure 4. Finally, there is the dotted line shown in all of its three possibly non-conflicting positions in the rightmost part.

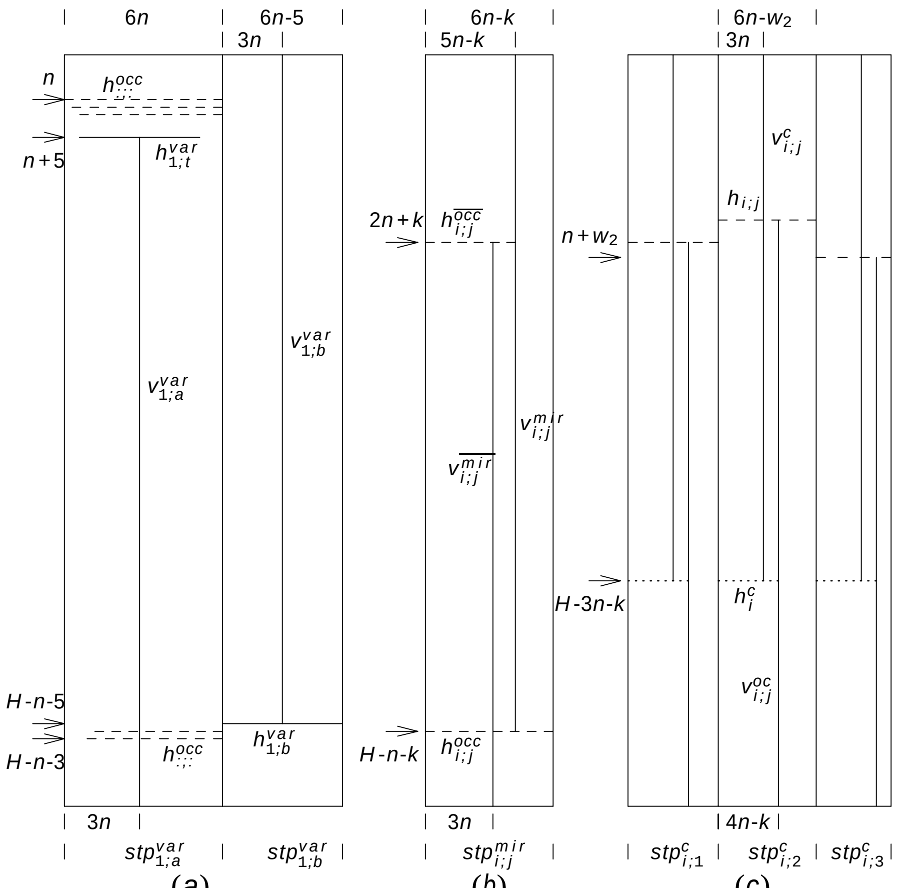

Fig. 4. Components of the map

The Variable Part

More precisely, we start constructing M by building a pair of vertical stripes $stp^{var}_{i;a}; stp^{var}_{i;b}$ as in Figure 4 (a) for each variable x_i (1 i m). Assume x_i occurs s_i times in . Furthermore, let $t_i = i + \sum_{j=1}^{i} s_j$, and $t_0 = 0$. Stripe $stp^{var}_{i;a}$ is set to width $w_{i;a} = 6n - t_{i-1}$ and the width of stripe $stp^{var}_{i;b}$ is $w_{i;b} = 6n - t_i + 1 = 6n - t_{i-1} - s_i$. You will see that $i = 1$ and $s_i = 5$ gives a picture as in Figure 4 (a). Note that in Figure 4, numbers above and below the map give the amount of free space between vertical lines whereas numbers to the left mark the row a horizontal line is put in.

Now we put vertical lines $v_{i;a}^{var}$ and $v_{i;b}^{var}$ in the middle of the stripes $stp_{i;a}^{var}$ and $stp_{i;b}^{var}$ to prevent unwanted placement of horizontal lines. Since all we need is that at most $3n$ columns are left free to either side (all horizontal lines will be wider), it has only to be roughly the middle. Thus we put vertical lines of height $N_v - n - t_i$ in both stripes at the respective position $3n + 1$.

These are complemented by horizontal lines $h_{i;t}^{var} = H(t_i - 1)$ and $h_{i;b}^{var} = \overline{H}(t_i - 1)$. Together, these lines construct a switch as depicted in Figure 5.

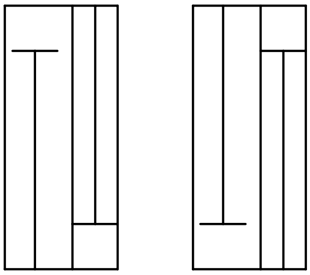
(a) (b)

Fig. 5. Variable switch

Our general principle of stripe construction explained above assures that the horizontal line $h_{i;t}^{var}; h_{i;b}^{var}$ must not be placed elsewhere without conflict. Thus, Figure 5 shows the essentially only two possible non-conflicting placements of the lines built so far. Here, it is unimportant which placement exactly a horizontal line $H(t_i - 1)$ or $\overline{H}(t_i - 1)$ gets in the left stripe $stp_{i;a}^{var}$.

Next, we add a horizontal line for each occurrence of the variable x_i. If $z_{j;j'}$ contains the kth occurrence of x_i in , then $h_{j;j'}^{occ}$ represents that occurrence. If $z_{j;j'} = x_i$ then $h_{j;j'}^{occ} = H(t_{i-1} + k)$, on the other hand, if $z_{j;j'} = \overline{x_i}$, then $h_{j;j'}^{occ} = \overline{H}(t_{i-1} + k)$. These are the dashed lines of Figure 4 (a).

If the switch is placed as in Figure 5 (a) all lines corresponding to positive occurrences of x_i can be placed without conflict in stripe $stp_{i;a}^{var}$. The lines corresponding to negative occurrences cannot be placed without conflict in the switch since stripe $stp_{i;a}^{var}$ is blocked and the stripe $stp_{i;b}^{var}$ is narrower than all of these lines. This placement will correspond to setting $x_i = 1$. Similarly, the other possible placement of the switch corresponds to setting $x_i = 0$.

We will end up using just $n = 3l + m$ di erent widths of horizontal lines, from $6n$ to $5n + 1$. Thus, the next step can be started by using width $5n$.

The Middle Part

The aim of the middle part is to mirror the horizontal lines representing the negative occurrences of variables into the upper half of the map. This is done by using a stripe $stp^{mir}_{i,j}$ as depicted in Figure 4 (b). Assume the occurrence under consideration is $z_{i,j} = \overline{x_u}$.

The new stripe $stp^{mir}_{i,j}$ has the width of the horizontal line to be mirrored. Let that line be $h^{\overline{occ}}_{i,j} = \overline{H}(k)$, $0 \leq k < n$. The width of stripe $stp^{mir}_{i,j}$ is $6n - k$. Now we add inside the stripe a vertical line $v^{mir}_{i,j}$ at position $(5n - k) + 1$ of height $N_v - 1 - (n + j))$. This reduces the width of the stripe for all inner rows from $6n - k$ to $5n - k$. Inside this reduced stripe, we put new lines $h^{\overline{occ}}_{i,j} = H(n + k)$ and a vertical line $\overline{v^{mir}_{i,j}}$ at position $3n + 1$ of height $N_v - 1 - (2n + j))$.

Let us look at the idea of this construction. If x_u is set to 0 (i.e. the lines $h^{var}_{u,t}, h^{var}_{u,b}, v^{var}_{u,a}, v^{var}_{u,b}$ are placed accordingly), $h^{occ}_{i,j}$ can be placed in $stp^{var}_{u,a}$. Then $\overline{v^{mir}_{i,j}}$ can be placed in lowest position. This again allows to place $h^{\overline{occ}}_{i,j}$ above it.

If x_u is set to 1, $h^{occ}_{i,j}$ must be placed in stripe $stp^{mir}_{i,j}$. Then both $v^{mir}_{i,j}$ and $\overline{v^{mir}_{i,j}}$ are \pushed upwards" which makes it impossible to place $h^{\overline{occ}}_{i,j}$ in $stp^{mir}_{i,j}$ without conflict.

The whole middle part of the map is made up out of stripes like that. Note that the new horizontal lines are all of different width since so where the original ones. We just reduce the width by n.

Stripes for Clauses

Finally, we go into constructing a triple of stripes $stp^{c}_{i,1}, stp^{c}_{i,2}, stp^{c}_{i,3}$ for each clause c_i. Let $c_i = z_{i,1} \vee z_{i,2} \vee z_{i,3}$, and let $4n - j > 3n$ be a new width, not used before. Consider the horizontal lines $h_{i,1}, h_{i,2}, h_{i,3}$ representing those occurrences in the upper half of rows. That means $h_{i,j} = h^{occ}_{i,j}$ if $z_{i,j}$ is a positive occurrence of a variable, and $h_{i,j} = h^{\overline{occ}}_{i,j}$ if it is a negative occurrence (for $j \in \{1, 2, 3\}$). For speaking about the width of the new stripes, let $h_{i,j} = H(w_j)$ for $j \in \{1, 2, 3\}$.

For each of the three occurrences, we construct a stripe $stp^{c}_{i,1}, i \in \{1, 2, 3\}$ analogously to those described for the middle part, except that we use as new line width for all three stripes the same value $4n - k$. Consequently, only one new horizontal line $h^{c}_{i} = \overline{H}(2n + k)$ is created. The width of stripe $stp^{c}_{i,j}$ is w_j for $j \in \{1, 2, 3\}$.

Each of the three strips will have new vertical lines are $v^{c}_{i,j}$ of height $N_v - 1 - n - w_j$ at position $4n - k + 1$ (reducing the width to $4n - k$) and $v^{\overline{c}}_{i,j}$ of height $N_v - 1 - 3n - k$ at position $3n$ (the \middle" line), see Figure 4 (c).

Now the overall effect of this is as follows. Assume one of the literals $z_{i,j}$ in a clause c_i is $z_{i,j} = x_u$, and x_u is set to 1, represented by placing the lines of the variable switch for x_u as described above. Then the corresponding horizontal line $h_{i,j} = h^{occ}_{i,j}$ can be placed in $stp^{var}_{u,a}$. This leaves free the place of that line in the corresponding stripe $stp^{c}_{i,j}$ of the clause. The vertical lines $v^{c}_{i,j}, v^{\overline{c}}_{i,j}$ of that stripe can be placed in highest position, and finally, the unique horizontal line

h_i^c can be placed in that stripe. Similarly, assume $z_{i;j} = \overline{x_u}$, and x_u is set to 1, represented by placing the lines of the variable switch for x_u as described above. Then $h_{i;j}^{occ}$ can be placed in $stp_{u;a}^{var}$, and $h_{i;j} = h_{i;j}^{\overline{occ}}$ can be placed in $stp_{i;j}^{mir}$ after placing $v_{i;j}^{mir}$ and $\overline{v_{i;j}^{mir}}$ in lowest position. Again, this leaves free the place of line $h_{i;j}$ in the corresponding stripe $stp_{i;j}^c$ of the clause, and nally, h_i^c can be placed in that stripe.

If on the other hand none of the three variables is set to ful ll the clause, that causes all three horizontal lines $h_{i;1}; h_{i;2}; h_{i;3}$ to be placed in their respective stripes $stp_{i;1}^c; stp_{i;2}^c; stp_{i;3}^c$. But this \pushes down" the vertical lines $v_{i;1}^c; v_{i;1}^{\overline{c}}; v_{i;2}^c; v_{i;2}^{\overline{c}}; v_{i;3}^c; v_{i;3}^{\overline{c}}$ which results in the impossibility to place the clause representing line h_i^c conflict free.

Overall, each non-satis ed clause corresponds to an unavoidable conflict. So far, we have shown our rst result.

Theorem 2. StrP *is NP-hard* ⊓⊔

3 A Bound of Approximability

In this section we will show that the previous construction can be used to obtain thresholds on approximating StrP.

Theorem 3. *For every small " > 0, the problem to distinguish* StrP *instances that can be placed conflict-free from those where at most a fraction of* $\frac{223}{224} + "$ *of the lines can be placed without conflict,* $(1; \frac{223}{224} + ") -$ StrP *for short, is NP-hard.*

Corollary 1. *For every small " > 0, there is no polynomial time* $(\frac{224}{223} - ")$*-approximation algorithm for* Max-StrP *unless P=NP, i.e.* Max-StrP *is APX-hard.*

Proof (of Theorem 3). In view of Theorem 1, it is su cient to show that the above construction satis es the following claims.

1. From a formula   containing l clauses, the above construction yields, in polynomial time, a map M containing at most $28l$ lines (if every variable is used at least twice).
2. There exists a polynomial time procedure P_a that works on a map M as follows. Given a placement p where m *horizontal* lines have conflicts, P_a outputs a placement p^0 where at most m horizontal lines have conflicts, such that each has only one conflict. Moreover, all horizontal lines with conflicts are of the type h_i^c constructed in the last part of the above construction.
3. There exists a polynomial time procedure P_b that works on a map M as follows. Given a placement p^0 as generated by P_a with m conflicts, P_b generates an assignment to the variables of   such that at most m clauses of   are not satis ed.

Then, assuming we would have an algorithm A deciding $(1; \frac{223}{224} + '') - \mathsf{StrP}$ in polynomial time, we could get one deciding $(1; \frac{7}{8} + 28'') - \mathsf{3SAT}$ in polynomial time as follows. (Note that we do not need the fact that P_a and P_b are e cient.)

Given $(1; \frac{7}{8} + 28'') - \mathsf{3SAT}$ instance with l clauses, construct M and apply the assumed decision algorithm.

If is satis able, there is a conflict-free placement in M as we have already seen in Section 2.

If on the other hand at most a fraction $\frac{7}{8} + 28''$ of the clauses of where satis able, we look at M . Assume p to be a placement where a maximal number of lines in M are placed without conflict. Let m be the number of horizontal lines having a conflict under placement p. That is, the fraction of lines which could be placed without conflict in M is at most $\frac{28l-m}{28l}$.

Now we apply procedures P_a and P_b, getting an assignment satisfying at least $l - m$ out of l clauses in . By assumption about , we have

$$\frac{l-m}{l} \quad \frac{7}{8} + 28''; \text{ that is, } m \quad \frac{1}{8} - 28'' \quad l;$$

and hence

$$\frac{28l-m}{28l} \quad \frac{28l - \left(\frac{1}{8} - 28''\right) l}{28l} = \frac{223}{224} + '':$$

Thus the result of algorithm A would decide $(1; \frac{7}{8} + 28'') - \mathsf{3SAT}$, in contradiction to Theorem 1.

It remains to prove the above claims.

1. As mentioned in Section 2, there are at most $\frac{3}{2}l$ variables if we assume every variable to be used at least twice. Furthermore, we have exactly $3l$ occurrences of variables and l clauses.
   Constructing the leftmost part of M , we have invented 6 lines per variable and 1 per occurrence which gives at most $12l$ lines. In the middle part, we need 4 new lines per negative occurrence, that is at most $6l$ lines (remember that at most half of the occurrences are negative). Finally, in the rightmost part of M , we use 9 new vertical and one new horizontal line per clause, being $10l$ new lines.
   Overall, there are at most $28l$ lines to be placed in M .
2. We use the names of the lines and stripes as given in Section 2.
   The basic idea of P_a is that each horizontal line is best placed in one of the vertical stripes \made for it". More precisely, P_a works by modifying p as follows.
   (a) For each variable x_u, place the lines $v^{var}_{u;a}; v^{var}_{u;b}; h^{var}_{u;b}; h^{var}_{u;t}$ in one of the two possible ways they may be placed without conflict among each other within the two stripes $stp^{var}_{u;a}; stp^{var}_{u;b}$ for x_u. Which one of the two possibilities is taken, is decided in such a way that the minimal number of conflicts between $v^{var}_{u;a}$ and those lines $h^{occ}_{i;j}$ (placed as in p) occurs where $z_{i;j}$ is an occurrence of x_u, and $h^{occ}_{i;j}$ is placed completely within stripe $stp^{var}_{u;a}$.

(b) Place every line $h^{occ}_{i;j}$ which still has a conflict in stripe $stp^{c}_{i;j}$ if $z_{i;j}$ is a positive occurrence, and in $stp^{mir}_{i;j}$ if $z_{i;j}$ is a negative occurrence.

(c) For each negative occurrence $z_{i;j}$, place $v^{mir}_{i;j}$ and $v^{\overline{mir}}_{i;j}$ in their highest position if $h^{occ}_{i;j}$ is placed in $stp^{mir}_{i;j}$, and in lowest position otherwise.

(d) If $v^{mir}_{i;j}; v^{\overline{mir}}_{i;j}$ are placed in highest position, place $h^{\overline{occ}}_{i;j}$ in $stp^{c}_{i;j}$. Otherwise, place $h^{\overline{occ}}_{i;j}$ in $stp^{mir}_{i;j}$.

(e) If $h_{i;j}$ is placed in stripe $stp^{c}_{i;j}$, place $v^{c}_{i;j}$ and $v^{\overline{c}}_{i;j}$ in their lowest position, otherwise in their highest position.

(f) It h^{c}_{i} still has a conflict, place it in any of the stripes $stp^{mir}_{i;1}$, $stp^{mir}_{i;2}$, or $stp^{mir}_{i;3}$.

The crucial point is that none of these steps increases the number of horizontal lines having conflicts. We check this step by step.

(a) First, $h^{var}_{u;b}; h^{var}_{u;t}$ are placed without conflict here. Secondly, due to the general principle of the construction, all other horizontal lines than those $h^{occ}_{i;j}$ representing occurrences of x_u cannot be placed without conflict within the stripes for x_u. And those representing occurrences of x_u may be placed without conflict only in the left stripe. All that remains to show is that the minimal number of conflict between $v^{var}_{u;a}$ and those lines is assumed in one of the extremal positions of $v^{var}_{u;a}$. But now, the fact that the rst and last n rows of M are not used for horizontal lines takes e ect. The height of $v^{var}_{u;a}$ is always at least $N_v - 2n$. Thus it is impossible to have horizontal lines placed both below *and* above $v^{var}_{u;a}$ at the same time. Consequently, from any placement of $v^{var}_{u;a}$, changing it into at least one of its extremal positions is save.

(b) Only horizontal lines already having conflicts are a ected.

(c) Only $h^{occ}_{i;j}$ and $h^{\overline{occ}}_{i;j}$ may be placed in $stp^{mir}_{i;j}$ without conflict. If $h^{occ}_{i;j}$ is present, this step may move a conflict from $h^{occ}_{i;j}$ to $h^{\overline{occ}}_{i;j}$. Otherwise, only a possible conflict of $h^{\overline{occ}}_{i;j}$ is resolved.

(d) Again, only horizontal lines having conflicts are a ected.

(e) Symmetrically to step (d), a conflict may only be moved from $h_{i;j}$ to h^{c}_{i}.

(f) Again, only horizontal lines already having conflicts are a ected.

This guarantees that the number of horizontal lines already having conflicts is not increased.

Moreover, steps (b) and (c) for negative occurrences, resp. (b) and (e) for positive occurrences, assure that lines $h^{occ}_{i;j}$ are placed without conflict. Remember that $h_{i;j} = h^{occ}_{i;j}$ for positive occurrences, and $h_{i;j} = h^{\overline{occ}}_{i;j}$ for negative. Steps (d) and (e) guarantee the same for $h^{\overline{occ}}_{i;j}$.

Finally, the lines h^{c}_{i} are placed in a stripe where they may have a conflict with at most some $v^{\overline{c}}_{i;j}$, i.e. have at most one conflict.

3. In Section 2, we have convinced ourselves that the \variable switches" in the leftmost part of M can be placed without conflict only in two ways, interpretable as setting the corresponding variable to 0 or 1. Since all conflicts are on the rightmost part of M, we can take that interpretation as an assignment to the variables. Also in Section 2, we have seen that this

assignment satis es a clause c_i if the line h^c_i can be placed without conflict. Thus, if there were initially m conflicts present in the placement, at most m clauses will be non-satis ed. ⊓⊔

4 Conclusion

We have presented the rst proofs of NP-completeness and APX-hardness for a simple map labeling problem. An algorithm with an approximation factor of two is easy. Just label either all horizontal or all vertical streets. It remains open to close the gap between both factors. Our results extend easily to the case where some regions of the map may not be used for labels. It is also interesting to look for more extensions.

References

[CMS92] Jon Christensen, Joe Marks, Stuart Shieber: Labeling Point Features on Maps and Diagrams, *Harvard CS*, 1992, TR-25-92.

[FW91] Michael Formann, Frank Wagner: A Packing Problem with Applications to Lettering of Maps, Proc. 7th Annu. ACM Sympos. Comput. Geom., 1991, pp. 281{288.

[Ha97] J. Hastad: Some optimal inapproximability results, in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 1{10.

[Ho96] D. S. Hochbaum (Ed.): *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company 1996.

[IL97] Claudia Iturriaga, Anna Lubiw: NP-Hardness of Some Map Labeling Problems, *University of Waterloo, Canada*, 1997, CS-97-18.

[KI88] T. Kato, H. Imai: The NP-Completeness of the Character Placement Problem of 2 or 3 degrees of freedom, *Record of Joint Conference of Electrical and Electronic Engineers in Kyushu*, 1988, pp. 1138.

[KR92] Donald E. Knuth, Arvind Raghunathan: The Problem of Compatible Representatives, *SIAM J. Discr. Math.*, 1992, Vol. 5, Nr. 3, pp. 422{427.

[MPS98] E. W. Mayr, H. J. Prömel, A. Steger (Eds.): *Lectures on Proof Veri cation and Approximation Algorithms. LNCS* 1967, Springer 1998.

[MS91] Joe Marks, Stuart Shieber: The Computational Complexity of Cartographic Label Placement, *Harvard CS*, 1991, TR-05-91.

[NW99] Gabriele Neyer, Frank Wagner: Labeling Downtown, *Department of Computer Science, ETH Zurich, Switzerland*, TR 324, May 1999

[SW99] Tycho Strijk, Alexander Wol : Labeling Points with Circles, *Institut für Informatik, Freie Universität Berlin*, 1999, B 99-08.

[WS99] Tycho Strijk, Alexander Wol : The Map Labeling Bibliography, *http://www.inf.fu-berlin.de/map-labeling/bibliography*.

Labeling Downtown

Gabriele Neyer[?1] and Frank Wagner[??2]

[1] Institut für Theoretische Informatik,
ETH Zürich, CH-8092 Zürich
neyer@inf.ethz.ch

[2] Transport-, Informatik- und Logistik- Consulting (TLC),
Weilburgerstraße 28, D-60326 Frankfurt am Main and
Institut für Informatik, Freie Universität Berlin, Takustraße 9, D-14195 Berlin
Frank.Wagner@tlc.de

Abstract American cities, especially their central regions usually have a very regular street pattern: We are given a rectangular grid of streets, each street has to be labeled with a name running along its street, such that no two labels overlap. For this restricted but yet realistic case an efficient algorithmic solution for the generally hard labeling problem gets in reach.
The main contribution of this paper is an algorithm that guarantees to solve every solvable instance. In our experiments the running time is polynomial without a single exception. On the other hand the problem was recently shown to be NP-hard.
Finally, we present efficient algorithms for three special cases including the case of having no labels that are more than half the length of their street.

1 Introduction

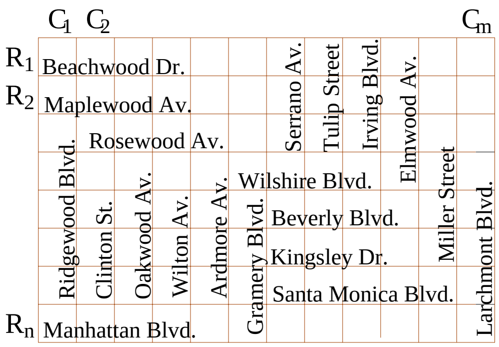

Fig.1. American downtown street map.

The general city map labeling problem is too hard to be automated yet [NH00]. In this paper we focus on the downtown labeling problem, a special case, that was recently shown to be NP-hard [US00].

The clearest way to model it, is to abstract a grid-shaped downtown street pattern into a chess board of adequate size. The names to be placed along their streets we abstract to be tiles that w.l.o.g. span an integer number of fields. A feasible labeling then is a conflict free tiling of the board placing all the labels along their streets.

[?] This work was partially supported by grants from the Swiss Federal Office for Education and Science (Projects ESPRIT IV LTR No. 21957 CGAL and No. 28155 GALIA), and by the Swiss National Science Foundation (grant "Combinatorics and Geometry").

[??] Frank Wagner is a Heisenberg–Stipendiat of the Deutsche Forschungsgemeinschaft, grant Wa1066/1

Our main algorithm is kind of an adaptive backtracking algorithm that is guaranteed to find a solution if there is one. Surprisingly it has an empirically strictly bounded depth of backtracking, namely one, which makes it an empirically efficient algorithm. Given this experience this makes the theoretical analysis of our algorithm even more interesting.

Using results from a well studied family of related problems from Discrete Tomography [Woe96, GG95], we provide a NP-hardness result for a slightly different labeling problem, taking place on a cylinder instead of a rectangle.

We round up the paper by giving efficient solutions to special cases: There is a polynomial algorithm, if

– no label is longer than half of its street length
– all vertical labels are of equal length
– the map is quadratic and each label has one of two label lengths

One general remark that helps to suppress a lot of formal overhead: Often, we only discuss the case of horizontal labels or row labels and avoid the symmetric cases of vertical labels and columns or vice versa.

2 Problem Definition

Let G be a grid consisting of n rows and m columns. Let $R = \{R_1, \dots, R_n\}$ and $C = \{C_1, \dots, C_m\}$ be two sets of labels. The problem is to label the i^{th} row of G with R_i and the j^{th} column of G with C_j such that no two labels overlap. We will represent the grid G by a matrix.

Definition 1 (Label problem (G, R, C, n, m)).

Instance: *Let $G_{n,m}$ be a two dimensional array of size $n \times m$, $G_{i,j} \in \{*, r, c\}$. Let R_i be the label of the i^{th} row and let r_i be the length of label R_i, $1 \leq i \leq n$. Let C_i be the label of the i^{th} column and let c_i be the length of label C_i.*

Problem: *For each row i set r_i consecutive fields of $G_{i,\cdot}$ to r and for each column j set c_j consecutive fields of $G_{\cdot,j}$ to c.*

Of cause no label can be longer than the length of the row or column, respectively.

Initially, we set $G_{i,j} = *$ which denotes that the field is not yet set. Let $[a, b[$ be an interval such that $G_{i,x} \in \{*, r\}$, for $x \in [a, b[$. We say that $G_{i,[a,b[}$ is free for row labeling. Furthermore, this interval has length $b - a$. We also say that $G_{i,\cdot}$ contains two disjoint *intervals* of length at least $\lfloor\frac{b-a}{2}\rfloor$ that are free for row labeling, namely $[a, a + \lfloor\frac{b-a}{2}\rfloor[$ and $[a + \lfloor\frac{b-a}{2}\rfloor, b[$.

3 General Rules

Assume we have a label with length longer than half of its street length. No matter how we position the label on its street, there are some central fields in the street that are always occupied by this label. We therefore can simply mark these fields *occupied*. It is easy to see that these occupied fields can produce more occupied fields. The following rules check whether there is sufficiently large space for each label and determines occupied fields.

Rule 3.1 (Conflict) *Let $I = [a, b[$ be the longest interval of row i that is free for row labeling. If $r_i > b - a$, then row i can not be labeled, since it does not contain enough free space for row labeling. In this case we say that a* conflict *occurred and it follows that the instance is not solvable.*

Rule 3.2 (Large labels) *Let $I = [a, b[$ be the only interval in $G_{i,}$ that is free for feasible row labeling. Observe that the fields that are occupied simultaneously when R_i is positioned leftmost and rightmost in I have to be occupied by R_i no matter where it is placed. These fields we set to r and call them preoccupied.*

Procedure 1 PREPROCESSING (G, R, C, n, m)

```
1) repeat {
2)     $G^0 = G$;
3)     run Rule 3.2 on $(G, R, C, n, m)$ and on $(G^T, C, R, m, n)$;
4)     if Rule 3.1 yields a conflict on $(G, R, C, n, m)$ or on $(G^T, C, R, m, n)$ then
5)         return "conflict";
6)} until $(G = G^0)$;
7) return true;
```

Our PREPROCESSING Procedure 1 iteratively executes the Rules 3.1 and 3.2 until none of them yields a further change to the label problem or a conflict occurs. In the latter case we have that the instance is not solvable. We will spell out special cases where the successful preprocessing implies solvability. Furthermore, the preprocessing underlies the following considerations.

For each unfixed label that is limited to just one interval of at most twice its length or to two intervals of exactly its length we can check whether these labels can be simultaneously positioned without conflicts. This can be done since all possible label positions of these rows and columns can be encoded in a set of 2SAT clauses, the satisfaction of which enforces the existence of a conflict free label positioning of these labels. On the other hand a conflict free label positioning of these labels implies a satisfying truth assignment to the set of clauses. Even, Itai and Shamir [EIS76] proposed a polynomial time algorithm that solves the 2SAT problem in time linear in the number of clauses and variables.

We therefore represent each matrix field $G_{i,j}$ by two boolean variables. We have the boolean variable $G_{i,j} = r$ and its negation $\overline{G_{i,j} = r}$ which means $G_{i,j} \neq r$ or $G_{i,j} \in \{\,, c\}$. As the second variable we have $G_{i,j} = c$ and its negation $\overline{G_{i,j} = c}$ which means $G_{i,j} \neq c$ or $G_{i,j} \in \{\,, r\}$. Of course these two variables are coupled by the relation $(G_{ij} = r) \rightarrow (\overline{G_{ij} = c})$.

Those rows and columns, where the possible label positions are limited to just one interval of at most twice its length or to two intervals of exactly it length, we call *dense*. We now encode all possible label positions of the dense rows and columns in a set of 2SAT clauses the satisfaction of which yields a valid labeling of these rows and columns and vice versa.

Property 1 (Density Property I). Let $G_{i;}$ be a row that contains exactly two maximal intervals each of length r_i that are disjoint and free for feasible row labeling. Let these intervals be $[a;b[$ and $[c;d[$, $1 \quad a < b < c < d \quad n + 1$. Then, a valid labeling exists if and only if the conditions

1. $(G_{i;a} = r) \ \$ \ (G_{i;a+1} = r) \ \$ \ (G_{i;a+2} = r) \ \$ \qquad \$ \ (G_{i;b-1} = r),$
2. $(G_{i;c} = r) \ \$ \ (G_{i;c+1} = r) \ \$ \ (G_{i;c+2} = r) \ \$ \qquad \$ \ (G_{i;d-1} = r),$
3. $(G_{i;a} = r) \nleftrightarrow (G_{i;c} = r)$

are fulfilled.

$(G_{i;a} = r) \ \$ \ (G_{i;a+1} = r)$ can be written as the 2SAT clauses $(\overline{G_{i;a} = r} \ \_ \ G_{i;a+1} = r)$, $(G_{i;a} = r \ \_ \ \overline{G_{i;a+1} = r})$ and since the condition $(G_{i;a} = r) \nleftrightarrow (G_{i;c} = r)$ can be written as $(\overline{G_{i;a} = r} \ \_ \ \overline{G_{i;c} = r})$, $(G_{i;a} = r \ \_ \ G_{i;c} = r)$ it is easy to see that the complete Density Property 1 can be written as a set of 2SAT clauses. The feasible label placements are $(G_{i;a} = r; :::; G_{i;b-1} = r)$ and $(G_{i;c} = r; :::; G_{i;d-1} = r)$.

Property 2 (Density Property II). Let $G_{i;}$ be a row that contains only one maximal interval $[a;b[$ that is free for feasible row labeling, $r_i < b - a \quad 2r_i$. Then, a valid labeling for the row exists if and only if the conditions

1. $(G_{i;a} = r) \ ! \ (G_{i;a+1} = r) \ ! \ (G_{i;a+2} = r) \ ! \qquad ! \ (G_{i;b-r_i-1} = r);$
2. $G_{i;b-r_i} = r; :::; G_{i;a+r_i-1} = r$, and
3. $(G_{i;a} = r) \nleftrightarrow (G_{i;a+r_i} = r); (G_{i;a+1} = r) \nleftrightarrow (G_{i;r_i+1} = r); :::; (G_{i;b-r_i-1} = r) \nleftrightarrow (G_{i;b} = r)$

are fulfilled.

Analogously to the first Density Property, the conditions of the second Density Property can be formulated as a set of 2SAT clauses. All feasible label placements are $(G_{i;a} = r; G_{i;a+1} = r; :::; G_{a+r_i-1} = r)$, $(G_{i;a+1} = r; G_{i;a+2} = r; :::; G_{a+r_i} = r)$, $(G_{i;a+2} = r; G_{i;a+3} = r; :::; G_{a+r_i+1} = r)$, $:::$, $(G_{i;b-r_i} = r; G_{i;b-r_i+1} = r; :::; G_{i;b} = r)$. Note that the properties work analogously for columns.

Theorem 1. *The 2SAT formula of all dense rows and columns can be created in $O(nm)$ time. The 2SAT instance can be solved in $O(nm)$ time.*

Proof: The number of variables is limited by $2nm$. For each dense row we have at most $\frac{3}{2}n$ clauses. Analogously, for each dense column we have at most $\frac{3}{2}m$ clauses. Altogether we have $O(nm)$ clauses. Thus, the satisfiability of the 2SAT instance can be checked in $O(nm)$ time [EIS76]. □

Procedure 2 calls Procedure 1, our *preprocessing*. In case of success, all dense rows and columns are encoded as a set of 2SAT clauses with the aid of Density Property 1 and 2. Then, their solvability is checked e.g. by invoking the 2SAT algorithm of Even, Itai and Shamir [EIS76].

Lemma 1. *The* PREPROCESSING *Procedure 1 and the* DRAW_CONCLUSIONS *Procedure 2 can be implemented to use at most $O(nm(n + m))$ time.*

Procedure 2 DRAW_CONCLUSIONS $(G; R; C; n; m)$

```
1) if PREPROCESSING(G,R,C,n,m) then {
2)     $F$ := the set of 2SAT clauses of the dense rows and column;
3)     if $F$ is satisfiable then return true; }
4) else return false;
```

Proof: The rules only need to be applied to those rows and columns in which an entry was previously set to r or c. A setting of a field $G_{i;j}$ can only cause new settings in row i or column j, which by themselves can again cause new settings. The application of the rules on a row and a column takes time $O(n + m)$. Since at most $2nm$ fields can be set we yield that the preprocessing can be implemented such that its running time is $O(nm(n + m))$. In Theorem 1 we proved that the 2SAT clauses can be generated and checked for solvability in $O(nm)$ time. Thus, we have a worst case time bound of $O(nm(n + m))$. □

Thus, we can solve dense problems:

Theorem 2. *In case that each row and each column of a preprocessed labeling instance* $(G; R; C; n; m)$ *either fulfills the Density Property 1 or 2, Procedure 2* DRAW_CONCLUSIONS *decides if the instance is solvable. In case of solvability we can generate a valid labeling from a truth assignment. The overall running time is bounded by* $O(nm(n + m))$.

4 A General Algorithm

In this section we describe an algorithmic approach with a backtracking component that solves any label problem. Empirically it uses its backtracking ability in a strictly limited way such that its practical runtime stays in the polynomial range. After performing the PREPROCESSING and satisfiability test for dense rows and columns (see Procedure 2 DRAW_CONCLUSIONS), we adaptively generate a tree that encodes all possible label settings of the label problem. Each node in the first level of the search tree corresponds to a possible label setting for the first row label. In the i^{th} level the nodes correspond to the possible label settings for the i^{th} row, depending on the label settings of all predecessor rows. Thus, we have at most m possible positions for a row label and at most n levels. Our algorithm searches for a valid label setting in this tree by traversing the tree, depth-first, generating the children of a node when necessary.

In the algorithm, we preprocess matrix G and check the solvability of the dense rows and columns by invoking Procedure 2 DRAW_CONCLUSIONS. We further mark all these settings permanently. When we branch on a possible label setting for a row, we increase the global timestamp, draw all conclusions this setting has for the other labels by invoking Procedure 2 DRAW_CONCLUSIONS and timestamp each new setting. These consequences can be a limitation on the possible positions of a label or even the impossibility of positioning a label without conflicts. After that, we select one of the newly generated children, increase the timestamp and again timestamp all implications. When a conflict occurs, the process resumes from the deepest of all nodes left behind, namely, from the nearest decision point with unexplored alternatives. We mark all timestamps invalid that correspond to nodes that lie on a deeper level than the decision point. This brings the matrix G into its previous state without storing each state separately. Let the algorithm return a valid label setting for all rows. Since Procedure 1 ensures that each column i contains an interval of length at least c_i that is free for column labeling we can simply label each column and yield a valid label setting. The algorithm is given in Algorithm 1, and in the Procedures 1, 2, and 3.

Algorithm 1 LABEL $(G; R; C; n; m)$

```
1) timestamp := 1;
2) if DRAW_CONCLUSIONS(G,R,C,n,m) yields a conflict
3)     return "not solvable";
4) timestamp each setting;
5) let w be the first row that is not yet labeled;
6) if POSITION_AND_BACKTRACK(w,G,R,C,n,m,timestamp) f
7)     label all columns that are not yet completely labeled;
8)     return G; g
9) else
10)    return "not solvable";
```

Procedure 3 POSITION_AND_BACKTRACK $(w; G; R; C; n; m; timestamp)$

```
1) while there are untested possible positions for label r_w in row w f
2)     local_timestamp := timestamp := timestamp + 1;
3)     label row w with r_w in one of these positions;
4)     timestamp each new setting;
5)     if DRAW_CONCLUSIONS(G,R,C,n,m) then f
6)         timestamp each new setting;
7)         if there is a row w that is not yet labeled f
8)             if POSITION_AND_BACKTRACK(w; G; R; C; n; m) then
9)                 return true;
10)        g
11)        else return true;
12)    g
13)    timestamp each new setting;
14)    mark local_timestamp invalid;
15) g
16) return false;
```

We implemented the backtracking algorithm and tested it on over 10000 randomly generated labeling instances with n and m at most 100. After at most one backtracking step per branch the solvability of any instance was decided. The algorithm is constructive; for each solvable instance a labeling was produced. This makes it reasonable to study the worst case run time behavior of the algorithm with backtracking depth one.

The algorithm behaves in the worst case when each label is positioned first in all places that cause a conflict, before it is positioned in a conflict free place. A row label can be positioned in at most m different places. Each time when a label is positioned the Procedure 2 DRAW_CONCLUSIONS is called, which needs at most $O(nm(n+m))$ time. Thus, the time for positioning a row label is bounded by $O(nm^2(n+m))$ time. Since n rows have to be labeled the backtracking approach with backtracking depth one needs at most $O(n^2m^2(n+m))$ time. If the assumption of limited backtracking behavior does not hold the runtime is exponential.

5 Complexity Status

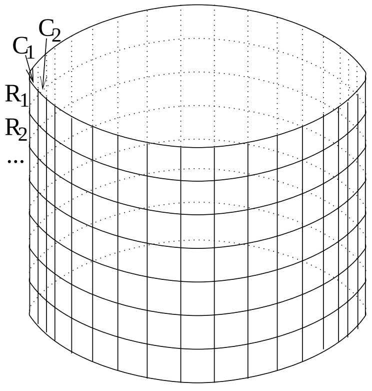

Fig.2. Cylinder Label problem

Although, Unger and Seibert recently proved the NP-completeness of the label problem [US00], we now show that a slight generalization, namely the labeling of a cylinder shaped downtown, is NP-hard. The reason is that our reduction could be helpful in understanding the complexity of the original problem. In addition it is quite intuitive and much shorter than that from Unger and Seibert. Instead of labeling an array we now label a cylinder consisting of n cyclic rows and m columns. Figure 2 shows an example of a cylinder instance. We show that this problem is NP-complete by reducing a version of the *Three Partition* problem to it. Our proof is similar to a NP-completeness proof of Woeginger [Woe96] about the reconstruction of polyominoes from their orthogonal projections. Woeginger showed that the reconstruction of a two-dimensional pattern from its two orthogonal projections H and V is NP-complete when the pattern has to be horizontally and vertically convex. This and other related problems, also discussed in [Woe96] show up in the area of discrete tomography.

Definition 2 (Cylinder Label problem (Z, R, C, n, m)).

Instance: *Let $Z_{n,m}$ be a cylinder consisting of n cyclic rows and m columns. Let R_i be the label of the i^{th} row and let r_i be the length of label R_i, $1 \le i \le n$. Let C_i be the label of the i^{th} column and let c_i be the length of label C_i, $1 \le i \le m$.*

Problem: *For each row i set r_i consecutive fields of $Z_{i,\cdot}$ to r, for each column j set c_j consecutive fields of $Z_{\cdot,j}$ to c.*

Our reduction is done from the following version of the NP-complete Three Partition problem [GJ79].

Problem 1 (Three Partition).

Instance: Positive integers $a_1, \ldots, a_{3k}$ that are encoded in unary and that fulfill the two conditions (i) $\sum_{i=1}^{3k} a_i = k(2B+1)$ for some integer B, and $(ii) (2B+1)/4 < a_i < (2B+1)/2$ for $1 \le i \le 3k$.

Problem: Does there exist a partition of $a_1, \ldots, a_{3k}$ into k triples such that the elements of every triple add up to exactly $2B + 1$?

Theorem 3. *The Cylinder Label problem is NP-complete.*

Proof: Cylinder Label problem $\in NP$: The Cylinder Label problem is in NP since it is easy to check whether a given solution solves the problem or not.

Transformation: Now let an instance of Tree Partition be given. From this instance we construct a Cylinder Label problem consisting of $n = k(2B + 2)$ rows and $m = 3k$ columns. The vector r defining the row label length is of the form:

$$(m, \underbrace{m-1, \ldots, m-1}_{(2B+1)\text{-}times}, m, \underbrace{m-1, \ldots, m-1}_{(2B+1)\text{-}times}, \ldots)$$

Since a row label of length m occupies the whole row, those rows with label length m have no free space for column labeling. Therefore the rows with label length m subdivide the rows in k blocks, each containing $2B + 1$ rows each of which has one entry that is free for column labeling when the row is labeled. The vector defining the column label length is of the form:

$$(a_1, a_2, \ldots, a_{3k})$$

The transformation clearly is polynomial.

The Tree Partition instance has a solution $\Leftrightarrow$ the Cylinder Label instance has a solution:
“$\Rightarrow$”: Let $(x_1, y_1, z_1), \ldots, (x_k, y_k, z_k)$ be a partition of $a_1, \ldots, a_{3k}$ into k triples such that $x_i + y_i + z_i = 2B + 1$, $1 \le i \le k$. For each i, $(x_i, y_i, z_i) = (a_f, a_g, a_h)$, for some indices f,g, and h, $1 \le i, f, g, h \le 3k$. We now label the columns f, g, and h among themselves in the i-th block of rows. More precisely, in column f we label the fields $Z_{f,(i-1)(2B+2)+2} = c, \ldots, Z_{f,(i-1)(2B+2)+1+c_{a_f}} = c$. In column g we label the fields $Z_{g,(i-1)(2B+2)+2+c_{a_f}} = c, \ldots, Z_{g,(i-1)(2B+2)+1+c_{a_f}+1+c_{a_g}} = c$. In column h we label the fields $Z_{h,(i-1)(2B+2)+2+c_{a_f}+c_{a_g}} = c, \ldots, Z_{h,(i-1)(2B+2)+1+c_{a_f}+c_{a_g}+c_{a_h}} = c$. It then follows that the rows $j(2B + 2) + 1$ are free for row labeling, for $0 \le j \le k$. Thus, we can label them with their labels of length $3k = m$. All other rows have exactly one entry occupied by a column label. Since the rows are cyclic we can label each of these rows with a label of length $3k - 1$.

“$\Leftarrow$”: Let Z be a solution of the Cylinder Label instance. Each row contains at most one entry that is occupied by a column label. Each column label a_i has length $(2B + 1)/4 < a_i < (2B + 1)/2$, $1 \le i \le 3k$. Therefore, exactly three columns are label in the rows $j(2B + 2) + 2, \ldots, (j + 1)(2B + 2)$, for $0 \le j \le k - 1$. Furthermore the label length of each triple sums up to $3k$ and thus partitions $a_1, \ldots, a_{3k}$ into k triples. Thus solves the Three Partition instance. □

6 Solvable Special Cases

In the following section we derive an $O(nm)$ time algorithm for the special case where no label is longer than half of its street length. We think that this case applies especially to large downtown maps, where the label length is short in respect to the street length. In Section 6.2 we solve the label problem when each vertical label is of equal length. In many American cities the streets in one orientation (e.g. north-south) are simply called *1-st Avenue, 2-nd Avenue,* $\ldots$. These names have all the same label length and thus the label problem can be solved with the algorithm in Section 6.2. Another solvable special case is the following: We are given a quadratic map, where each label has one of two label lengths. Such an instance has a solution if and only if no conflict occurred in the PREPROCESSING Procedure 1. Due to space limitations, the length of the algorithm, and its proof we refer to the technical report of this paper for this special case [NW99]. The algorithms of the last two cases have runtime $O(nm(n+m))$.

6.1 Half Size

Let (G,R,C,n,m) be a label problem. In this section we study the case where each row label has length at most $\lfloor \frac{m}{2} \rfloor$ and each column label has length at most $\lfloor \frac{n}{2} \rfloor$. We show that the label problem is solvable in this case.

Algorithm 2 HALF_SOLUTION (G,R,C,n,m)

```
1) label the rows $1,\ldots,\lfloor \frac{n}{2} \rfloor$ leftmost;
2) label the rows $\lfloor \frac{n}{2} \rfloor + 1 \ldots n$ rightmost;
3) label the columns $1,\ldots,\lfloor \frac{m}{2} \rfloor$ bottommost;
4) label the columns $\lfloor \frac{m}{2} \rfloor + 1,\ldots,m$ topmost;
```

Theorem 4. *Let (G,R,C,n,m) be a label problem. Let $r_i \leq \lfloor \frac{m}{2} \rfloor$ and let $c_i \leq \lfloor \frac{n}{2} \rfloor$. Then, Algorithm 2 computes a solution to Problem 1 in $O(nm)$ time.*

Proof: Take a look at Figure 3.

6.2 Constant Vertical Street Length

In this section we consider the special case of the label problem (G,R,C,n,m) where all column labels have length l. This problem we denote with (G,R,C,n,m,l). We show that we can decide whether the label problem (G,R,C,n,m,l) is solvable or not. We further give a simple algorithm that labels a solvable instance correctly. All results of this section are assignable for the constant row length case.

Theorem 5 (Constant Column Length). *Let (G,R,C,n,m,l) be a label problem with $c_i = l$, $1 \leq i \leq n$. The instance is solvable if and only if no conflict occurred in the* PREPROCESSING *Procedure 1.*

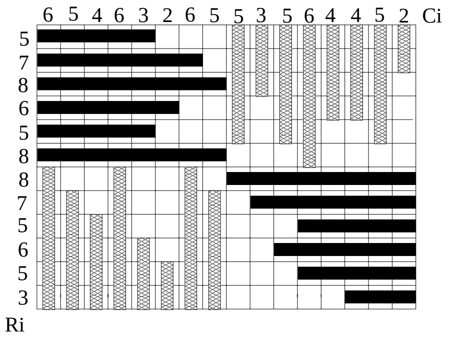

Fig.3. Solution of a typical half size label problem according to Algorithm 2.

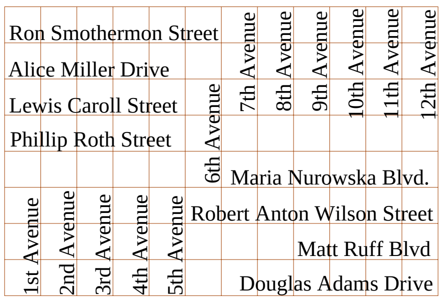

Fig.4. Typical downtown map where the vertical street names have constant length.

We assume that $l \leq \lfloor \frac{m}{2} \rfloor$. Otherwise, row $\lceil \frac{n}{2} \rceil$ contains no fields that are free for row labeling. The next lemma states that the preoccupied fields are symmetric to the vertical central axis of G.

Lemma 2. *Let (G, R, C, n, m, l) be a successfully preprocessed label problem. After the preprocessing each row has the form $[aba]$, where*

$$G_{i,1} = \emptyset, \ldots, G_{i,a} = \emptyset, G_{i,a+1} = x, \ldots, G_{i,a+b} = x, G_{i,a+b+1} = \emptyset, \ldots, G_{i,m} = \emptyset$$

for $x = r$ or $x = c$, $m \geq b \geq 0$ and $2a + b = m$.

Proof: Initially we have $G_{i,j} = \emptyset$ for $1 \leq i \leq n$, $1 \leq j \leq m$. Executing Rule 3.2 on each row i with length $r_i > \frac{m}{2}$ yields $G_{i,m-r_i+1} = r, \ldots, G_{i,r_i} = r$. Thus, all r-entries of G are symmetric to the vertical mid axis of G. Remember that each column has label length l. Therefore, executing Rule 3.2 on each column i yields that for each entry $G_{i,j}$ that is set to c the fields $G_{i,j+1} = c, \ldots, G_{i,m-j+1} = c$, if $1 \leq \frac{m}{2}$; and $G_{i,m-j+1} = c, \ldots, G_{i,j-1} = c$, if $\frac{m}{2} \leq i \leq m$. Therefore, all c-entries of G are symmetric to the central vertical axis of G. Thus, until now each row i has the form $[aba]$, where $G_{i,1} = \emptyset, \ldots, G_{i,a} = \emptyset, G_{i,a+1} = x, \ldots, G_{i,a+b} = x, G_{i,a+b+1} = \emptyset, \ldots, G_{i,m} = \emptyset$ for $x, y \in \{r, c\}$, $b \geq 0$, $2a + b = m$ and $1 \leq i \leq n$. Assume that the repeated execution of Rule 3.2 on row i of form $[aba]$ and $x = c$ does change an entry of $G_{i,\cdot}$. In this case $a < r_i$ and it follows that the instance is not solvable. Therefore, the repeated execution of Rule 3.2 on a column can not change the instance and the lemma follows. □

Lemma 3. *Let (G, R, C, n, m, l) be a successfully preprocessed label problem. Then Algorithm 3 computes a feasible solution to (G, R, C, n, m, l) in $O(nm(n + m))$ time.*

Proof: Since (G, R, C, n, m, l) is preprocessed successively it follows that each column contains an interval of length at least l that is free for column labeling. Assume that after processing steps 2-3 there exists a row i not containing an interval of length r_i that is free for row labeling. We make a case distinction according to the length of R_i:

Algorithm 3 CONSTANT COLUMN LENGTH $(G; R; C; n; m; l)$

```
1) if PREPROCESSING$(G; R; C; n; m)$ $f$
2)     label the columns $1; : : : ; d\frac{m}{2}e - 1$ bottommost;
3)     label the columns $d\frac{m}{2}e; : : : ; m$ topmost;
4)     label the rows $1; : : : ; n$ in the free space;
5) $g$
```

Case $r_i > d\frac{m}{2}e$: We know that the fields $G_{i;m-r_i+1} = r; : : : ; G_{i;r_i} = r$ were set in the preprocessing. Furthermore, Lemma 2 yields that no other entry of $G_{i;}$ was set to c in the preprocessing. Therefore, each column $G_{;j}$ with $1 \quad j < m - r_i + 1$ or $r_i + 1 \quad j \quad m$ contains either one interval of length at least $2l$ that is free for column labeling or two intervals each of length at least l that is free for column labeling. From the symmetry of the label problem and since the column labels of the columns j with $1 \quad j < m - r_i + 1$ are labeled bottom most and the labels j with $r_i + 1 \quad j \quad m$ are labeled top most it follows that either the fields $G_{i;1}; : : : ; G_{i;m-r_i+1}$ are free for row labeling or the fields $G_{i;r_i+1}; : : : ; G_{i;m}$. Thus, $G_{i;}$ contains an interval of length r_i that is free for row labeling. Contradiction.

Case $r_i \quad d\frac{m}{2}e$: Lemma 2 yields that this row has the form $[aba]$ with $G_{i;1} = ; ; : : : ; G_{i;a} = ; ; G_{i;a+1} = c; : : : ; G_{i;a+b} = c; G_{i;a+b+1} = ; ; : : : ; G_{i;m} = ;$, $b \quad 0$ and $2a + b = m$. Since the instance is solvable it follows that $a \quad r_i$. With the same arguments as above it follows that either the fields $G_{i;1}; : : : ; G_{i;r_i}$ are free for row labeling or the fields $G_{i;m-r_i+1}; : : : ; G_{i;m}$ are free for row labeling. Contradiction.

The running time is dominated by the preprocessing and thus $O(nm(n + m))$. □

See Figure 5 and 6 for an example. Figure 4 shows a typical downtown city map in which all vertical streets have the same length.

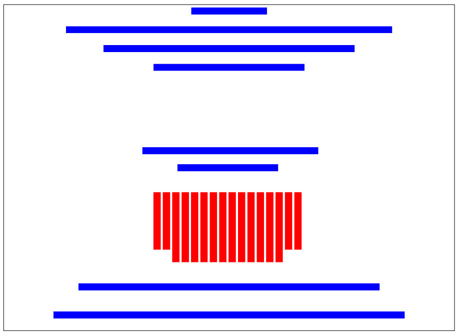
Fig.5. Matrix of a constant column length problem after the successful preprocessing. Entries that are set in the preprocessing are colored black and gray.

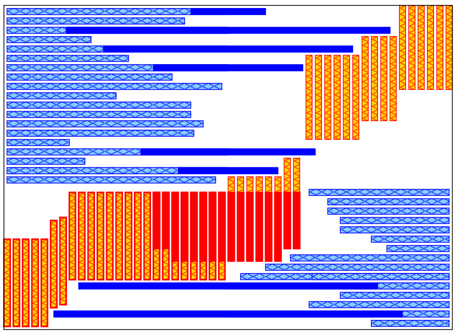
Fig.6. Solution of the label problem of the left figure.

Acknowledgements

The authors would like to thank Peter Widmayer and Wolfram Schlickenrieder for interesting and fruitful discussions.

References

[EIS76] Even S., Itai A., Shamir A.: On the Complexity of Timetable and Multicommodity Flow Problems. SIAM Journal on Computing, **5**(4):691-703, (1976)

[GG95] Gardner R.J., Gritzmann P.: Descrete Tomography: Determination of Finite Sets by X-Rays. Technical Report TR-95-13, Institute of Computer Science, University of Trier, Austria, (1995)

[GJ79] Garey M.R., Johnson D.S.: Computers and Intractibility, A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York, (1979)

[NH00] Neyer G., Hennes H.: Map Labeling with Application to Graph Labeling. GI Forschungsseminar: Zeichnen von Graphen, Teubner Verlag, to appear, (2000)

[NW99] Neyer G., Wagner F.: Labeling Downtown. Technical Report TR 324, ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/3xx/324.ps.gz, Institute of Computer Science, ETH Zurich, Switzerland, (1999)

[US00] Unger W., Seibert S.: The Hardness of Placing Street Names in a Manhattan Type Map. Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC'2000), Lecture Notes in Computer Science, Springer-Verlag, this volume, (2000)

[Woe96] Woeginger G.J.: The Reconstruction of Polyominoes from their Orthogonal Projections. Technical Report SFB-Report 65, TU Graz, Institut für Mathematik, A-8010 Graz, Austria, (1996)

The Online Dial-a-Ride Problem under Reasonable Load[⋆]

Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau

Konrad-Zuse-Zentrum für Informationstechnik Berlin,
Takustr. 7, 14195 Berlin, GERMANY,
fhauptmeier,krumke,rambaug@zib.de

Abstract. In this paper, we analyze algorithms for the online dial-a-ride problem with request sets that ful ll a certain worst-case restriction: roughly speaking, a set of requests for the online dial-a-ride problem is reasonable if the requests that come up in a su ciently large time period can be served in a time period of at most the same length. This new notion is a stability criterion implying that the system is not overloaded. The new concept is used to analyze the online dial-a-ride problem for the minimization of the maximal resp. average flow time. Under reasonable load it is possible to distinguish the performance of two particular algorithms for this problem, which seems to be impossible by means of classical competitive analysis.

1 Introduction

It is a standard assumption in mathematics, computer science, and operations research that problem data are given. However, many aspects of life are online. Decisions have to be made without knowing future events relevant for the current choice. Online problems, such as vehicle routing and control, management of call centers, paging and caching in computer systems, foreign exchange and stock trading, had been around for a long time, but no theoretical framework existed for the analysis of online problems and algorithms.

Meanwhile, competitive analysis has become the standard tool to analyze online-algorithms [4,6]. Often the online algorithm is supposed to serve the requests one at a time, where a next request becomes known when the current request has been served. However, in cases where the requests arrive at certain points in time this model is not su cient. In [3,5] each request in the request sequence has a release time. The sequence is assumed to be in non-decreasing order of release times. This model is sometimes referred to as the *real time model*. A similar approach was used in [1] to investigate the *online dial-a-ride problem* | OlDarp for short | which is the example for the new concept in this paper.

Since in the real time model the release of a new request is triggered by a point in time rather than a decision of the online algorithm we essentially do not

[⋆] Research supported by the German Science Foundation (grant 883/5-2) need a total order on the set of requests. Therefore, for the sake of convenience, we will speak of *request sets* rather than request sequences.

In the problem OlDarp objects are to be transported between points in a given metric space X with the property that for every pair of points $(x; y)\ 2\ X$ there is a path $p : [0; 1]\ \ !\ \ X$ in X with $p(0) = x$ and $p(1) = y$ of length $d(x; y)$. An important special case occurs when X is induced by a weighted graph.

A request consists of the objects to be transported and the corresponding source and target vertex of the transportation request. The requests arrive online and must be handled by a server which starts and ends its work at a designated origin. The server picks up and drops objects at their starts and destinations. It is assumed that neither the release time of the last request nor the number of requests is known in advance.

A feasible solution to an instance of the OlDarp is a schedule of moves (i.e., a sequence of consecutive moves in X together with their starting times) in X so that every request is served and that no request is picked up before its release time. The goal of OlDarp is to nd a feasible solution with \minimal cost", where the notion of \cost" depends on the objective function used. The focus of this paper is the investigation of the notoriously di cult task to minimize the maximal or average flow time online.

Recall that an online-algorithm A is called *c-competitive* if there exists a constant c such that for any request set (or request sequence if we are concerned with the classical online model) the inequality A( ) c OPT( ) holds. Here, X( ) denotes the objective function value of the solution produced by algorithm X on input and OPT denotes an optimal o ine algorithm. Sometimes we are dealing with various objectives at the same time. We then indicate the objective *obj* in the superscript, as in $\mathrm{X}^{obj}(\ )$.

Competitive analysis of OlDarp provided the following (see [1]):

{ There are competitive algorithms (IGNORE and REPLAN, de nitions see below) for the goal of minimizing the *total completion time* of the schedule.
{ For the task of minimizing the *maximal (average) waiting time* or the *maximal (average) flow time* there can be no algorithm with constant competitive ratio. In particular, the algorithms IGNORE and REPLAN have an unbounded competitive ratio.

We do not claim originality for the actual online-algorithms IGNORE and REPLAN; instead we show a new method for their analysis. As the reader will see in the de nitions, both REPLAN and IGNORE are straight-forward online strategies based on the ability to solve the o ine version of the problem to optimality or a constant-factor approximation thereof (with respect to the minimization of the total completion time).

The rst | to the best of our knowledge | occurrence of the strategy IGNORE can be found in the paper by Shmoys, Wein, and Williamson [13]: They show a fairly general result about obtaining competitive algorithms for minimizing the total completion time in machine scheduling problems when the jobs arrive over time: If there is a -approximation algorithm for the o ine version, then this implies the existence of a 2 -competitive algorithm for the online-version, which is essentially the IGNORE strategy. The results from [13] show that IGNORE-type strategies are 2-competitive for a number of online-scheduling problems. The strategy REPLAN is probably folklore; it can be found also under di erent names like REOPT or OPTIMAL.

It should be noted that the corresponding o ine-problems with release times (where all requests are known at the start of the algorithm) are NP-hard to solve for the objective functions of minimizing the average or maximal flow time | it is even NP-hard to nd a solution within a constant factor from the optimum [11]. The o ine problem without release times of minimizing the total completion time is polynomially solvable on special graph classes but NP-hard in general [8,2,7,10].

If we are considering a continuously operating system with continuously arriving requests (i.e., the request set may be in nite) then the total completion time is meaningless. Bottom-line: in this case, the existing positive results cannot be applied and the negative results tell us that we cannot hope for performance guarantees that may be relevant in practice, such as bounds for the maximal or average flow time. In particular, the two algorithms IGNORE and REPLAN cannot be distinguished by classical competitive analysis because it is easy to see that no online-algorithm can have a constant competitiveness ratio with respect to minimizing the maximal or average flow time.

The point here is that we do not know any notion from the literature to describe what a particular set of requests should look like in order to allow for a continuously operating system. In queuing theory this is usually modelled by a stability assumption: the rate of incoming requests is at most the rate of requests served. To the best of our knowledge, so far there has been nothing similar in the existing theory of discrete online-algorithms. Since in many instances we have no exploitable information about the distributions of requests we want to develop a worst-case model rather than a stochastic model for stability of a continuously operating system.

Our idea is to introduce the notion of *-reasonable* request sets. A set of requests is -reasonable if | roughly speaking | requests released during a period of time can be served in time at most . A set of requests R is *reasonable* if there exists a < 1 such that R is -reasonable. That means, for non-reasonable request sets we nd arbitrarily large periods of time where requests are released faster than they can be served | even if the server has the optimal o ine schedule. When a system has only to cope with reasonable request sets, we call this situation *reasonable load*. Section 3 is devoted to the exact mathematical setting of this idea.

We now present our main result on the OlDarp under -reasonable which we prove in Sects. 4 and 5.

Theorem 1. *For the* OlDarp *under -reasonable load,* IGNORE *yields a maximal and an average flow time of at most* 2 *, whereas the maximal and the average flow time of* REPLAN *are unbounded.*

The algorithms IGNORE and REPLAN have to solve a number of o ine instances of OlDarp, minimizing the total completion time, which is in general

NP-hard, as we already remarked. We will show how we can derive results for IGNORE when using an approximate algorithm for solving o ine instances of OlDarp (for approximatation algorithms for o ine instances of OlDarp, refer to [8,2,7,10]). For this we re ne the notion of reasonable request sets again, introducing a second parameter that tells us, how \fault tolerant" the request set is. In other words, the second parameter tells us, how \good" the algorithm has to be to show stable behavior. Again, roughly speaking, a set of requests is (,)-reasonable if requests released during a period of time can be served in time at most = . If = 1, we get the notion of -reasonable as described above. For > 1, the algorithm is allowed to work \sloppily" (e.g., employ approximation algorithms) or have break-downs to an extent measured by and still show a stable behavior.

Note that our performance guarantee is with respect to the \reasonableness" of the input set | not with respect to an optimal o ine solution. One might ask whether IGNORE is competitive with respect to minimizing the maximal or average flow time. This follows trivially from our main result if the length of a single request is bounded from below; we leave it as an exercise for the reader to show that without this assumption there can be no competitive online algorithm for these objective functions | even under reasonable load.

The algorithms under investigation compute o ine locally optimal schedules with respect to the minimization of the total completion time in order to globally minimize the maximal or average flow times. This is of practical relevance because minimizing the total completion time o ine is easier than minimizing the maximal or average flow time o ine (see [11]). It is an open problem whether locally optimal schedules minimizing the maximal or average flow time yield better results. However, then the locally optimal schedules would be much harder to compute. Thus, such algorithms would not be feasible in practice.

2 Preliminaries

Let us rst sketch the problem under consideration. We are given a metric space $(X;d)$. Moreover, there is a special vertex $o\ 2\ X$ (the origin). Requests are triples $r = (t;a;b)$, where a is the start point of a transportation task, b its end point, and t its release time, which is | in this context | the time where r becomes known. A *transportation move* is a quadruple $m = (t;a;b;R)$, where a is the starting point and b the end point, and t the starting time, while R is the set (possibly empty) of requests carried by the move. The *arrival time* of a move is the sum of its starting time and $d(a;b)$. A *(closed) transportation schedule* is a sequence $(m_1;m_2;:::)$ of transportation moves such that

{ the rst move starts in the origin o;
{ the starting point of m_i is the end point of m_{i-1};
{ the starting time of m_i carrying R is no earlier than the maximum of the arrival time of m_i and the release times of all requests in R.
{ the last move ends in the origin o.

An *online-algorithm* for OlDarp has to move a server in X so as to fulfill all released transportation tasks without preemption (i.e., once an object has been picked up it is not allowed to be dropped at any other place than its destination), while it does not know about requests that come up in the future. In order to plan the work of the server, the online-algorithm may maintain a preliminary (closed) transportation schedule for all known requests, according to which it moves the server. A posteriori, the moves of the server induce a complete transportation schedule that may be compared to an offline transportation schedule that is optimal with respect to some objective function (competitive analysis).

We are concerned with the following objective functions:

- The *total completetion time* (or *makespan*) $\mathsf{A}^{comp}(\sigma)$ of the solution produced by algorithm A on a request set σ is the time need by algorithm A to serve all requests in σ.
- The *maximal resp. average flow time* $\mathsf{A}^{maxflow}(\sigma)$ resp. $\mathsf{A}^{flow}(\sigma)$ is the maximal resp. average of the differences between the completion times produced by A and the release times of the requests in σ.

We start with some useful notation.

Definition 1. *The* release time *of a request r is denoted by $t(r)$.*
The offline version *of $r = (t, a, b)$ is the request*

$$r^{offline} := (0, a, b).$$

The offline version *of R is the request set*

$$R^{offline} := \left\{ r^{offline} : r \in R \right\}.$$

An important characteristic of a request set with respect to system load considerations is the time period in which it is released.

Definition 2. *Let R be a finite request set for* OlDarp*. The* release span $\delta(R)$ *of R is defined as*

$$\delta(R) := \max_{r \in R} t(r) - \min_{r \in R} t(r).$$

Provably good algorithms exist for the total completion time and the weighted sum of completion times. How can we make use of these algorithms in order to get performance guarantees for minimizing the maximal (average) waiting (flow) times? We suggest a way of characterizing request sets which we want to consider "reasonable".

3 Reasonable Load

In a continuously operating system we wish to guarantee that work can be accomplished at least as fast as it is presented. In the following we propose a mathematical set-up which models this idea in a worst-case fashion. Since we are always working on nite subsets of the whole request set the request set itself may be in nite, modeling a continuously operating system.

We start by relating the release spans of nite subsets of a request set to the time we need to ful ll the requests.

De nition 3. *Let R be a request set for the* OlDarp. *A weakly monotone function*

$$f\colon \begin{array}{l} \mathbb{R} \;!\; \mathbb{R}; \\ \quad 7! \; f(\;); \end{array}$$

is a load bound *on R if for any $\; 2\, \mathbb{R}$ and any nite subset S of R with (S) the completion time $\mathrm{OPT}^{comp}(S^{o\;\;ine})$ of the optimum schedule for the o ine version $S^{o\;\;ine}$ of S is at most $f(\;)$. In formula:*

$$\mathrm{OPT}^{comp}(S^{o\;\;ine}) \quad f(\;):$$

Remark 1. If the whole request set R is nite then there is always the trivial load bound given by the total completion time of R. For every load bound f we may set $f(0)$ to be the maximum completion time we need for a single request, and nothing better can be achieved.

A stable situation would be characterized by a load bound equal to the identity on $\mathbb{R}$. In that case we would never get more work to do than we can accomplish. If it has a load bound equal to a function $id=\;$, where id is the identity and where $\quad 0$, then measures the tolerance of the request set: An algorithm that is by a factor worse then optimal will still accomplish all the work that it gets. However, we cannot expect that the identity (or any linear function) is a load bound for OlDarp because of the following observation: a request set consisting of one single request has a release span of 0 whereas in general it takes non-zero time to serve this request. In the following de nition we introduce a parameter describing how far a request set is from being load-bounded by the identity.

De nition 4. *A load bound f is* (,)-reasonable *for some $\;;\; 2\,\mathbb{R}$ if*

$$f(\;) \qquad \text{for all}$$

A request set R is (,)-reasonable *if it has a (,)-reasonable load bound. For* $= 1$, *we say that the request set is -reasonable.*

In other words, a load bound is *(,)-reasonable,* if it is bounded from above by $1= \; id(x)$ for all x and by the constant function with value $1=$ otherwise.

Remark 2. If is su ciently small so that all request sets consisting of two or more requests have a release span larger than then the rst-come- rst-serve strategy is good enough to ensure that there are never more than two unserved requests in the system. Hence, the request set does not require scheduling the requests in order to provide for a stable system. (By \stable" we mean that the number of unserved requests in the system does not become arbitrarily large.)

In a sense,   is a measure for the combinatorial di culty of the request set R. Thus, it is natural to ask for performance guarantees for algorithms in terms of this parameter. This is done for the algorithm IGNORE in the next section.

4 Bounds for the Flow Times of IGNORE

We are now in a position to prove bounds for the maximal resp. average flow time in the OlDarp for algorithm IGNORE stated in Theorem 1. We start by recalling the algorithm IGNORE from [1]

De nition 5 (Algorithm IGNORE). *Algorithm* IGNORE *works with an internal buffer. It may assume the following states (initially it is IDLE):*

IDLE *Wait for the next point in time when requests become available. Goto PLAN.*

BUSY *While the current schedule is in work store the upcoming requests in a bu er (\ignore them"). Goto IDLE if the bu er is empty else goto PLAN.*

PLAN *Produce a preliminary transportation schedule for all currently available requests R (taken from the bu er) with (approximately) minimal total completion time $\mathrm{OPT}^{comp}(R^{o\ \ ine})$ for $R^{o\ \ ine}$. (Note: This yields a feasible transportation schedule for R because all requests in R are immediately available.) Goto BUSY.*

We assume that IGNORE solves o ine instances of OlDarp employing a -approximation algorithm. Recall that a -approximation algorithm is a polynomial algorithm that always nds a solution that is at most   times the optimum objective value.

Let us consider the intervals in which IGNORE organizes its work in more detail. The algorithm IGNORE induces a dissection of the time axis IR in the following way: We can assume, w.l.o.g., that the rst set of requests arrives at time 0. Let $\ _0 = 0$, i.e., the point in time where the rst set of requests is released that are processed by IGNORE in its rst schedule. For $i > 0$ let $\ _i$ be the duration of the time period the server is working on the requests that have been ignored during the last $\ _{i-1}$ time units. Then the time axis is split into the intervals

$$[\ _0 = 0;\ _0]; (\ _0;\ _1]; (\ _1;\ _1 + \ _2]; (\ _1 + \ _2;\ _1 + \ _2 + \ _3]; \ldots$$

Let us denote these intervals by $I_0; I_1; I_2; \ldots$. Moreover, let R_i be the set of those requests that come up in I_i. Clearly, the complete set of requests R is the disjoint union of all the R_i.

At the end of each interval I_i we solve an o ine problem: all requests to be scheduled are already available. The work on the computed schedule starts immediately (at the end of interval I_i) and is done $\ _{i+1}$ time units later (at the end of interval I_{i+1}). On the other hand, the time we need to serve the schedule is not more than   times the optimal completion time of ${R_i}^{o\ \ ine}$. In other words:

Lemma 1. *For all* i 0 *we have*

$$_{i+1} \quad \mathsf{OPT}^{comp}(R_i^{o\quad ine}):$$

Let us now prove the rst statement of Theorem 1 in a slightly more general version.

Theorem 2. *Let* > 0 *and* 1. *For all instances of* OlDarp *with* (;)*-reasonable request sets,* IGNORE *employing a -approximate algorithm for solving o ine instances of* OlDarp *yields a maximal flow time of no more than* 2 .

Proof. Let r be an arbitrary request in R_i for some i 0, i.e., r is released in I_i. By construction, the schedule containing r is nished at the end of interval I_{i+1}, i.e., at most $_i$ + $_{i+1}$ time units later than r was released. Thus, for all $i > 0$ we get that

$$\mathsf{IGNORE}^{maxflow}(R_i) \quad _i + \; _{i+1}:$$

If we can show that $_i$ for all $i > 0$ then we are done. To this end, let f : IR ! IR be a (;)-reasonable load bound for R. Then $\mathsf{OPT}^{comp}(R_i^{o\quad ine})$ $f(\;_i)$ because (R_i) $_i$.

By Lemma 1, we get for all $i > 0$

$$_{i+1} \quad \mathsf{OPT}^{comp}(R_i^{o\quad ine}) \quad f(\;_i) \quad \max f \;_i; \;\; g:$$

Using $_0 = 0$ the claim now follows by induction on i.

The statement of Theorem 1 concerning the average flow time of IGNORE follows from the fact that the average is never larger then the maximum.

Corollary 1. *Let* > 0. *For all -reasonable request sets algorithm* IGNORE *yields a average flow time no more than* 2 .

5 A Disastrous Example for REPLAN

We rst recall the strategy of algorithm REPLAN for the OlDarp. Whenever a new request becomes available, REPLAN computes a preliminary transportation schedule for the set R of all available requests by solving the problem of minimizing the total completion time of $R^{o\quad ine}$.

Then it moves the server according to that schedule until a new request arrives or the schedule is done. In the sequel, we provide an instance of OlDarp and a -reasonable request set R such that the maximal and the average flow time $\mathsf{REPLAN}^{maxflow}(R)$ is unbounded, thereby proving the remaining assertions of Theorem 1.

Theorem 3. *There is an instance of* OlDarp *under reasonable load such that the maximal and the average flow time of* REPLAN *is unbounded.*

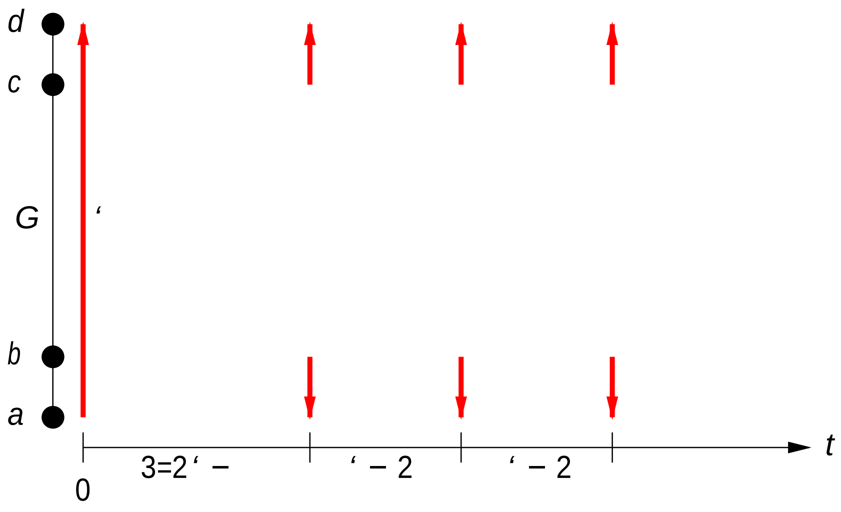

Fig. 1. A sketch of a $(2\frac{2}{3}‘)$-reasonable instance of OlDarp ($‘ = 18$).

Proof. In Fig. 1 there is a sketch of an instance for the OlDarp. The metric space is a path on four nodes $a; b; c; d$; the length of the path is ‘, the distances are $d(a; b) = d(c; d) =$, and hence $d(b; c) = ‘ - 2$. At time 0 a request from a to d is issued; at time $3{=}2‘ -$, the remaining requests periodically come in pairs from b to a and from c to d, resp. The time distance between them is $‘ - 2$.

We show that for $‘ = 18$ the request set R indicated in the picture is $2\frac{2}{3}‘$-reasonable. Indeed: it is easy to see that the rst request from a to d does not influence reasonableness. Consider an arbitrary set R_k of k adjacent pairs of requests from b to a resp. from c to d. Then the release span (R_k) of R_k is

$$(R_k) = (k - 1)(‘ - 2):$$

The o ine version $R_k{}^{o \quad ine}$ of R_k can be served in time

$$\mathrm{OPT}^{comp}(R_k{}^{o \quad ine}) = 2‘ + (k - 1) \quad 4 :$$

In order to nd the smallest paramter for which the request set R_k is -reasonable we solve for the integer $k - 1$ and get

$$k - 1 = \frac{2‘}{‘ - 6} = 3:$$

Hence, we can set to

$$:= \mathrm{OPT}^{comp}(R_4{}^{o \quad ine}) = 2\tfrac{2}{3}‘:$$

Now we de ne

$$f : \begin{cases} \mathbb{R} \; ! \; \mathbb{R}; & \\ \quad 7\!! & \text{for} \quad < \quad ; \\ & \text{otherwise}: \end{cases}$$

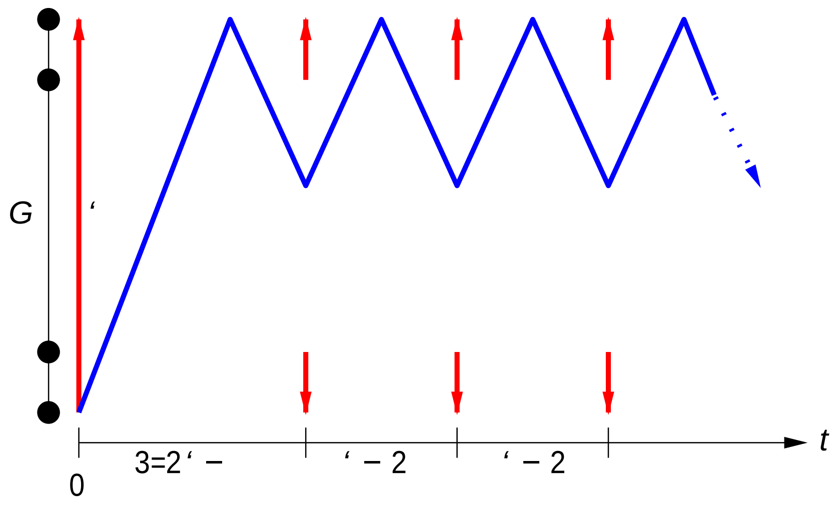

Fig. 2. The track of the REPLAN-server. Because a new pair of requests is issued exactly when the server is still closer to the requests at the top all the requests at the bottom will be postponed in an optimal preliminary schedule. Thus, the server always returns to the top when a new pair of requests arrives.

By construction, f is a load bound for R_4. Because the time gap after which a new pair of requests occurs is certainly larger than the additional time we need to serve it (o ine), f is also a load bound for R. Thus, R is -reasonable, as desired.

Now: how does REPLAN perform on this instance? In Fig. 2 we see the track of the server following the preliminary schedules produced by REPLAN on the request set R.

The maximal flow time of REPLAN on this instance is realized by the flow time of the request (3=2‘ − ; b; a), which is unbounded.

Moreover, since all requests from b to a are postponed after serving all the requests from c to d we get that REPLAN produces an unbounded average flow time as well.

In Fig. 3 we show the track of the server under the control of the IGNORE-algorithm. After an initial ine cient phase the server ends up in a stable operating mode. This example also shows that the analysis of IGNORE in Sect. 4 is sharp.

6 Reasonable Load as a General Framework

We introduced the new concept of reasonable request sets, using as example the problem OlDarp. However, the concept can be applied to any combinatorial

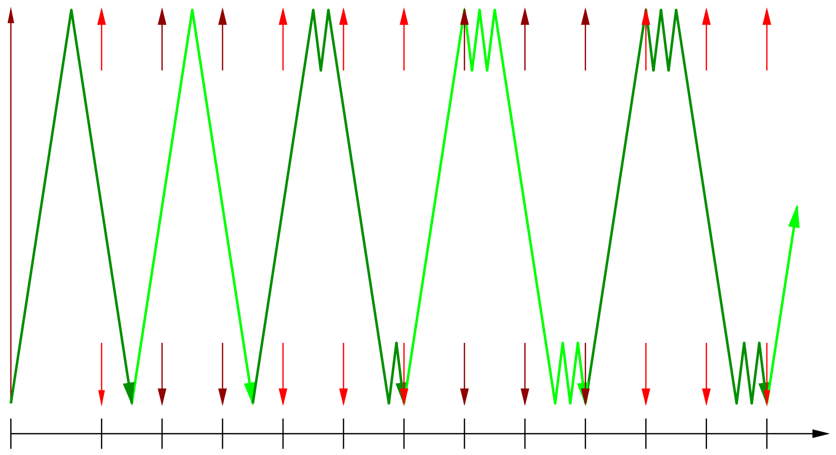

Fig. 3. The track of the IGNORE-server.

online problem with (possibly in nte) sets of time stamped requests, such as on-line scheduling, e.g., as described by Sgall [12], or the Online Traveling Salesman Problem, studied by Ausiello et al. [3].

The algorithms IGNORE and REPLAN represent general \online paradigms" which can be used for any online problem with time-stamped requests. We notice that the proof of the result that the average and maximal flow and waiting times of IGNORE are bounded by 2 has not explicitly drawn on any speci c property of OlDarp | this result holds for all combinatorial online problems with requests given by their release times.

The proof that the maximal flow and waiting time of a -reasonable request set is unbounded for REPLAN is equally applicable to the Online Traveling Salesman Problem by Ausiello et.al. [3]. We expect that the same is true for any \suf- ciently di cult" online problem with release times | for very simple problems, such as OlDarp on a zero dimensional space, the result trivially does not hold.

7 Conclusion

We have introduced the mathematical notion -reasonable describing the combinatorial di culty of a possibly in nite request set for OlDarp. For reasonable request sets we have given bounds on the maximal resp. average flow time of algorithm IGNORE for OlDarp; in contrast to this, there are instances of OlDarp where algorithm REPLAN yields an unbounded maximal and average flow time. One key property of our results is that they can be applied in continuously working systems. Computer simulations have meanwhile supported the theoretical results in the sense that algorithm IGNORE does not delay individual requests for an arbitraryly long period of time, whereas REPLAN has a tendency to do so [9].

While the notion of -reasonable is applicable to minimizing maximal flow time, it would be of interest to investigate an average analogue in order to prove non-trivial bounds for the average flow times.

References

1. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, 2000. To appear.
2. Mikhail J. Atallah and S. Rao Kosaraju. E cient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17:849{869, 1988.
3. Georgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line traveling salesman. Algorithmica, to appear.
4. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
5. Esteban Feuerstein and Leen Stougie. On-line single server dial-a-ride problems. Theoretical Computer Science, special issue on on-line algorithms, to appear.
6. Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.
7. Greg N. Frederickson and D. J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15:29{60, 1993.
8. Greg N. Frederickson, Matthew S. Hecht, and Chul Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7:178{193, 1978.
9. Martin Grötschel, Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau. Simulation studies for the online dial-a-ride-problem. Preprint SC 99-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.
10. Dietrich Hauptmeier, Sven O. Krumke, Jörg Rambau, and Hans C. Wirth. Euler is standing in line | dial-a-ride problems with fo-precedence-contraints. In Peter Widmeyer, Gabriele Neyer, and Stephan Eidenbenz, editors, *Graph Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 42{54. Springer, Berlin Heidelberg New York, 1999.
11. Hans Kellerer, Thomas Tautenhahn, and Gerhard Woeginger. Approximability and nonapproximabiblity results for minimizing total flow time on a single machine. In *Proceedings of the Symposium on the Theory of Computing*, 1996.
12. Jir Sgall. Online scheduling. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.
13. David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313{1331, December 1995.

The Online-TSP against Fair Adversaries

Michiel Blom[1]*, Sven O. Krumke[2]**, Willem de Paepe[3], and Leen Stougie[4]***

[1] KPN-Research, P. O. Box 421, 2260AK Leidschendam, The Netherlands,
[2] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, D-14195 Berlin-Dahlem, Germany,
krumke@zib.de
[3] Faculty of Technology Management, Technische Universiteit Eindhoven, P. O. Box 513, 5600MB Eindhoven, The Netherlands,
w.e.d.paepe@tm.tue.nl
[4] Faculty of Mathematics and Computer Science, Technische Universiteit Eindhoven, P. O. Box 513, 5600MB Eindhoven, The Netherlands,
L.Stougie@tue.nl

Abstract. In the online traveling salesman problem requests for visits to cities (points in a metric space) arrive online while the salesman is traveling. The salesman moves at no more than unit speed and starts and ends his work at a designated origin. The objective is to find a routing for the salesman which finishes as early as possible.

We consider the online traveling salesman problem when restricted to the non-negative part of the real line. We show that a very natural strategy is $3/2$-competitive which matches our lower bound. The main contribution of the paper is the presentation of a "*fair adversary*", as an alternative to the omnipotent adversary used in competitive analysis for online routing problems. The fair adversary is required to remain inside the convex hull of the requests released so far. We show that on $\mathbb{R}_0^+$ algorithms can achieve a strictly better competitive ratio against a fair adversary than against a conventional adversary. Specifically, we present an algorithm against a fair adversary with competitive ratio $(1+\sqrt{17})/4 \approx 1.28$ and provide a matching lower bound. We also show competitiveness results for a special class of algorithms (called diligent algorithms) that do not allow waiting time for the server as long as there are requests unserved.

1 Introduction

The traveling salesman problem is a well studied problem in combinatorial optimization. In the classical setting, one assumes that the complete input of an instance is available for an algorithm to compute a solution. In many cases this

* Supported by the TMR Network DONET of the European Community ERB TMRX-CT98-0202

** Research supported by the German Science Foundation (DFG, grant Gr 883/5-2)

*** Supported by the TMR Network DONET of the European Community ERB TMRX-CT98-0202 o ine optimization model does not reflect the real-world situation appropriately. In many applications not all requests for points to be visited are known in advance. Decisions have to be made *online* without knowledge about future requests.

Online algorithms are tailored to cope with such situations. Whereas o ine algorithms work on the complete input sequence, online algorithms only see the requests released so far and thus, in planning a route, have to account for future requests that may or may not arise at a later time. A common way to evaluate the quality of online algorithms is *competitive analysis* [3,5].

In this paper we consider the following online variant of the traveling salesman problem (called Oltsp in the sequel) which was introduced in [2]. Cities (requests) arrive online over time while the salesman is traveling. The requests are to be handled by a salesman-server that starts and ends his work at a designated origin. The objective is to nd a routing for the server which nishes as early as possible (in scheduling theory this goal is usually referred to as minimizing the *makespan*). In this model it is feasible for the server to wait at the cost of time that elapses. Decisions are revocable, as long as they have not been executed. Only history is irrevocable.

Previous Work. Ausiello et al. [2] present a 2-competitive algorithm for Oltsp which works in general metric spaces. The authors also show that for general metric spaces no deterministic algorithm can be c-competitive with $c < 2$. For the special case that the metric space is $\mathbb{R}$, the real line, they give a lower bound of $(9 + \sqrt{17})/8 \approx 1.64$ and a $7/4$-competitive algorithm. Just very recently Lipmann [7] devised an algorithm that is best possible for this case with competitive ratio equal to the before mentioned lower bound.

Our Contribution. In this paper the e ect of restricting the class of algorithms allowed and restricting the power of the adversary in the competitive analysis is studied. We introduce and analyze a new class of online algorithms which we call *diligent algorithms*. Roughly speaking, a diligent algorithm never sits idle while there is work to do. A precise de nition is presented in Section 3 where we also show that in general diligent algorithms are strictly weaker than algorithms that allow waiting time. In particular we show that no diligent algorithm can achieve a competitive ratio lower than $7/4$ for the Oltsp on the real line. The $7/4$-competitive algorithm in [2] is in fact a diligent algorithm and therefore best possible within this restricted class of algorithms.

We then concentrate on the special case of Oltsp when the underlying metric space is $\mathbb{R}_0^+$, the non-negative part of the real line. In Section 4 we show that an extremely simple and natural diligent strategy is $3/2$-competitive and that this result is best possible for (diligent and non-diligent) deterministic algorithms on $\mathbb{R}_0^+$. The main contribution is contained in Section 5. Here we deal with an objection frequently encountered against competitive analysis concerning the unrealistic power of the adversary against which performance is measured. Indeed, in the Oltsp on the real line the before mentioned $7/4$-competitive algorithm reaches its competitive ratio against an adversary that moves away from the previously released requests without giving any information to the online algorithm. We introduce a *fair adversary* that is in a natural way restricted in the context of the online traveling salesman problem studied here. It should be seen as a more reasonable adversary model. A fair adversary always keeps its server within the convex hull of the requests released so far. We show that this adversary model indeed allows for lower competitive ratios. For instance, the above mentioned $3/2$-competitive diligent strategy against the conventional adversary is $4/3$-competitive against the fair adversary. This result is best possible for diligent algorithms against a fair adversary.

	Diligent Algorithms	General Algorithms
General Adversary	LB = UB = $3/2$	LB = UB = $3/2$
Fair Adversary	LB = UB = $4/3$	LB = UB = $(1+\sqrt{17})/4$

Table 1. Overview of the lower bound (LB) and upper bound (UB) results for the competitive ratio of deterministic algorithms for Oltsp on $\mathbb{R}_0^+$ in this paper.

We also present a non-diligent algorithm with competitive ratio $(1+\sqrt{17})/4 \approx 1.28 < 4/3$ competing against the fair adversary. Our result is the first one that shows that waiting is actually advantageous in the Oltsp. The before mentioned algorithm in [7] also uses waiting, but became known after the one presented in this paper and has not been officially published yet. Such results are known already for online scheduling problems (see e.g. [6,4,8]) and, again very recently, also for an online dial-a-ride problem [1]. Our competitiveness result is complemented by a matching lower bound on the competitive ratio of algorithms against the fair adversary. Table 1 summarizes our results for Oltsp on $\mathbb{R}_0^+$.

2 Preliminaries

An instance of the *Online Traveling Salesman Problem* (Oltsp) consists of a metric space $M = (X, d)$ with a distinguished origin $o \in M$ and a sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests. In this paper we are mainly concerned with the special case that M is $\mathbb{R}_0^+$, the non-negative part of the real line endowed with the Euclidean metric, i.e., $X = \mathbb{R}_0^+ = \{x \in \mathbb{R} : x \geq 0\}$ and $d(x, y) = |x - y|$. A server is located at the origin o at time 0 and can move at most at unit speed.

Each *request* is a pair $\sigma_i = (t_i, x_i)$, where $t_i \in \mathbb{R}$ is the time at which request σ_i is released (becomes known), and $x_i \in X$ is the point in the metric space requested to be visited. We assume that the sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests is given in order of non-decreasing release times. For a real number t we denote by $\sigma_{\leq t}$ and $\sigma_{<t}$ the subsequence of requests in σ released up to time t and strictly before time t, respectively.

It is assumed that the online algorithm does neither have information about the time when the last request is released nor about the total number of requests.

An online algorithm for Oltsp must determine the behavior of the server at a certain moment t of time as a function of all the requests in $_t$ (and the current time t). In contrast, the o ine algorithm has information about all requests in the whole sequence already at time 0. A feasible online/o ine solution is a route for the server which serves all requested points, where each request is served not earlier than the time it is released, and which starts and ends in the origin o.

The objective in the Oltsp is to minimize the total completion time (also called the \makespan" in scheduling) of the server, that is, the time when the server has served all requests and returned to the origin.

Let alg( ) denote the completion time of the server moved by algorithm alg on the sequence of requests. We use opt to denote the optimal o ine algorithm. An online algorithm alg for Oltsp is *c-competitive*, if there exists a constant c such that for every request sequence the inequality alg( ) c opt( ) holds.

3 Diligent Algorithms

In this section we introduce a particular class of algorithms for Oltsp which we call *diligent algorithms*. Intuitively, a diligent algorithm should never sit and wait when it could serve yet unserved requests. A diligent server should also move towards work that has to be done directly without any detours. To translate this intuition into a rigorous de nition some care has to be taken.

De nition 1 (Diligent Algorithm). *An algorithm* alg *for* Oltsp *is called* diligent*, if it satis es the following conditions:*

1. *If there are still unserved requests, then the direction of the server operated by* alg *changes only if a new request becomes known, or the server is either in the origin or at a request that has just been served.*
2. *At any time when there are yet unserved requests, the server operated by* alg *either moves towards an unserved request or the origin at maximum (i.e. unit) speed. The latter case is only allowed if the server operated by* alg *is not yet in the origin.*

We emphasize that a diligent algorithm is allowed to move its server towards an unserved request and change his direction towards another unserved request or to the origin at the moment a new request becomes known.

Lemma 1. *No diligent online algorithm for* Oltsp *on the real line* $\mathbb{R}$ *has competitive ratio of less than* 7=4.

Proof. Suppose that alg is a diligent algorithm for Oltsp on the real line. Consider the following adversarial input sequence. At time 0 two requests $_1$ = (0;1=2) and $_2$ = (1=2;0) are released. There will be no further requests before time 1. Thus, by the diligence of the algorithm the server will be at the origin at time 1.

At time 1 two new requests at points 1 and −1, respectively, are released. Since the algorithm is diligent, starting at time 1 it must move its server to one of these requests at maximum, i.e., unit, speed. Without loss of generality assume that this is the request at 1. alg's server will reach this point at time 2. Starting at time 2, alg will have to move its server either directly towards the unserved request at −1 or towards the origin, which essentially gives the same movement and implies that the server is at the origin at time 3. At that time, the adversary issues another request at 1. Thus, alg's server will still need at least 4 time units to serve −1 and 1 and return at the origin. Therefore, he will not be able to complete its work before time 7.

The adversary handles the sequence by rst serving the request at −1, then the two requests at 1 and nally returns to the origin at time 4, yielding the desired result. ⊓⊔

This lower bound shows that the 7=4-competitive algorithm presented in [2], which is in fact a diligent algorithm is best possible within the class of diligent algorithms for the Oltsp on the real line.

4 The Oltsp on the Non-negative Part of the Real Line

We rst consider Oltsp on $\mathbb{R}_0^+$ when the o ine adversary is the conventional (omnipotent) opponent.

Theorem 1. *No deterministic algorithm for* Oltsp *on* $\mathbb{R}_0^+$ *has a competitive ratio of less than* 3=2.

Proof. At time 0 the request $_1 = (0; 1)$ is released. Let T be the time that the server operated by alg has served the request $_1$ and returned to the origin 0. If T 3, then no further request is released and alg is no better than 3=2-competitive since opt($_1$) = 2. Thus, assume that $T < 3$. In this case the adversary releases a new request $_2 = (T; T)$. Clearly, opt($_1$; $_2$) = $2T$. On the other hand alg($_1$; $_2$) $3T$, yielding a competitive ratio of 3=2. ⊓⊔

The following extremely simple strategy achieves a competitive ratio that matches this lower bound (as we will show below):

Strategy **mrin**(\Move-Right-If-Necessary") If a new request is released and the request is to the right of the current position of the server operated by mrin, then the mrin-server starts to move right at full speed. The server continues to move right as long as there are yet unserved requests to the right of the server. If there are no more unserved requests to the right, then the server moves towards the origin 0 at full speed.

It is easy to verify that Algorithm mrin is in fact a diligent algorithm. The following theorem shows that the strategy has a best possible competitive ratio for Oltsp on $\mathbb{R}_0^+$.

Theorem 2. *Strategy* mrin *is a diligent* 3=2*-competitive algorithm for the* Oltsp *on the non-negative part* $\mathbb{R}_0^+$ *of the real line.*

Proof. We show the theorem by induction on the number of requests in the sequence . It clearly holds if contains at most one request. The induction hypothesis is that it holds for any sequence of $m - 1$ requests.

Suppose that request $_m = (t; x)$ is the last request of $= {}_1; \ldots ; {}_{m-1}; {}_m$. If $t = 0$, then mrin is obviously 3=2-competitive, so we will assume that $t > 0$. Let f be the position of the request unserved by the mrin-server at time t (excluding $_m$), which is furthest away from the origin.

In case x f, mrin's cost for serving is equal to the cost for serving the sequence consisting of the rst $m-1$ requests of . Since new requests can never decrease the optimal o ine cost, the induction hypothesis implies the theorem.

Now assume that $f < x$. Thus, at time t the request in x is the furthest unserved request. mrin will complete its work no later than time $t + 2x$. The optimal o ine cost opt( ) is bounded from below by $\max f t + x; 2x g$. Therefore,

$$\frac{\mathrm{mrin}(\ )}{\mathrm{opt}(\ )} \quad \frac{t+x}{\mathrm{opt}(\ )} + \frac{x}{\mathrm{opt}(\ )} \quad \frac{t+x}{t+x} + \frac{x}{2x} = \frac{3}{2}.$$

□

The result established above can be used to obtain competitiveness results for the situation of the Oltsp on the real line when there are more than one server, and the goal is to minimize the time when the last of its servers returns to the origin 0 after all requests have been served.

Lemma 2. *There is an optimal o ine strategy for* Oltsp *on the real line with* k 2 *servers such that no server ever crosses the origin.*

Proof. Omitted in this abstract. □

Corollary 1. *There is a* 3=2*-competitive algorithm for the* Oltsp *with* k 2 *servers on the real line.* □

5 Fair Adversaries

The adversaries used in the bounds of the previous section are abusing their power in the sense that they can move to points where they know a request will pop up without revealing the request to the online server before reaching the point. As an alternative we propose the following more reasonable adversary that we baptized *fair adversary*. We show that we can obtain better competitive ratios for the Oltsp on $\mathbb{R}_0^+$ under this model. We will also see that under this adversary model there does exist a distinction in competitiveness between diligent and non-diligent algorithms. Recall that $_{<t}$ is the subsequence of consisting of those requests with release time strictly smaller than t.

De nition 2 (Fair Adversary). *An o ine adversary for the* Oltsp *in the Euclidean space* $(\mathbb{R}^n, \|.\|)$ *is* fair*, if at any moment* t*, the position of the server operated by the adversary is within the convex hull of the origin* o *and the requested points from* $_{<t}$.

In the special case of $\mathbb{R}_0^+$ a fair adversary must always keep its server in the interval $[0, F]$, where F is the position of the request with the largest distance to the origin 0 among all requests released so far. The following lower bound result shows that the Oltsp on the real line against fair adversaries is still a non-trivial problem.

Theorem 3. *No deterministic algorithm for* Oltsp *on* $\mathbb{R}$ *has competitive ratio less than* $(5+\sqrt{57})/8 \quad 1.57$ *against a fair adversary.*

Proof. Suppose that there exists a c-competitive online algorithm. The adversarial sequence starts with two requests at time 0, $_1 = (0, 1)$ and $_2 = (0, -1)$. Without loss of generality, we suppose that the rst request that is served is $_1$. At time 2 the online server can't have served both requests. We distinguish two main cases divided in some sub-cases.
Case 1: *None of the requests has been served at time* 2.

{ If at time 3 request $_1$ is still unserved, let t' be the rst time the server crosses the origin after serving the request. Clearly, $t' \quad 4$. At time t' the online server still has to visit the request in -1. If $t' > 4c - 2$ the server can not be c-competitive because the fair adversary can nish the sequence at time 4.
Thus, suppose that $4 \quad t' \quad 4c - 2$. At time t' a new request $_3 = (t', 1)$ is released. The online server can not nish the complete sequence before $t' + 4$, whereas the adversary needs at $t' + 1$. Therefore, $c \quad \frac{t'+4}{t'+1}$ and for $4 \quad t' \quad 4c - 2$ we have that

$$c \quad \frac{(4c-2)+4}{(4c-2)+1} = \frac{4c+2}{4c-1}$$

implying $c \quad (5+\sqrt{57})/8$.

{ If at time 3 the request $_1$ has already been served, the online server can not be to the left of the origin at time 3 (given the fact that at time 2 no request had been served). The adversary now gives a new request $_3 = (3, 1)$. There are two possibilities: either $_2$, the request in -1, is served before $_3$ or the other way round.
If the server decides to serve $_2$ before $_3$ then it can not complete before time 7. Since the adversary completes the sequence in time 4, the competitive ratio is at least $7/4$.
If the online server serves $_3$ rst, then again, let t' be the time that the server crosses the origin after serving $_3$. As before, we must have $4 \quad t' \quad 4c - 2$. At time t' the fourth request $_4 = (t', 1)$ is released. The same arguments as above apply to show that the algorithm is at least $(5+\sqrt{57})/8$-competitive.

Case 2: *One of the requests has been served at time* 2 *by the online server.* We assume without loss of generality that σ_1 has been served. At time 2 the third request $\sigma_3 = (2; 1)$ is released. In fact, we are back in the situation in which at time 2 none of the two requests are served. In case the movements of the online server are such that no further request is released by the adversary, the latter will complete at time 4. In the other cases the last released requests are released after time 4 and the adversary can still reach them in time. □

For comparison, the lower bound on the competitive ratio for the Oltsp in $\mathbb{R}$ against an adversary that is not restricted to be fair is $(9 + \sqrt{17})/8$ [2]. As mentioned before, only recently Lipmann [7] presented a $(9 + \sqrt{17})/8$-competitive algorithm against a non-fair adversary. He conjectures that a similar type of algorithm will also be best possible against a fair adversary. In contrast, the picture for the problem on the non-negative part of the real line is already complete (see Theorems 5 and 6 for diligent algorithms and Theorems 4 and 7 for non-diligent algorithms below).

Theorem 4. *No deterministic algorithm for* Oltsp *on* $\mathbb{R}_0^+$ *has competitive ratio of less than* $(1 + \sqrt{17})/4 \approx 1.28$ *against a fair adversary.*

Proof. Suppose that alg is c-competitive. At time 0 the adversary releases request $\sigma_1 = (0; 1)$. Let T denote the time that the server operated by alg has served this request and is back at the origin. For alg to be c-competitive, we must have that $T \leq c \cdot \text{opt}(\sigma_1) = 2c$, otherwise no further requests will be released. At time T the adversary releases a second request $\sigma_2 = (T; 1)$. The completion time of alg becomes then at least $T + 2$.

On the other hand, starting at time 0 the fair adversary moves its server to 1, lets it wait there until time T and then goes back to the origin 0 yielding a completion time of $T + 1$. Therefore,

$$\frac{\text{alg}(\sigma)}{\text{opt}(\sigma)} \geq \frac{T+2}{T+1} \geq \frac{2c+2}{2c+1} = 1 + \frac{1}{2c+1},$$

given the fact that $T \leq 2c$. Since by assumption alg is c-competitive, we have that $1 + 1/(2c + 1) \leq c$, implying that $c \geq (1 + \sqrt{17})/4$. □

For diligent algorithms we can show a higher lower bound against a fair adversary.

Theorem 5. *No deterministic diligent algorithm for* Oltsp *on* $\mathbb{R}_0^+$ *has competitive ratio of less than* $4/3$ *against a fair adversary.*

Proof. Consider the adversarial sequence $\sigma_1 = (0; 1)$, $\sigma_2 = (1; 0)$, and $\sigma_3 = (2; 1)$. By its diligence the online algorithm will start to travel to 1 at time 0, back to 0 at time 1, arriving there at time 2. Then its server has to visit 1 again, so that he will finish no earlier than time 4. Obviously, the optimal offline solution is to leave 1 not before time 2, and finishing at time 3. □

We show now that the algorithm mrin presented before has a better competitive ratio against the fair adversary than the ratio of 3=2 achieved against a conventional adversary. In fact we show that the ratio matches the lower bound for diligent algorithms proved in the previous theorem.

Theorem 6. *Strategy* mrin *is a* 4=3-*competitive algorithm for the* Oltsp *on* $\mathbb{R}_0^+$ *against a fair adversary.*

Proof. Omitted in this abstract.

Thus, Algorithm mrin attains a best possible competitive ratio against the fair adversary among all diligent algorithms. Given the lower bound for general non-diligent algorithms in Theorem 4 we aim now at designing an online algorithm that obtains better competitive ratios against a fair adversary. In view of Theorem 5 such an algorithm will have to be non-diligent, i.e., incorporate waiting times.

The problem with Algorithm mrin is that shortly after it starts to return towards the origin from the furthest previously unserved request, a new request to the right of its server arrives. In this case the mrin-server has to return to a position it just left. Algorithm ws presented below attempts successfully to avoid this pitfall.

Strategy ws(\Wait Smartly") The ws-server moves right if there are yet unserved requests to the right of his present position. Otherwise, it takes the following actions. Suppose it arrives at his present position, which is a currently rightmost unserved request, $s(t)$ at time t.

1. Compute the the optimal o ine solution value opt($_t$) for all requests released up to time t.
2. Determine a waiting time W := opt($_t$) $-$ $s(t)$ $-$ t, with = $(1 + \sqrt{17})=4$.
3. Wait at point $s(t)$ until time $t + W$ and then start to move back to the origin 0.

We notice that when the server is moving back to the origin and no new requests are released until time $t + W + s(t)$, then the ws-server reaches the origin 0 at time $t + W + s(t) =$ opt($_t$) having served all requests released so far. If a new request is released at time t^0 $W + t + s(t)$ and the request is to the right of $s(t^0)$, then the ws-server starts to move to the right immediately until it reaches the furthest unserved request.

Theorem 7. *Algorithm* ws *is -competitive with* = $(1 + \sqrt{17})=4$ $1{:}28$ *for the* Oltsp *on* $\mathbb{R}_0^+$ *against a fair adversary.*

Proof. By the de nition of the waiting time it is su cient to prove that at any point where a waiting time is computed this waiting time is non-negative. In that case the server will always return at o before time opt( ). This is clearly true if the sequence contains at most one request. We make the induction hypothesis that it is also true for any sequence of at most $m - 1$ requests.

Let = $_1; : : : ;$ $_m$ be any sequence of requests and let $_m$ =: $(t; x)$ be the request released last. If t = 0, then there is nothing left to show, so we will assume for the remainder of the proof that t > 0.

We denote by $s(t)$ and s (t) the positions of the ws- and the fair adversary's server at time t, respectively. We also let $r_f = (t_f; f)$ be the furthest (i.e. most remote from the origin) yet unserved request by ws at time t excluding the request $_m$. Finally, let $r_F = (t_F; F)$ be the furthest released request in $_1; : : : ;$ $_{m-1}$. Obviously f F. Again, we distinguish three di erent cases depending on the position of x relative to f and F.

Case 1: x f

Since the ws-server has to travel to f anyway and by the induction hypothesis there was a non-negative waiting time in f or $s(t)$ (depending on whether $s(t) >$ f or $s(t)$ f) before request $_m$ was released, the waiting time in f or $s(t)$ can not decrease since the optimal o ine completion time can not decrease by an additional request).

Case 2: f $x < F$

If $s(t)$ x, then again by the induction hypothesis and the fact that the route length of ws's server does not increase, the possible waiting time at $s(t)$ is non-negative.

Thus we can assume that $s(t) < x$. The ws-server will now travel to point x, arrive there at time $t + d(s(t); x)$, and possibly wait there some time W before returning to the origin, with

$$W = \quad \mathrm{opt}(\ ) - (t + d(s(t); x)) - x:$$

Inserting the obvious lower bound opt( ) $t + x$ yields

$$W \quad (\ - 1)\mathrm{opt}(\ ) - d(s(t); x): \tag{1}$$

To bound opt( ) in terms of $d(s(t); x)$ consider the time t^0 when the ws-server had served the request at F and started to move left. Clearly $t^0 < t$ since otherwise $s(t)$ could not be smaller than x as assumed. Thus, the sub-sequence $_{t'}$ of does not contain $(t; x)$. By the induction hypothesis, ws is -competitive for the sequence $_{t'}$. At time t^0 when he left F he would have arrived in the origin at time opt($_{t'}$), i.e.,

$$t^0 + F = \quad \mathrm{opt}(\ _{t'}): \tag{2}$$

Notice that t $t^0 + d(F; s(t))$. Since opt($_{t'}$) $2F$ we obtain from (2) that

$$t \quad 2F - F + d(F; s(t)) = (2 \ - 1)F + d(s(t); F): \tag{3}$$

Since by assumption we have $s(t) < x < F$ we get that $d(s(t); x)$ $d(s(t); F)$ and $d(s(t); x)$ F, which inserted in (3) yields

$$t \quad (2 \ - 1)d(s(t); x) + d(s(t); x) = 2 \ d(s(t); x): \tag{4}$$

We combine this with the previously mentioned lower bound opt( ) $t + x$ to obtain:

$$\mathrm{opt}(\ ) \quad 2 \ d(s(t); x) + x \quad (2 \ + 1)d(s(t); x): \tag{5}$$

Using inequality (5) in (1) gives

$$\begin{aligned} W &\quad (\; - 1)(2 \; + 1)d(s(t); x) - d(s(t); x) \\ &= (2 \;^2 - \; - 2)d(s(t); x) \\ &= \left(\frac{9 + \sqrt{17}}{4} - \frac{1 + \sqrt{17}}{4} - 2 \right) d(s(t); x) \\ &= 0. \end{aligned}$$

This completes the proof for the second case.

Case 3: $f \quad F \quad x$

Starting at time t the ws-server moves to the right until he reaches x, and after waiting there an amount W returns to 0, with

$$W = \quad \text{opt}(\;) - (t + d(s(t); x)) - x. \tag{6}$$

We will show that also in this case $W \quad 0$. At time t the adversary's server still has to travel at least $d(s \; (t); x) + x$ units. This results in

$$\text{opt}(\;) \quad t + d(s \; (t); x) + x.$$

Since the o ine adversary is fair, its position $s \; (t)$ at time t can not be strictly to the right of F.

$$\text{opt}(\;) \quad t + d(F; x) + x. \tag{7}$$

Insertion into (6) yields

$$W \quad (\; - 1)\text{opt}(\;) - d(s(t); F) \tag{8}$$

since $F > s(t)$ by de nition of the algorithm.

The rest of the arguments are similar to those used in the previous case. Again that ws's server started to move to the left from F at some time $t' < t$, and we have

$$t' + F = \quad \text{opt}(\; _{t'}). \tag{9}$$

Since $t \quad t' + d(s(t); F)$ and $\text{opt}(\; _{t'}) \quad 2F$ we obtain from (9) that

$$t \quad 2F - F + d(s(t); F) = (2 \; - 1)F + d(s(t); F) \quad 2 \; d(s(t); F).$$

We combine this with (7) and the fact that $x \quad d(s(t); F)$ to achieve

$$\text{opt}(\;) \quad 2 \; d(s(t); F) + d(F; x) + x \quad (2 \; + 1)d(s(t); F).$$

Using this inequality in (8) gives

$$\begin{aligned} W &\quad (\; - 1)(2 \; + 1)d(s(t); F) - d(s(t); F) \\ &= (2 \;^2 - \; - 2)d(s(t); F) \\ &= \left(\frac{9 + \sqrt{17}}{4} - \frac{1 + \sqrt{17}}{4} - 2 \right) d(s(t); F) \\ &= 0. \end{aligned}$$

This completes the proof. ◻

6 Conclusions

We introduced an alternative more fair performance measure for online algorithms for the traveling salesman problem. The rst results are encouraging. On the non-negative part of the real line the fair model allows a strictly lower competitive ratio than the conventional model with an omnipotent adversary.

Next to that we considered a restricted class of algorithms for the online traveling salesman problems, suggestively called diligent algorithms. We showed that in general diligent algorithms have strictly higher competitive ratios than algorithms that sometimes leaves the server idle, to wait for possible additional information. In online routing companies, like courier services or transportation companies waiting instead of immediately starting as soon as requests are presented is common practice. Our results support this strategy.

It is still open to nd a best possible non-diligent algorithm for the problem on the real line against a fair adversary. However, it is very likely that an algorithm similar to the best possible algorithm presented in [7] against a non-fair adversary will appear to be best possible for this case.

We notice here that for general metric spaces the lower bound of 2 on the competitive ratio of algorithms in [2] is established with a fair adversary as opponent. Moreover, a diligent algorithm is presented which has a competitive ratio that meets the lower bound.

We hope to have encouraged research into ways to restrict the power of adversaries in online competitive analysis.

Acknowledgement: Thanks to Maarten Lipmann for providing the lower bound in Theorem 3.

References

1. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, 2000. To appear.
2. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line traveling salesman. *Algorithmica*, 1999. To appear.
3. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
4. B. Chen, A. P. A. Vestjens, and G. J. Woeginger. On-line scheduling of two-machine open shops where jobs arrive over time. *Journal of Combinatorial Optimization*, 1:355{365, 1997.
5. A. Fiat and G. J. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.
6. J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proceedings of the 5th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, pages 404{414, 1996.

7. M. Lipmann. The online traveling salesman problem on the line. Master's thesis, Department of Operations Research, University of Amsterdam, The Netherlands, 1999.
8. C. Philips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. In *Proceedings of the 4th Workshop on Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 86{97, 1995.

QuickHeapsort, an Efficient Mix of Classical Sorting Algorithms

Domenico Cantone and Gianluca Cincotti

Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I–95125 Catania, Italy
{cantone,cincotti}@cs.unict.it

Abstract. We present a practically efficient algorithm for the internal sorting problem. Our algorithm works *in-place* and, on the average, has a running-time of $O(n \log n)$ in the length n of the input. More specifically, the algorithm performs $n \log n + 3n$ comparisons and $n \log n + 2.65n$ element moves on the average.
An experimental comparison of our proposed algorithm with the most efficient variants of Quicksort and Heapsort is carried out and its results are discussed.

Keywords: In-place sorting, heapsort, quicksort, analysis of algorithms.

1. Introduction

The problem of sorting an initially unordered collection of keys is one of the most classical and investigated problems in computer science. Many different sorting algorithms exist in literature. Among the comparison based sorting methods, Quicksort [7, 17, 18] and Heapsort [4, 21] turn out, in most cases, to be the most efficient general-purpose sorting algorithms.

A good measure of the running-time of a sorting algorithm is given by the total number of key comparisons and the total number of element moves performed by it. In our presentation, we mainly focus our attention on the number of comparisons, since this often represents the dominant cost in any reasonable implementation. Accordingly, to sort n elements the classical Quicksort algorithm performs $1.386 n \log n - 2.846 n + 1.386 \log n$ key comparisons on the average and $O(n^2)$ key comparisons in the worst-case, whereas the classical Heapsort algorithm, due to Floyd [4], performs $2n \log n + \Theta(n)$ key comparisons in the worst-case.

Several variants of Heapsort are reported in literature. One of the most efficient is the Bottom-Up-Heapsort algorithm discussed by Wegener in [19], which performs $n \log n + f(n)n$ key comparisons on the average, where $f(n) \in [0.34..0.39]$, and no more than $1.5 n \log n + O(n)$ key comparisons in the worst-case. In [9, 10], Katajainen uses a median-finding procedure to reduce the number of comparisons required by Bottom-Up-Heapsort, completely eliminating the sift-up phase. This idea has been further refined by Rosaz in [16]. It is to be noted, though, that the algorithms described in [9, 10, 16] are mostly of theoretical interest only, due to the overhead introduced by the median- nding procedure.

This paper tries to build a bridge between theory and practice. More speci -cally, our goal is to produce a *practical* sorting algorithm which couples some of the theoretical ideas introduced in the algorithms cited above with the e cient strategy used by Quicksort.

Compared to Quicksort, our proposed algorithm, called QuickHeapsort, works \in-place", i.e. no stack is needed for recursion. Moreover, its average number of comparisons is shown to be less than $n \log n + 3n$. Its behavior is also analyzed from an experimental point of view by comparing it to that of Heapsort, Bottom-Up-Heapsort, and some variants of Quicksort. The results show that QuickHeapsort has a good practical behavior especially when key comparison operations are computationally expensive.

The paper is organized as follows. In Section 2 we introduce a variant of the Heapsort algorithm which does not work in-place, just to present the main idea upon which QuickHeapsort is based. The QuickHeapsort algorithm is fully described and analyzed in Section 3. An experimental session with some empirical results aiming at evaluating and comparing its e ciency in practice is discussed in Section 4. Section 5 concludes the paper with some nal remarks.

2. A Not In-Place Variant of the Heapsort Algorithm

In this section we illustrate a variant of the Heapsort algorithm, External-Heapsort, which uses an external array to store the output. For this reason, External-Heapsort is mainly of theoretical interest and we present it just to introduce the main idea upon which the QuickHeapsort algorithm, to be described in the next section, is based. External-Heapsort sorts n elements in $(n \log n)$ time by performing at most $n\lfloor\log n\rfloor + 2n$ key comparisons, and at most $n\lfloor\log n\rfloor + 4n$ element moves in the worst-case.

We begin by recalling some basic concepts about the classical binary-heap data structure. A *max-heap* is a binary tree with the following properties:

1. it is *heap-shaped*: every level is complete, with the possible exception of the last one; moreover the leaves in the last level occupy the leftmost positions;
2. it is *max-ordered*: the key value associated with each non-root node is not larger than that of its parent.

A *min-heap* can be de ned by substituting the max-ordering property with the dual min-ordering one. The root of a max-heap (resp. min-heap) always contains the largest (resp. smallest) element of the heap. We refer to the number of elements in a heap as its *size*; the *height* of a heap is the height of the associated binary tree.

A heap data structure of size n can be implicitly stored in an array $A[1..n]$ with n elements without using any additional pointer as follows. The root of the heap is the element $A[1]$. Left and right sons (if they exist) of the node stored into $A[i]$ are, respectively, $A[2i]$ and $A[2i+1]$, and the parent of the node stored into $A[i]$ (with $i > 1$) is $A[\lfloor \frac{i}{2} \rfloor]$.

In all Heapsort algorithms, the input array is sorted in ascending order, by rst building a max-heap and then by performing n extractions of the root element. After each extraction, the element in the last leaf is rstly moved into the root and subsequently moved down along a suitable path until the max-ordering property is restored.

Bottom-Up-Heapsort works much like the classical Heapsort algorithm. The only di erence lies is the rearrangement strategy used after each max-extraction: starting from the root and iteratively moving down to the child containing the largest key, when a leaf is reached it climbs up until it nds a node x with a key larger than the root key. Subsequently, all elements in the path from x to the root are shifted one position up and the old root is moved into x.

The algorithm External-Heapsort, whose pseudo-code is shown in Fig. 1, takes the elements of the input array $A[1..n]$ and returns them in ascending sorted order into the output array $Ext[1..n]$.

External-Heapsort starts by constructing a heap and successively performs n extractions of the largest element. Extracted elements are moved into the output array in reverse order. After each extraction, the heap property is restored by a procedure similar to the bottom-phase of Bottom-Up-Heapsort. Speci cally, starting at the root of the heap, the son with the largest key is chosen and it is moved one level up. The same step is iteratively repeated until a leaf, called *special leaf*, is reached. At this point the value $-\infty$ is stored into the special leaf key eld.[1] The path from the root to the special leaf is called a *special path*.

Notice that no sift-up phase is executed and that the length of special paths does not decrease during the execution of the algorithm.

External-Heapsort makes use of the procedure Build-Heap and the function Special-Leaf. Build-Heap rearranges the input array $A[1..n]$ into a classical max-heap, e.g., by using the standard heap-construction algorithm by Floyd [4]. The function Special-Leaf, whose pseudo-code is also shown in Fig. 1, assumes that the value contained in the root of the heap has already been removed.

Correctness of the algorithm follows by observing that if a node x contains the key $-\infty$ than the whole sub-tree rooted at x contains $-\infty$'s. It is easy to check that the max-ordering property is ful lled at each extraction.

We proceed now to the analysis of the number of key comparisons and elements moves performed by External-Heapsort both in the worst and in the average case.[2]

Many variants for building heaps have been proposed in the literature [1, 6, 13, 19, 20], requiring quite involved implementations. Since our goal is to give a practical and e cient general-purpose sorting algorithm, we simply use the

[1] We assume that a key value $-\infty$, smaller than all keys occurring in $A[1..n]$, is available.

[2] In the average-case analysis we make use of the assumption that all permutations are equally likely.

External-Heapsort

```
PROCEDURE  External-Heapsort (Input Array $A$, Integer $n$; Output Array $Ext$)
  Var Integer $j, l$;
      Begin
            Build-Heap ($A, n$);
            For $j := n$ downto 1 do
                $Ext[j] := A[1]$;
                $l :=$ Special-Leaf ($A, n$);
                Key($A[l]$) $:= -\infty$;
            End for;
      End;

FUNCTION   Special-Leaf (Input Array $A$, Integer $n$) : Integer
  Var Integer $i$;
      Begin
            $i := 2$;
            While $i < n$ do
                If Key($A[i]$) $<$ Key($A[i+1]$) then $i := i + 1$; End if;
                $A[\lfloor \frac{i}{2} \rfloor] := A[i]$;
                $i := 2i$;
            End while;
            If $i = n$ then
                $A[\frac{i}{2}] := A[n]$;
                $i := 2i$;
            End if;
            Return $\frac{i}{2}$;
      End;
```

Fig. 1. Pseudo-code of External-Heapsort algorithm

classical heap-construction procedure due to Floyd [4, 11]. In such a case we need the following partial results [2, 11, 15]:

Lemma 1. *In the worst-case, the classical heap-construction algorithm builds a heap with n elements by performing at most $2n$ key comparisons and $2n$ element moves.* □

Lemma 2. *On the average, constructing a full-heap, i.e. a heap of size $n = 2^l - 1, l > 0$, with the classical algorithm requires $1.88n$ key comparisons and $1.53n$ element moves.* □

The number of key comparisons and element moves performed by the Special-Leaf function obviously depends only on the size of the input array, so that worst- and average-case values coincides for it.

Lemma 3. *Given a heap of size n, the Special-Leaf function performs exactly $b\log nc$ or $b\log nc - 1$ key comparisons and the same number of element moves.* □

The preceding lemmas yield immediately the following result.

Theorem 1. *External-Heapsort sorts n elements in $(n \log n)$ worst-case time by performing fewer than $nb\log nc + 2n$ key comparisons and $nb\log nc + 4n$ element moves.*[3] *Moreover, on the average, $H^{[c]}_{avg}(n)$ key comparisons and $H^{[m]}_{avg}(n)$ element moves are performed, where*

$$nb\log nc + 0{:}88n \quad H^{[c]}_{avg}(n) \quad nb\log nc + 1{:}88n;$$
$$nb\log nc + 0{:}53n \quad H^{[m]}_{avg}(n) \quad nb\log nc + 3{:}53n:$$

□

In the following, we will also use a min-heap variant of the External-Heapsort algorithm. In particular, special paths in min-heaps are obtained by following the children with smallest key and the value -1 is replaced by $+1$. Obviously, the same complexity analysis can be carried out for the min-heap variant of External-Heapsort.

3. QuickHeapsort

In this section, a practical and e cient in-place sorting algorithm, called QuickHeapsort, is presented. It is obtained by a mix of two classical algorithms: Quicksort and Heapsort. More speci cally, QuickHeapsort combines the Quicksort partition step with two *adapted* min-heap and max-heap variants of the External-Heapsort algorithm presented in the previous section, where in place of the in nity keys 1, only occurrences of keys in the input array are used.

As we will see, QuickHeapsort works in place and is completely iterative, so that additional space is not required at all.

The computational complexity analysis of the proposed algorithm reveals that the number of key comparisons performed is less than $n \log n + 3n$ on the average, with n the size of the input, whereas the worst-case analysis remains the same of classical Quicksort. From an implementation point of view, QuickHeapsort preserves Quicksort e ciency, and it has in many cases better running times than Quicksort, as the experimental Section 4 illustrates.

Analogously to Quicksort, the rst step of QuickHeapsort consists in choosing a *pivot*, which is used to partition the array. We refer to the sub-array with the smallest size as *heap area*, whereas the largest size sub-array is referred to as *work area*. Depending on which of the two sub-arrays is taken as heap-area, the

[3] As will be clear in the next section, it is convenient to count the assignment of -1 to a node as an element move.

adapted max-heap or min-heap variant of External-Heapsort is applied and the work area is used as an external array. At the end of this stage, the elements moved in the work area are in correct sorted order and the remaining unsorted part of the array can be processed iteratively in the same way.

A detailed description of the algorithm follows.

1. Let $A[1::n]$ be the input array of n elements to be sorted in ascending order. A pivot M, of index m, is chosen in the set $fA[1];A[2];:::;A[n]g$. As in Quicksort, the choice of the pivot can be done in a deterministic way (with or without sampling) or randomly. The computational complexity analysis of the algorithm is influenced by the choice adopted.
2. The array $A[1::n]$ is partitioned into two sub-arrays, $A[1::Pivot-1]$ and $A[Pivot+1::n]$, such that $A[Pivot]=M$, the keys in $A[1::Pivot-1]$ are larger than or equal to M, and the keys in $A[Pivot+1::n]$ are smaller than or equal to M.[4] The sub-array with the smallest size is assumed to be the *heap area*, whereas the other one is treated as the *work area* (if the two sub-arrays have the same size, a choice can be made non-deterministically).
3. Depending on which sub-array is taken as heap area, the adapted max-heap or min-heap variant of External-Heapsort is applied using the work area as external array. Moreover occurrences of keys contained in the work area are used in place of the in nity values 1.
   More precisely, if $A[1::Pivot-1]$ is the heap area, then the max-heap version of External-Heapsort is applied to it using the right-most region of the work area as external array. In this case, at the end of the stage, the right-most region of the work area will contain the elements formerly in $A[1::Pivot-1]$ in ascending sorted order.
   Similarly, if $A[Pivot+1::n]$ is the heap area, then the min-heap version of External-Heapsort is applied to it using the left-most region of the work area as external array. In this case, at the end of the stage, the left-most region of the work area will contain the elements formerly in $A[Pivot+1::n]$ in ascending sorted order.
4. The element $A[Pivot]$ is moved in the correct place and the remaining part of $A[1::n]$, i.e. the heap area together with the unused part of the work area, is iteratively sorted.

Correctness of QuickHeapsort follows from that of the max-heap and min-heap variants of External-Heapsort, by observing that assigning to a special leaf a key value taken in the work area is completely equivalent to assigning the key value -1, in the case of the max-heap variant, or the key value $+1$, in the case of the min-heap variant.

Complexity results in the average case are summarized in the theorem below. For simplicity, the results have been obtained only in the case in which pivots are chosen deterministically and without sampling, e.g. always the rst element of the array is chosen.

[4] Observe that Quicksort partitions the array in the reverse way.

Lemma 4. *Let* $H(n) = n \log n + \ n$ *and* $f_1(n); f_2(n)$ *be functions of type* $\ n + o(n)$ *for all* $n\ 2$ N, *with* $\ ; \ 2$ R. *The solution to the following recurrence equations, with initial conditions* $C(1) = 0$ *and* $C(2) = 1$*:*

$$C(2n) = \frac{1}{2n}\left[(2n+1)\ \ C(2n-1) - C(n-1) + H(n-1) + f_1(n)\right]; \quad (1)$$

$$C(2n+1) = \frac{1}{2n+1}\left[2(n+1)\ \ C(2n) - C(n) + H(n) + f_2(n)\right]; \quad (2)$$

for all $n\ 2$ N, *is:*

$$C(n) = n \log n + (\ \ + \ \ - 2{:}8854)n + o(n){:}$$

Proof. Among the reasonable solutions, we posit the trial solution:

$$C(n) = an \log n + bn + c \log n \qquad \text{with } a; b; c\ 2\ \mathrm{R}{:} \quad (3)$$

In several of the calculations we need to manipulate expressions of the form $\log(m + t)$ with $m\ 2$ N$; m > 1$ and $t = \ \ 1$. The expansion of the natural logarithm for small $x\ 2$ R to second order (we do not need any further here), $\ln(1+x) = x - \frac{x^2}{2}$, gives $\ln(m+t) = \ln[m\ \ (1+\frac{t}{m})] = \ln m + \frac{t}{m} - \frac{t^2}{2m^2}$. Multiplying by $s = 1{=}(\ln 2)\ \ 1{:}4427$, we get:

$$\log(m+t) = \log m + \frac{st}{m} - \frac{s}{2m^2}{:}$$

Using such expansion in the de nition of $H(n)$ and in (3), we get:

$$H(n-1) = n \log n + \ \ n - \log n - (\ \ + s) + \frac{s}{2n}; \quad (4)$$

$$C(n-1) = an \log n + bn + (c-a)\log n - (as+b) + s\frac{a-2c}{2n}; \quad (5)$$

$$C(2n-1) = 2an \log n + 2(a+b)n + (c-a)\log n + \\ + (c-a-b-as) + s\frac{a-2c}{4n}; \quad (6)$$

where the lowest order terms are not considered.

Let $S = s(c - \frac{1}{2}a + 1)$; substituting (4), (5) and (6) into equation (1) and simplifying, we nd the following:

$$(1-a)n \log n + (\ \ + \ \ - b - 2as)n - \log n + (c - a - \ \ - S) + \frac{S}{2n} + o(n) = 0{:}$$

Examining the leading coe cients in such equality, we get the asymptotic *consistency requirements*:

$$a = 1; \qquad b = \ \ + \ \ - 2s{:}$$

Such constraints are similarly obtained expanding equation (2).

Clearly, the missing requirement about c, simply means the posited solution does not have the *exact* functional form. The two leading terms of the solution are surprisingly precise, indeed by numerical computation, solution (3) tracks the behavior of $C(n)$ fairly well for an extended range of n. ■

Theorem 2. *QuickHeapsort sorts n elements in-place in $O(n \log n)$ average-case time. Moreover, on the average, it performs no more than $n \log n + 3n$ key comparisons and $n \log n + 2.65n$ element moves.*

Proof. First, we estimate the average number of key comparisons.

Let $H_{avg}(n)$ (resp. $H'_{avg}(n)$) be the average number of key comparisons to sort n elements with the adapted max-heap (resp. min-heap) version of External-Heapsort (see the beginning of Section 3). Plainly, we have $H_{avg}(i) = H'_{avg}(i) = 0$ with $i = 0, 1$.

Let $C(n)$ be the average number of key comparisons to sort n elements with QuickHeapsort. We have $C(1) = 0$ and, for all $n \in \mathbb{N}$:

$$C(2n) = p(2n) + \frac{1}{2n}\left[\sum_{j=1}^{n}[H_{avg}(j-1) + C(2n-j)] + \right.$$

$$\left. + \sum_{j=n+1}^{2n}[H'_{avg}(2n-j) + C(j-1)]\right];$$

$$C(2n+1) = p(2n+1) + \frac{1}{2n+1}\left[\sum_{j=1}^{n}[H_{avg}(j-1) + C(2n+1-j)] + \right.$$

$$\left. + [H_{avg}(n) + C(n)] + \sum_{j=n+2}^{2n+1}[H'_{avg}(2n+1-j) + C(j-1)]\right].$$

To compute the total average number of key comparisons we add the number of comparisons $p(m) = m+1$ needed to partition the array of size m to the average number of comparisons needed to sort the two sub-arrays obtained. The index j denotes all the possible choices (uniformly distributed) for the pivot. Obviously $H'_{avg}(n) = H_{avg}(n)$, so by simple indices manipulation, we obtain the following recurrence equations:

$$(2n) \cdot C(2n) = (2n) \cdot p(2n) + 2\sum_{j=1}^{n}[H_{avg}(j-1) + C(2n-j)]; \quad (7)$$

$$(2n+1) \cdot C(2n+1) = (2n+1) \cdot p(2n+1) + [H_{avg}(n) + C(n)] +$$

$$+ 2\sum_{j=1}^{n}[H_{avg}(j-1) + C(2n+1-j)]. \quad (8)$$

They depend on the previous history but can be reduced to semi-first order recurrences. Let $(8')$ be the equation obtained from (8) by substituting the index n with $n-1$. Subtracting equation (7) from $(8')$ and from (8) we obtain the following equations:

$$(2n) \cdot C(2n) = (2n+1) \cdot C(2n-1) - C(n-1) + H_{avg}(n-1) + f_1(n);$$

$$(2n+1) \cdot C(2n+1) = (2n+2) \cdot C(2n) - C(n) + H_{avg}(n) + f_2(n);$$

where $f_1(n) = (2n)\,p(2n) - (2n-1)\,p(2n-1) = 4n$ and $f_2(n) = (2n+1)\,p(2n+1) - (2n)\,p(2n) = 4n+2$. By using Lemma 4 and the upper bound in Theorem 1, it can be shown that the recurrence equation satisfies $C(n) \le n\log n + 3n$.

Analogous recurrence equations can be written to get the average number of element moves. In such a case, the function $H_{avg}(n)$ (resp. $H'_{avg}(n)$) denotes the average number of element moves to sort n elements with the adapted max-heap (resp. min-heap) version of External-Heapsort; whereas $p(m)$ is three times the average number $q(m)$ of exchanges used during the partitioning stage of a size m array.

If the chosen pivot $A[1]$ is the k-th smallest element in the array of size m, $q(m)$ is the number of keys among $A[2], \ldots, A[k]$ which are smaller than the pivot. There are exactly t such keys with probability $p^{(m)}_{k,t} = \binom{m-k}{t}\binom{k-1}{k-1-t} \Big/ \binom{m-1}{k-1}$. Averaging on t and k, we get:

$$q(m) = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=0}^{k-1}\left[t\;p^{(m)}_{k,t}\right] =$$

$$= \frac{1}{m}\sum_{k=1}^{m}\frac{m-k}{\binom{m-1}{k-1}}\left[\sum_{t=0}^{k-1}\binom{m-k-1}{t-1}\binom{k-1}{k-1-t}\right] = \frac{1}{6}(m-2),$$

where the last equality is obtained by two applications of Vandermonde's convolution.

From $p(m) = \frac{1}{2}(m-2)$, we get $f_1(n) = 2n - \frac{3}{2}$ and $f_2(n) = 2n - \frac{1}{2}$. Thus, Lemma 4 and the upper bound in Theorem 1 yield immediately that the average number of element moves is no more than $n\log n + 2.65n$. ■

4. Experimental Results

In this section we present some empirical results concerning the performance of our proposed algorithm. Specifically, we will compare the number of basic operations and the timing results of both QuickHeapsort (QH) and its variant clever-QuickHeapsort (c-QH) (which implements the median of three elements strategy) with those of the following comparison-based sorting algorithms:

- the classical Heapsort algorithm (H), implemented with a trick which saves some element moves;
- the Bottom-Up-Heapsort algorithm (BU), implemented with bit shift operations, as suggested in [19];
- the iterative version of Quicksort (i-Q), implemented as described in [3];
- the Quicksort algorithm (Q), implemented with bounded stack usage, as suggested in [5];
- the very efficient LEDA [12] version of clever-Quicksort (c-Q), where the median of three elements is used as pivot.

Our implementations have been developed in standard C (GNU C compiler ver. 2.7) and all experiments have been carried out on a PC Pentium (133 MHz) 32MB RAM with the Linux 2.0.36 operating system.

The choice to use C, rather than C++ extended with the LEDA library is motivated by precise technical reasons. In order to get running-times independent of the implementation of the data type <array> provided by the LEDA library, we preferred to implement all algorithms by simply using C *arrays*, and accordingly by suitably rewriting the source code supplied by LEDA for Quicksort.

Observe that all implementation tricks, as well as the various policies to choose the pivot, used for Quicksort can be applied to QuickHeapsort too.

For each size $n = 10^i; i = 1::6$, a xed sample of 100 input arrays has been given to each sorting algorithm; each array in such a sample is a randomly generated permutation of the keys $1::n$. For each algorithm, the average number of key comparisons executed, $E[C_n]$, is reported together with its relative standard deviation, $[C_n]=n$, normalized with respect to n. Analogously, $E[A_n]$ and $[A_n]=n$ refers to the number of element moves. Experimental results are shown in Table 1.

They con rm pretty well the theoretical results hinted at in the previous section. Notice that most of the numbers quoted in Table 1 about Heapsort and Quicksort are in perfect agreement with the detailed experimental study of Moret and Shapiro [14].

We are mainly interested in the number of key comparisons since these represent the dominant cost, in terms of running-times, in any reasonable implementation. Observe that, in agreement with intuition, the improvement of c-Q relative to Q (in terms of number of key comparisons) is more sensible than that of c-QH relative to QH. With the exception of BU, when n is large enough c-QH executes the smallest number of key comparisons, on the average; moreover, according to theoretical results, QH always beat both Q and i-Q. It is also interesting to note that H and BU are very stable, in the sense that they present a small variance of the number of key comparisons.

In Table 2, we report the average running times required by each algorithm to sort a xed sample of 10 randomly chosen arrays of size $n = 10^i$, with $i = 4::6$.

Such results depend on the data type of the keys to be ordered (integer or double) and the type of comparison operation used (either built-in or via a user-de ned function *cmp*). In particular, six di erent cases are considered. In the rst two cases, the comparison operation used is the built-in one. In the third and fourth case, a simple comparison function cmp_1 is used. Finally, in the last two cases, two computationally more expensive comparison functions cmp_2 and cmp_3 are used (but only with keys of type integer), to simulate situations in which the cost of a comparison operation is much higher than that of an element move.[5] The function cmp_2 (resp. cmp_3) has been obtained from a simple function

[5] For instance, such situations arise when the array to sort contains pointers to the actual records. A move is then just a pointer assignment, but a comparison involves at least one level of indirection, so that comparisons become the dominant factor.

	$n = 10$				$n = 10^2$			
	$E[C_n]$	$[C_n]{=}n$	$E[A_n]$	$[A_n]{=}n$	$E[C_n]$	$[C_n]{=}n$	$E[A_n]$	$[A_n]{=}n$
H	39	(.21)	73	(.17)	1030	(.08)	1078	(.07)
BU	35	(.21)	73	(.17)	709	(.08)	1078	(.07)
i-Q	63	(.96)	43	(.64)	990	(.61)	685	(.26)
Q	41	(.63)	27	(.32)	868	(.64)	500	(.19)
c-Q	28	(.19)	37	(.53)	638	(.29)	617	(.20)
QH	39	(.48)	54	(.39)	806	(.58)	847	(.23)
c-QH	29	(.20)	60	(.51)	714	(.22)	870	(.22)

	$n = 10^3$				$n = 10^4$			
	$E[C_n]$	$[C_n]{=}n$	$E[A_n]$	$[A_n]{=}n$	$E[C_n]$	$[C_n]{=}n$	$E[A_n]$	$[A_n]{=}n$
H	16848	(.031)	14074	(.024)	235370	(.010)	174198	(.007)
BU	10422	(.021)	14074	(.024)	137724	(.006)	174198	(.007)
i-Q	14471	(.605)	9146	(.106)	194279	(.878)	114419	(.092)
Q	13297	(.609)	7285	(.095)	179948	(.654)	95807	(.072)
c-Q	10299	(.355)	8543	(.102)	142443	(.401)	109141	(.065)
QH	11881	(.630)	11838	(.202)	152789	(.664)	152155	(.201)
c-QH	11135	(.333)	11959	(.182)	146643	(.323)	152909	(.121)

	$n = 10^5$				$n = 10^6$			
	$E[C_n]$	$[C_n]{=}n$	$E[A_n]$	$[A_n]{=}n$	$E[C_n]$	$[C_n]{=}n$	$E[A_n]$	$[A_n]{=}n$
H	3019638	(.0031)	2074976	(.0025)	36793760	(.0010)	24048296	(.0008)
BU	1710259	(.0024)	2074976	(.0025)	20401466	(.0007)	24048296	(.0008)
i-Q	2421867	(.7037)	1374534	(.0689)	28840152	(.6192)	16068733	(.0649)
Q	2249273	(.6828)	1189502	(.0726)	27003832	(.5389)	14212076	(.0635)
c-Q	1816706	(.3367)	1328265	(.0546)	22113966	(.2962)	15649667	(.0497)
QH	1869769	(.6497)	1854265	(.2003)	21891874	(.6473)	21901092	(.1853)
c-QH	1799240	(.3254)	1866359	(.1675)	21355988	(.3282)	21951600	(.1678)

Table 1. Average number of key comparisons and element moves (sample size = 100).

cmp_1 by adding one call (resp. two calls) to the function log of the C standard mathematical library.

For each case considered, an approximation of the average times required by a single key comparison t_c and by a single element move t_m is also reported.

Table 2 con rms the good behaviour of all Quicksort variants i-Q, Q, c-Q; moreover, we can see that BU su ers from higher overhead due to internal bookkeeping. In most cases the running-times of QH and c-QH are between those of the variants of Heapsort, H and BU, and those of the variants of Quicksort, Q, i-Q, and c-Q.

For each trial, the best running time (represented as boxed value in Table 2) is always achieved by a \clever" algorithm, namely either c-Q or c-QH. In particular, when each key comparison operation is computationally expensive, c-QH turns out to be the best algorithm, on the average, in terms of running times.

	Integer			Double			cmp_1 (Double)		
	t_c = 0:05 sec , t_m = 0:06 sec			t_c = 0:05 sec , t_m = 0:07 sec			t_c = 0:3 sec , t_m = 0:07 sec		
n =	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
H	0.05	0.75	12.37	0.10	1.40	20.35	0.14	1.96	27.28
BU	0.09	1.25	18.80	0.13	1.83	25.88	0.15	2.00	27.62
i-Q	0.04	0.40	4.81	0.07	0.80	9.74	0.10	1.25	15.10
Q	0.03	0.37	4.54	0.06	0.74	8.92	0.09	1.12	13.49
c-Q	$\boxed{0.03}$	$\boxed{0.33}$	$\boxed{4.08}$	$\boxed{0.05}$	$\boxed{0.69}$	$\boxed{8.34}$	$\boxed{0.08}$	$\boxed{0.99}$	$\boxed{11.99}$
QH	0.05	0.64	10.43	0.08	1.10	16.85	0.10	1.42	19.96
c-QH	0.04	0.65	10.48	0.08	1.11	16.92	0.10	1.38	20.05

	cmp_1 (Integer)			cmp_2 (Integer)			cmp_3 (Integer)		
	t_c = 0:19 sec , t_m = 0:06 sec			t_c = 2:9 sec , t_m = 0:06 sec			t_c = 4:7 sec , t_m = 0:06 sec		
n =	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
H	0.10	1.35	19.16	0.54	7.04	88.29	1.00	12.92	159.66
BU	0.11	1.52	21.23	0.37	4.72	58.94	0.64	8.09	98.42
i-Q	0.07	0.85	10.12	0.42	5.44	64.96	0.79	10.24	120.69
Q	0.07	0.80	9.60	0.40	4.93	59.53	0.74	9.20	110.29
c-Q	$\boxed{0.05}$	$\boxed{0.68}$	$\boxed{8.30}$	0.35	4.42	53.82	0.64	8.22	99.24
QH	0.08	0.99	13.99	0.37	4.57	54.57	0.66	8.17	95.11
c-QH	0.07	0.98	13.98	$\boxed{0.34}$	$\boxed{4.22}$	$\boxed{52.15}$	$\boxed{0.61}$	$\boxed{7.63}$	$\boxed{91.58}$

Table 2. Average running times in seconds (sample size = 10).

Cache performance has considerably less influence on the behaviour of sorting algorithms than does paging performance (cf. [14], Chap. 8); for such reason, we believe that we can ignore completely possible negative e ects due to caching.

Concerning virtual memory problems, i.e. demand paging, all Quicksort algorithms show good locality of reference, whereas Heapsort algorithms, and also QuickHeapsort algorithms, tend to use pages that contain the top of the heap heavily, and to use in a random manner pages that contain the bottom of the heap (cf. [14]). Such observation allows us to conclude that an execution of c-Q cannot be more penalized than an execution of c-QH by delays due to paging problems. Hence, we can reasonably conclude that the success of c-QH is not due to paging performance.

5. Conclusions

We presented QuickHeapsort, a new practical \in-place" sorting algorithm obtained by merging some characteristics of Bottom-Up-Heapsort and Quicksort. Both theoretical analysis and experimental tests con rm the merits of QuickHeapsort.

The experimental results obtained show that it is convenient to use clever-QuickHeapsort when the input size n is large enough and each key comparison operation is computationally expensive.

Acknowledgments

The authors wish to thank Micha Hofri for helping them in solving the recurrence equation reported in Lemma 4.

References

[1] S. Carlsson, *A variant of heapsort with almost optimal number of comparisons,* Information Processing Letters, Vol. 24, pp. 247-250, 1987.
[2] E.E. Doberkat, *An average analysis of Floyd's algorithm to construct heaps,* Information and Control, Vol.61, pp.114-131, 1984.
[3] B. Durian, *Quicksort without a stack,* Lect. Notes Comp. Sci. Vol. 233, pp.283-289, Proc. of MFCS 1986.
[4] R.W. Floyd, *Treesort 3 (alg. 245),* Comm. of ACM, Vol. 7, p. 701,1964.
[5] G. Gonnet, R. Baeza-Yates, *Handbook of algorithms and data structures,* Addison-Wesley, Reading, MA, 1991.
[6] G. Gonnet, J. Munro, *Heaps on Heaps,* Lect. Notes Comp. Sci. Vol. 140, Proc. of ICALP'82, 1982.
[7] C.A.R. Hoare, *Algorithm 63(partition) and algorithm 65(nd),* Comm. of ACM, Vol. 4(7), pp. 321-322, 1961.
[8] M. Hofri, *Analysis of Algorithms: Computational Methods & Mathematical Tools,* Oxford University Press, New York, 1995.
[9] J. Katajainen, *The Ultimate Heapsort,* DIKU Report 96/42, Department of Computer Science, Univ. of Copenhagen, 1996.
[10] J. Katajainen, T. Pasanen, J. Tehuola, *Top-down not-up heapsort,* Proc. of The Algorithm Day in Copenhagen, Dept. of Comp. Sci., University of Copenhagen, pp. 7-9, 1997.
[11] D.E. Knuth, *The Art of Computer Programming,* Volume 3: Sorting and Searching, Addison-Wesley, 1973.
[12] LEDA, *Library of E cient Data structures and Algorithms,* http://www.mpi-sb.mpg.de/LEDA/leda.html.
[13] C.J. McDiarmid, B.A. Reed, *Building Heaps Fast,* Journal of algorithms, Vol. 10, pp. 352-365, 1989.
[14] B.M.E. Moret, H.D. Shapiro, *Algorithms from P to NP,* Volume 1: Design and E ciency, The Benjamin Cummings Publishing Company, 1990.
[15] T. Pasanen, *Elementary average case analysis of Floyd's algorithms to construct heaps,* TUCS Technical Report N. 64, 1996.
[16] L. Rosaz, *Improving Katajainen's Ultimate Heapsort,* Technical Report N.1115, Laboratoire de Recherche en Informatique, Universite de Paris Sud, Orsay, 1997.
[17] R. Sedgewick, *Quicksort,* Garland Publishing, New York, 1980.
[18] R. Sedgewick, *Implementing quicksort programs,* Comm. of ACM 21(10) pp.847-857, 1978.
[19] I. Wegener, *Bottom-Up-Heapsort, a new variant of Heapsort beating, on an average, Quicksort (if n is not very small),* Theorical Comp. Sci., Vol. 118, pp. 81-98, 1993.
[20] I. Wegener, *The worst case complexity of McDiarmid and Reed's variant of Bottom-Up heap sort is less than nlogn+1.1n,* Information and Computation, Vol. 97, pp. 86-96, 1992.
[21] J.W. Williams, *Heapsort (alg.232),* Comm. of ACM, Vol. 7, pp. 347-348, 1964.

Triangulations without Minimum-Weight Drawing[1]

Cao An Wang[2], Francis Y. Chin[3], and Boting Yang[2]

[2] Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1B 3X5
wang@garfield.cs.mun.ca
[3] Department of Computer Science and Information Systems, The University of Hong Kong, Pokfulam Road, Hong Kong
chin@csis.hku.hk

Abstract. It is known that some triangulation graphs admit straight-line drawings realizing certain characteristics, e.g., greedy triangulation, minimum-weight triangulation, Delaunay triangulation, etc.. Lenhart and Liotta [12] in their pioneering paper on "drawable" minimum-weight triangulations raised an open problem: 'Does every triangulation graph whose skeleton is a forest admit a minimum-weight drawing?' In this paper, we answer this problem by disproving it in the general case and even when the skeleton is restricted to a tree or, in particular, a star.

Keywords: Graph drawing, Minimum-weight triangulation.

1 Introduction

Drawing of a graph on the plane is a pictorial representation commonly used in many applications. A "good" graph drawing has some basic characteristics [4], e.g., planarity, straight-line edges, etc. One of the problems facing graph drawing is where to place the graph vertices on the plane, so as to realize these characteristics. For example, the problem of *Euclidean minimum spanning tree* (MST) *realization* is to locate the tree vertices such that the minimum spanning tree of these vertices is isomorphic to the given tree. However, not all trees have a MST realization, it can be shown easily that there is no MST realization of any tree with a vertex of degree 7 or more. In fact, the MST realization of a tree with maximum vertex degree 6 is NP-complete [6].
Recently, researchers have paid a great deal of attention to the graph drawing of certain triangulations. A planar graph *G=(E,V)* is a *triangulation*, if all faces of *G* are bounded by exactly three edges, except for one which may bounded by more than

[1]This work is supported by NSERC grant OPG0041629 and RGC grant HKU 541/96E.

three edges, and this face is called the *outerface*. A *minimum-weight triangulation realization* of *G=(E,V)* is to place *V* in the plane so that the minimum weight triangulation of V, (MWT(V)), is isomorphic to *G*. An excellent survey on drawability and realization for general graphs can be found in [2] and a summary of results related to our work can be found in the following table.

	GRAPH	REALIZATION	RESULT
1	Planar Graph	Straight-line drawing	Always possible [8]
2	Tree	Minimum spanning tree	Maximum vertex degree ≤ 5 polynomial time = 6 NP-complete [6] > 6 non-drawable [15]
3	Triangulation	Delaunay triangulation	Drawable and non-drawable conditions [5]
4	Maximal Outerplanar Graph	Minimum-weight triangulation	Linear time algorithm and non-drawable condition [11]
5	Triangulation	Minimum-weight triangulation	Non-drawable condition [12]
6	Maximal Outerplaner Graph	Maximum-weight triangulation	Non-drawable condition [17]
7	Caterpillar Graph	Inner edges of Maximum-weight triangulation of a convex point set	Linear time [17]

Table 1

(1) Every planar graph has a straight-line drawing realization [8].
(2) Monma and Suri [15] showed that a tree with maximum vertex degree of more than six does not admit a straight-line drawing of minimum spanning tree. Eades and Whitesides [6] proved that the realization of Euclidean minimum spanning trees of maximum vertex degree six is NP-hard.
(3) Dillencourt [5] presented a necessary condition for a triangulation admitting a straight-line drawing of Delaunay triangulation and also a condition for non-drawability.
(4) Lenhart and Liotta [11] studied the minimum-weight drawing for a maximal outerplanar graph, and discovered a characteristic of the minimum-weight triangulation of a regular polygon using the combinatorial properties of its dual trees. With this characteristic, they devised a linear-time algorithm for the drawing.
(5) Lenhart and Liotta [12] further demonstrated some examples of 'non-drawable' graphs for minimum-weight realizations, and also proved that if any graph contains such non-drawable subgraph, then it is not *minimum-weight drawable*.
(6) Wang, et. al. studied the maximum-weight triangulation and graph drawing, a simple condition for non-drawability of a maximal outerplanar graph is given in [17].

(7) A *caterpillar* is a tree such that all internal nodes connect to at most 2 non-leaf nodes. Wang, et. al. [17] showed that caterpillars are always linear-time realizable by the inner edges of maximum-weight triangulation of a convex point set.

In this paper, we investigate the open problem raised by Lenhart and Liotta [12] to determine whether or not every triangulation whose 'skeleton' is a forest admits a minimum-weight drawing. The *skeleton* of a triangulation graph is the remaining graph after removing all the boundary vertices and their incident edges on the outerface. Intuitively, the answer to this open problem seems to be affirmative by adapting the same idea in the drawing of wheel graphs or k-spined graphs [12]. That is, one can stretch the vertices of a tree in the forest-skeleton arbitrarily far apart from each other as well as from other trees. In this manner, all the vertices in the forest-skeleton would be "isolated" from each other. The edges of the trees would be minimum-weight and the edges connecting the removed vertices would also be minimum-weight in hoping that the "long distance" will make such a localization. However, this intuition turns out to be false as the removed part of the graph plays an indispensable role in the MWT. As matter of a fact, there exist some minimum-weight non-drawable triangulations whose skeleton is a forest or a tree. It is worth noting that the proof of some graphs being 'non-drawable' is similar to the proof of a lower bound of a problem, which requires some non-trivial observation. In Section 3, we derive a combinatorial non-drawability sufficient condition for any minimum weight triangulation. Then we apply this condition to show that some triangulations with forest skeletons are not minimum-weight drawable. In Section 4, we further disprove the conjecture by showing the existence of a tree-skeleton triangulation, in particular, a star-skeleton triangulation which is not minimum-weight drawable. In Section 5, we conclude our work.

2 Preliminaries

Definition 1: Let S be a set of points in the plane. A *triangulation* of S, denoted by $T(S)$, is a maximal set of non-crossing line segments with their endpoints in S.
The weight of a triangulation $T(S)$ is given by $\omega(T(S)) = \sum_{\overline{s_i s_j} \in T(S)} d(s_i s_j)$, where $d(s_i s_j)$ is the Euclidean distance between s_i and s_j of S. A *minimum-weight triangulation* of S, denoted by $MWT(S)$, is defined as, for all possible $T(S)$, $\omega(MWT(S)) = min\ \{\omega(T(S))\}$. ❑

Property 1: (Implication property)

A triangulation $T(S)$ is called *k-gon local minimal* or simply *k-minimal*, denoted by $T_k(S)$, if any k-gon extracted from $T(S)$ is a minimum-weight triangulation for this k-gon. Let '$a \succ b$' denote 'a implies b' and *a contains b*. Then following implication property holds:

$$MWT(S) \succ T_{n-1}(S) \succ \cdots \succ T_4(S) \succ T(S).$$ ❑

Figure 1(a) illustrates an example which is 4-minimal but not 5-minimal. Note in the figure that every quadrilateral has a minimum-weight triangulation but not the pentagon *abdef*. So Figure 1(a) is a T_4 but not T_5 nor *MWT*. On the other hand, Figure 1(b) gives the *MWT* of the same vertex set.

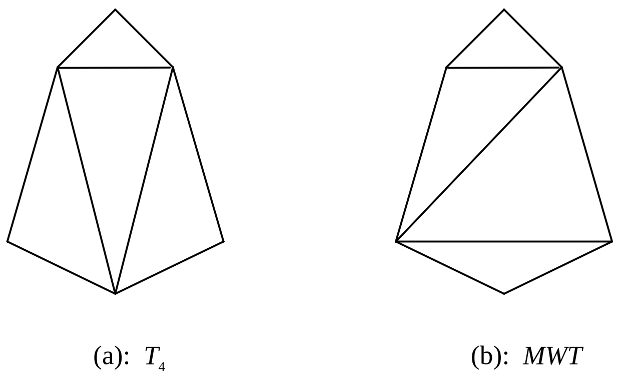

Figure 1: 4-minimal but not minimum

Definition 2: Let *e* be an internal edge of any triangulation. Then, *e* is a *diagonal* of a quadrilateral inside the triangulation, say *abcd* with *e* = (*a*, *c*). Angles $\angle abc$ and $\angle cda$ are called *facing angles* w.r.t. *e*. Note that each internal edge of a triangulation has exactly two facing angles (Figure 2). ❑

Property 2: Let Δabc be a triangle in the plane and *d* be an internal vertex in Δabc. Then, at most one of $\angle adb$, $\angle bdc$, and $\angle cda$ can be acute. ❑

Lemma 1: *Let abcd denote a quadrilateral with diagonal (a, c) and with two obtuse facing angles, $\angle abc$ and $\angle cda$. If such a quadrilateral always exists in any drawing of a given triangulation, then this triangulation is minimum-weight non-drawable, in particular, 4-minimal non-drawable.*

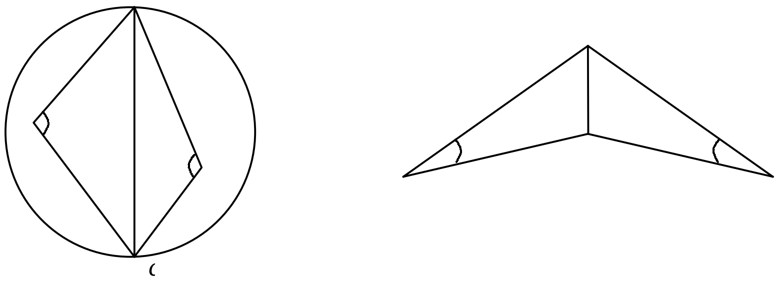

Figure 2: For the proof of Lemma 1

Proof: Since both facing angles $\angle abc$ and $\angle cda$ of edge (a, c) are greater than 90°, the quadrilateral *abcd* must be convex (refer to Figure 2). Then, edge (b, d) lies inside *abcd* and $(b, d) < (a, c)$, quadrilateral *abcd* is not 4-minimal. Since such a quadrilateral always exists in any drawing of the triangulation by the premise, the triangulation is not a 4- minimal, nor minimum-weight drawable. ❑

3 Forest-Skeleton Triangulations

In this section, we shall give a combinatorial sufficient condition for a triangulation to be minimum-weight non-drawable. With the condition, we can prove that there exists a forest-skeleton which is minimum-weight non-drawable, thus, disprove the conjecture by Lenhart and Liotta [12].

3.1 Non-drawable Condition for Minimum-Weight Triangulations

In the following, we shall provide a combinatorial sufficient condition for a triangulation to be 4-minimal non-drawable.

N4o-Condition: Let *G* be a triangulation such that

(1) *G* contains a simple circuit *C* with non-empty set *V* of internal vertices.

(2) Inside *C*, let *V'* denote the subset of *V* such that each element in *V'* is of degree three; each element in $V'' = V - V'$ is of degree more than three; and let *f* be the number of faces after the removal of vertices in *V'* and their incident edges. Then, *G* satisfies the following conditions:

(i) $|V''| > 1$, and

(ii) $f < |V'| + (|V''| - 1)/2$. ❑

It is easy to see that no two vertices in *V'* are adjacent to each other and thus $|V'| \leq f$. Figure 3 gives a subgraph which satisfies the N4o-Condition, $|V'|=11$, $|V''|=4$, and $f = 12 < |V'| + (|V''| - 1)/2 = 12.5$.

Lemma 2: *Let G be a triangulation. If G satisfies the N4o-Condition, then G is 4-minimal non-drawable.*

Proof: Let G_c denote the portion of *G* enclosed by *C*. Let G'_c denote the remaining graph of G_c after the removal of *V'* (the vertices of degree three) as well as their incident edges. In G'_c, let *f, e,* and *n* denote the number of faces, the number of edges, and the number of vertices, respectively; let *f'* denote the number of faces originally not containing any vertex of *V'*; let *e'* denote the number of edges not lying on C; and let n_c denote the number of vertices on C. Since all faces in G_c are triangles, we have $f = \dfrac{2(e-n_c)+n_c}{3}$. Together with Euler formula on G_c, $f - e + n = 1$, we have that

$$e = 3n - 3 - n_c\,, \quad f = 2n - 2 - n_c \,\ldots\ldots\ldots\ (1).$$

By part (ii) of N4o-Condition, $f < |V'| + (|V''| - 1)/2$. As $f = f' + |V'|$, $f' + |V'| < |V'| + (|V''| - 1)/2$. Then, $f' < (|V''| - 1)/2$, or $2f' < |V''| - 1$. Note that $|V''| = n - n_c$, we have

$$2f' < n - n_c - 1 \ \ldots\ldots\ (2).$$

As $e - n_c = e'$, we have by (1) and (2) that

$$f + 2f' < e' \ \ldots\ldots\ldots\ (3).$$

Note that an edge of G'_c not on C can be regarded as the diagonal of a quadrilateral in G_c. As every diagonal has two facing angles and G'_c contains e' internal edges, there are exactly $2e'$ facing angles. Moreover, G'_c contains f faces, $f - f'$ of them have a white node inside and thus each of these faces contributes at most one acute facing angle (Property 2). On the other hand, each of the f' faces contributes at most three acute facing angles. Thus, the total number of acute facing angles for these e' interior edges in G_c is at most $f - f' + 3f'$ (= $f' + 2f'$). By (3), the number of internal edges is greater than the number of acute facing angles in G_c. Thus, at least one of the e' internal edges (diagonals) is not associated with an acute facing angle and must have two obtuse facing angles. By Lemma 1, G_c cannot admit a 4-minimal drawing. ❑

By Property 1 and Lemma 2, we have

Theorem 1: *Let G be a triangulation. If G satisfies the N4o-Condition, then G is minimum-weight non-drawable.* ❑

Note that Theorem 1 is applicable to any triangulation G_T by treating the hull of G_T as the circuit C stated in the **N4o-Condition**. Refer to Figure 3.

3.2 A Minimum-Weight Non-drawable Example for a Forest-Skeleton Triangulation

We shall construct a triangulation whose skeleton is a forest and which satisfies the **N4o-Condition**. Then, the non-drawable claim follows from Theorem 1, which answers the open problem that not all triangulations with a forest-skeleton are minimum-weight drawable.

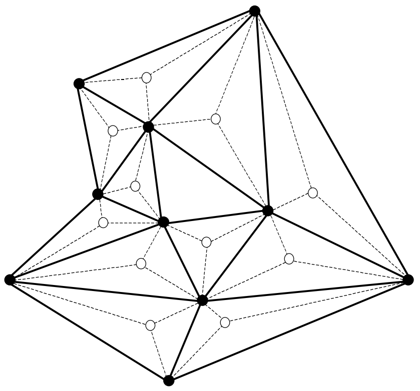

Figure 3: An example of a 4-optimal non-drawable triangulation. The darken vertices are of degree more than three and the white vertices are of degree three. The sizes of V' and V" are 11 and 4 respectively, n = 10, e = 21, f = 12, n_c = 6, f' = 1, e' = 15.

Theorem 2: *There exists a triangulation with a forest-skeleton which is not minimum-weight drawable.*

Proof: The triangulation shown in Figure 4 has a forest-skeleton (the darken edges). It contains a simple cycle $C_G = (v_1, v_2, v_3, v_4)$. Inside C_G, $|V''| = 2$, i.e., $\{v_{11}, v_{12}\}$; $|V'| = 6$, i.e., $\{v5, v_6, v_7, v_8, v_9, v_{10}\}$; $f = 6$ (the number of faces after the removal of V'). Thus, part (1) of **N4o-Condition**: $|V''| > 1$ is satisfied and part (2) of **N4o-Condition**: $f < |V'| + (|V''| - 1)/2$ is also satisfied. Then, G is not minimum-weight drawable by Theorem 1. ❑

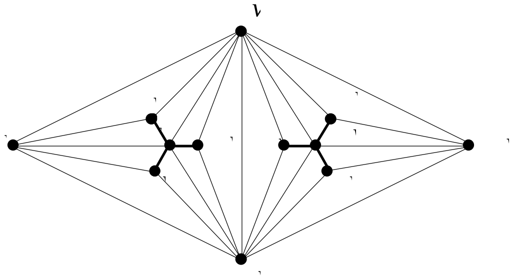

Figure 4: A non-drawable forest-skeleton triangulation

4 Tree-Skeleton Triangulation

In this section, we shall show that there exist tree-skeleton triangulations which are minimum-weight non-drawable further disproving the claim in [12]. Let us consider a triangulation in the plane with two adjacent triangles, Δ*abc* and Δ*bcd*, each of which has an internal vertex with degree 3, as shown in Figure 5. As agreed previously, each of the internal degree-3 vertices can contribute at most one acute angle. In the following, we shall prove that if the only acute angle in Δ*abc*, ∠*aeb*, and that in Δ*cbd*, ∠*bfd*, are facing edge (*a,b*) and edge *(b,d)* respectively, then by Lemma 1, the triangulation with these two adjacent triangles is not minimum-weight drawable. This is because *e* and *f* will be on the quadrilateral *bfce* and edge *(e,f)* crosses edge *(b,c)* and is shorter than edge *(b,c)*.

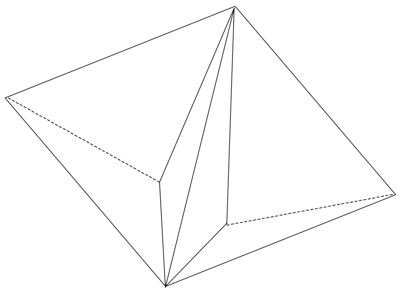

Figure 5: An non-drawable case

Let us consider a convex polygon *P* with $n \geq 13$ vertices. We shall show that *P* has at least 3 consecutive inner angles with degree > 90°.

Lemma 3: *Any convex polygon cannot have more than 4 acute inner angles.*

Proof: If the convex *n*-gon has 5 or more acute angles, then the sum of angles is no more than $180°(n\text{-}5) + 5 \times 90° = 180°\,n - 900° + 450° = 180°\,n - 450° = 180°(n\text{-}2) - 90° < 180°(n\text{-}2)$. This contradicts the fact that the sum of inner angles of a convex *n*-gon must be $180°(n\text{-}2)$. ❑

Lemma 4: *If P is a convex polygon with* $n \geq 13$ *vertices, then P has at least 3 consecutive obtuse inner angles.*

Proof: The proof is by contradiction. Assume *P* has at most 2 consecutive obtuse inner angles, then there must exist at least 5 acute inner angles to separate the other obtuse inner angles in *P* for $n \geq 13$. This contradicts Lemma 3. ❑

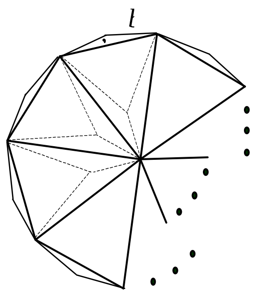

Figure 6: A non-drawable tree-/star-skeleton triangulation

Theorem 3: *There exists a tree-skeleton (star-skeleton) triangulation which does not admit a minimum-weight drawing.*

Proof: Refer to Figure 6. By Lemma 4, *P* contains at least three consecutive obtuse inner angles. Without loss of generality, let $\angle a'$, $\angle b'$, and $\angle a'bb'$ be three consecutive obtuse inner angles of *P*. In order for P to be drawable, the region *caa'bb'd* must also be drawable. It follows that edges *(a,b), (b,d),* and *(b,c)* must be drawable. Note that $\angle aeb$ and $\angle bfd$ must be acute since $\angle a'$ and $\angle b'$ are already obtuse. Then, the angle $\angle bec$ in triangle Δabc and the angle $\angle bfc$ in triangle Δbcd must be obtuse by Property 2. Then, edge *(b,c)* cannot be an edge in any minimum-weight drawing. As the graph is a star-skeleton triangulation and star is a subclass of tree, the theorem also applies to the tree-skeleton triangulations. ❑

5 Conclusion

In this paper, we investigated the minimum-weight drawability of triangulations. We show that triangulations with forest-skeletons or even with tree-/star-skeleton are not minimum-weight drawable, disproving the conjecture by [12]. Furthermore, we found that in addition to wheel graph and k-spined graph, a subclass of star-skeleton graph, regular star-graph is minimum-weight drawable. It will be sketched out in the following appendix.

Appendix: Drawable Triangulation with Minimum-Weight

In this section, we shall show that some special triangulation, namely, regular star skeleton graph, admits a minimum-weight drawing.

Definition 3: There exist only three types of edges in a triangulation whose skeleton is a forest , namely, (1) skin-edge (simply, s-edge): both vertices of the edge are on the hull, (2) tree-edge (simply, t-edge): both vertices of the edge are not on the hull, and (3) bridge-edge (simply, b-edge): one vertex of the edge is on the hull and the other is not. A *base* skin-edge is the most `inner' layer of s-edges. A graph *G* is a regular star skeleton graph, denoted by *RSSG*, if *G* has a star skeleton, *G* contains only base skin, and all the b-edges of a branch on the same side are connected to the vertex of its neighboring b-edge.

By the definition, an *RSSG* can always be decomposed into a wheel and several k-spined triangles, where *k* can be different for different triangles. Each k-spined triangle consists of two fans, and the apex of a fan is a vertex of b-edge in the wheel and the boundary of the fan is a branch of the star-skeleton. We shall give a high-level description of the algorithm.

Algorithm: The algorithm first identifies if *G* is an *RSSG*. If it is, then label the b-edges and t-edges for the wheel and the fans of this *RSSG*. For a given resolution of the drawing, we can determine the size of a fan that is a function of the number of radial edges, *k*, and the distance δ between the apex and its closest boundary vertex. Now, the algorithm will draw a wheel. During the arrangement of its radial edges, we take the size of the attached fans into a count. There are two types of drawings for fans: FAN1 and FAN2.
FAN1: Let *v* be the apex of a fan and $(v_1, v_2, ..., v_k)$ be the sequence of vertices on its boundary (the interior vertices of the (k-1)-spined triangle. The drawing is similar to that in [12] (refer to Lemma 7 of [12]).
FAN2: The apex *v*' lies on the opposite side of the apex *v* along $(v_1, v_2, ..., v_k)$. Since all the edges in FAN2 are stable edges, they belong to *MWT(S)* [15].

Theorem 4. *Graph RSSG is minimum-weight drawable.*

Proof Sketch: We shall prove that each edge of the drawing belongs to the *MWT* of this point set. There are three types of edges in the drawings: the base s-edges, the b-edges, and the t-edges. The s-edges obviously belong to the *MWT* of this point set since all these edges are on the convex hull of this point set. Let us consider b-edges and t-edges. By our algorithm and by Lemma 7 in [12], all individual fans belong to their own *MWT*s respectively. We can show that they also belong to the final *MWT* by proving all the b-edges separating them belong to *MWT(S)* (using the local replacing argument [16]).

References

[1] Bose P., Di Battista G., Lenhart W., and Liotta G., Constraints and representable trees, *Proceedings of GD94*, LNCS 894, pp. 340-351.

[2] Di Battista G., Lenhart W., and Liotta G., Proximity drawability: a survey, *Proceedings of GD94*, LNCS 894, pp. 328-339.

[3] Di Battista G., Eades P., Tamassia R., and Tollis I.G., Algorithms for automatic graph drawing: an annotated bibliography, *Computational Geometry: Theory and Applications*, **4**, 1994, pp. 235-282.

[4] Di Battista G., Eades P., Tamassia R., and Tollis I.G., *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999 (ISBN: 0-13-301615-3).

[5] Dillencourt M., Toughness and Delaunay triangulations, *Discrete and Computational Geometry*, **5,** 1990, pp. 575-601.

[6] Eades P. and Whitesides S., The realization problem for Euclidean minimum spanning tree is NP-hard, *Proceedings of 10th ACM Symposium on Computational Geometry*, Stony Brook, NY, 1994, pp. 49-56.

[7] ElGindy H., Liotta G., Lubiw A., Mejier H., and Whitesides S., Recognizing rectangle of influence drawable graphs, *Proceedings of GD94*, LNCS 894, pp. 252-263.

[8] Fary I., On straight lines representations of planar graphs, *Acta Sci. Math.*, Szeged, **11**, 1948, pp. 229-233.

[9] Gimikowski R, Properties of some Euclidean proximity graphs, *Pattern Recognition letters*, **13**, 1992, pp. 417-423.

[10] Keil M., Computing a subgraph of the minimum-weight triangulation, *Computational Geometry: Theory and Applications*, **4**, 1994, pp. 13-26.

[11] Lenhart W. and Liotta G., Drawing outerplanar minimum-weight triangulations, *Information Processing Letters*, **57**, 1996, pp. 253-260.

[12] Lenhart W. and Liotta G., Drawable and forbidden minimum-weight triangulations, *Proceedings of GD97*, LNCS 894, pp. 1-12.

[13] Matula D. and Sokal R., Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane, *Geographical Analysis*, **12**(3), 1980, pp. 205-222.

[14] Monma C. and Suri S., Transitions in geometric minimum spanning trees, *Proceedings of 7th ACM Symposium on Computational Geometry*, North Conway, NH, 1991, pp. 239-249.

[15] Preparata F. and Shamos M., *Computational Geometry*, 1985, Springer-Verlag.

[16] Wang C.A., Chin F., and Xu Y., A new subgraph of minimum-weight triangulations, *Journal of Combinatorial Optimization*, 1997, pp. 115-127.

[17] Wang C.A., Chin F., and Yang B.T., Maximum Weight Triangulation and Graph Drawing, *Information Processing Letters*, **70**(1), 1999, pp. 17-22.

Faster Exact Solutions for Max2Sat

Jens Gramm and Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
gramm,niedermr@informatik.uni-tuebingen.de

Abstract. Given a boolean 2CNF formula F, the Max2Sat problem is that of nding the maximum number of clauses satis able simultaneously. In the corresponding decision version, we are given an additional parameter k and the question is whether we can simultaneously satisfy at least k clauses. This problem is NP-complete. We improve on known upper bounds on the worst case running time of Max2Sat, implying also new upper bounds for Maximum Cut. In particular, we give experimental results, indicating the practical relevance of our algorithms.

Keywords: NP-complete problems, exact algorithms, parameterized complexity, Max2Sat, Maximum Cut.

1 Introduction

The (unweighted) *Maximum Satis ability* problem (MaxSat) is to assign values to boolean variables in order to maximize the number of satis ed clauses in a CNF formula. Restricting the clause size to two, we obtain Max2Sat. When turned into \yes{no" problems by adding a goal k representing the number of clauses to be satis ed, MaxSat and Max2Sat are *NP*-complete [7]. E cient algorithms for MaxSat, as well as Max2Sat, have received considerable interest over the years [2]. Furthermore, there are several papers which deal with Max2Sat in detail, e.g., [3,4,6]. These papers present approximation and heuristic algorithms for Max2Sat. In this paper, by way of contrast, we introduce algorithms that give optimal solutions within provable bounds on the running time. The arising solutions for Max2Sat are both fast and exact and show themselves to be interesting not only from a theoretical point of view, but also from a practical point of view due to the promising experimental results we have found.

The following complexity bounds are known for Max2Sat: There is a deterministic, polynomial time approximation algorithm with approximation factor 0:931 [6]. On the other hand, unless $P = NP$, the approximation factor cannot be better than 0:955 [9]. With regard to exact algorithms, research so far has concentrated on the general MaxSat problem [1,12]. As a rule, the algorithms which are presented there (as well as our own) are based on elaborate case distinctions. Taking the case distinctions in [12] further, Bansal and Raman [1] have recently presented the following results: Let jFj be the length of the given input formula and K be the number of clauses in F. Then MaxSat can be solved in times $O(1.3413^K |F|)$ and $O(1.1058^{|F|})$. The latter result implies that, using $|F| = 2K$, Max2Sat can be solved in time $O(1.2227^K)$, this being the best known result for Max2Sat so far. Moreover, Bansal and Raman have shown that, given the number k of clauses which are to be satisfied in advance, MaxSat can be solved in $O(1.3803^k k^2 + |F|)$ time.

Our main results are as follows: Max2Sat can be solved in times $O(1.0970^{|F|})$, $O(1.2035^K)$, and $O(1.2886^k k + |F|)$, respectively. In addition, we show that if each variable in the formula appears at most three times, then Max2Sat, still NP-complete, can be solved in time $O(1.2107^k |F|)$. In reference to modifications of our algorithms done in [8], we find that Maximum Cut in a graph with n vertices and m edges can be solved in time $O(1.3197^m)$. If restricted to graphs with vertex degree at most three, it can be solved in time $O(1.5160^n)$, and, if restricted to graphs with vertex degree at most four, in time $O(1.7417^n)$. In addition, the same algorithm computes a Maximum Cut of size at least k in time $O(m + n + 1.7445^k k)$, improving on the previous time bounds of $O(m + n + 4^k k)$ [11] and $O(m + n + 2.6196^k k)$ [12].

Aside from the theoretical improvements gained by the new algorithms we have developed, an important contribution of our work is also to show the practical significance of the results obtained. Although our algorithms are based on elaborate case distinctions which show themselves to be complicated upon analysis, they are relatively easy to apply when dealing with the number of cases the actual algorithm has to distinguish. Unlike what is known for the general MaxSat problem [1,12], we thereby have for Max2Sat a comparatively small number of easy to check cases, making our implementation practical. Moreover, analyzing the frequency of how often different rules are applied, our experiments also indicate which rules might be the most valuable ones. Our algorithms can compete well with heuristic ones, such as the one described by Borchers and Furman [3].

Independent from our work, Hirsch [10] has simultaneously developed upper bounds for the Max2Sat problem. He presents an algorithm with bounds of $O(1.0905^{|F|})$ with respect to the formula length $|F|$ and $O(1.1893^K)$ with respect to the number of clauses K, which are better than the bounds shown for our algorithms. Moreover, he points out that his algorithm also works for weighted versions of Max2Sat. On the other hand, however, he does not give any bound with respect to k, the number of satisfiable clauses. His analysis is simpler than ours, as he makes use of a result by Yannakakis [15]. The algorithm itself, however, seems much more complex and is not yet accompanied by an implementation. The reduction step of Hirsch's algorithm has a polynomial complexity, as a maximum flow computation has to be done, and it would be interesting to see whether this will turn out to be efficient in practice.

Due to the lack of space, we omitted several details and refer to [8] for more material.

2 Preliminaries and Transformation Rules

We use primarily the same notation as in [1,12]. We study boolean formulas in 2CNF, represented as multisets of sets (clauses). A subformula, i.e., a subset of clauses, is denoted *closed* if it is a minimal subset of clauses allowing no variable within the subset to occur outside of this subset as well. A clause that contains the same variable positively and negatively, e.g., $\{x, \bar{x}\}$, is satis ed by every assignment. We will not allow for these clauses here, and assume that such clauses are always replaced by a special clause $\top$, denoting a clause that is always satis ed. We call a clause containing r literals simply an *r-clause*. Its *length* is therefore r. A formula in *2CNF* is one consisting of 1- and 2-clauses. We assume that 0-clauses do not appear in our formula, since they are clearly unsatis able. The *length of a clause* is its cardinality, and the *length of a formula* is the sum of the lengths of its clauses. Let l be a literal occurring in a formula F. We call it an *(i,j)-literal* if the variable corresponding to l occurs exactly i times as l and exactly j times as $\bar{l}$. Analogously, we obtain (i^+,j)-, (i,j^+)-, and *(i^+,j^+)-literals* by replacing \exactly" with \at least" at the appropriate positions, and get (i^-,j)-, (i,j^-)- and *(i^-,j^-)-literals* by replacing \exactly" with \at most". Following Bansal and Raman [1], we call an (i,j)-literal an $(i,j)[p_1,\ldots,p_i][n_1,\ldots,n_j]$-literal if the clauses containing l are of length $p_1 \geq \ldots \geq p_i$ and those containing $\bar{l}$ are of length $n_1 \geq \ldots \geq n_j$. For a literal l and a formula F, let $F[l]$ be the formula originating from F by replacing all clauses containing l with $\top$ and removing $\bar{l}$ from all clauses where it occurs. We say x occurs in a clause C if $x \in C$ or $\bar{x} \in C$. We write $\#_x$ for the number of occurrences of x in the formula. Should variable x and variable y occur in the same clause, we call this instance a *common occurrence* and write $\#_{xy}$ for the number of their common occurrences in the formula. In the same way, we write $\#_{xy}$ for the number of common occurrences of literals x and y.

As with earlier exact algorithms for MaxSat [1,12], our algorithms are recursive. They go through a number of transformations and branching rules, where the given formula is simpli ed by assigning boolean values to some carefully selected variables. The fundamental di erence between transformation and branching rules is that when the former has been given a formula, it is replaced by *one* simpler formula, whereas in the latter a formula is replaced by *at least two* simpler formulas. The asymptotic complexity of the algorithm is governed by the branching rules. We will use recurrences to describe the size of the corresponding *branching trees* created by our algorithms. Therefore, we will apply one of the transformation rules whenever possible, as they avoid a branching of recursion.

In the rest of this section, we turn our attention to the transformation rules. Our work here follows that of [12] closely, as the rst 4 rules have also been used there. Their correctness is easy to check.

1. *Pure Literal Rule:* Replace F with $F[l]$ if l is a $(1^+,0)$-literal.
2. *Dominating 1-Clause Rule:* If $\bar{l}$ occurs in i clauses and l occurs in at least i 1-clauses of F, then replace F with $F[l]$.

3. *Complementary 1-Clause Rule:* If $F = ffxg; fxgg [G$, then replace F with G, increasing the number of satis ed clauses by one.
4. *Resolution Rule:* If $F = ffx; l_1g; fx; l_2gg [G$ and G does not contain x, then replace F with $ffl_1; l_2gg [G$, increasing the number of satis ed clauses by one.
5. *Almost Common Clauses Rule:* If $F = ffx; yg; fx; ygg [G$, then replace F with $fxg [G$, increasing the number of satis ed clauses by one.
6. *Three Occurrence Rules:* We consider two subcases:
   (a) If x is a $(2; 1)$-literal, $F = ffx; yg; fx; yg; fx; ygg [G$, and G does not contain x, then replace F with G, increasing the number of satis ed clauses by three.
   (b) If x is $(2; 1)$-literal, and either $F = ffx; yg; fx; yg; fx; l_1gg [G$ or $F = ffx; yg; fx; l_1g; fx; ygg [G$, then replace F with $ffy; l_1gg [G$ or $ffy; l_1gg [G$, respectively, increasing the number of satis ed clauses by two.

The Almost Common Clauses Rule was introduced by Bansal and Raman [1]. In the rest of this paper, we will call a formula *reduced* if none of the above transformation rules can be applied to it. The correctness of many of the branching rules that we will present relies heavily on the fact that we are dealing with reduced formulas.

3 A Bound in the Number of Satis able Clauses

Theorem 1 *For a 2CNF formula F, it can be computed in time $O(jFj + 1{:}2886^k k)$ whether or not at least k clauses are simultaneously satis able.*

Theorem 1 is of special interest in so-called parameterized complexity theory [5]. The corresponding bound for formulas in CNF is $O(jFj + 1{:}3803^k k^2)$ [1]. In this expression $1{:}3803^k$ gives an estimation of the branching tree size. The time spent in each node of the tree is $O(jFj)$, which for CNF formulas is shown to be bounded by k^2[11]. For 2CNF formulas, however, we can improve this factor for every node of the tree from k^2 to k: Note that the case where $k \quad d\frac{K}{2}e$ with K as the number of clauses is trivial, since for a random assignment, either this assignment or its inverse satisfy $d\frac{K}{2}e$ clauses. For $k > d\frac{K}{2}e$, however, Max2Sat formulas have $jFj = O(k)$.

Before sketching the remaining proof of Theorem 1, we give a corollary. Consider a 2CNF input formula in which every variable occurs at most three times. This problem is also *NP*-complete [13], but we can improve our upper bounds by excluding some of the cases necessary for general 2CNF formulas, thus obtaining a better branching than in Theorem 1. We omit details.

Corollary 2 *For a 2CNF formula F where every variable occurs at most three times, it can be computed in time $O(jFj + 1{:}2107^k k)$ whether or not at least k clauses are simultaneously satis able.*

We now sketch the proof of Theorem 1. We present algorithm A with the given running time. As an invariant of our algorithm, observe that the subsequently described branching rules are only applied if the formula is reduced, that is, there is no transformation rule to apply. The idea of branching is based on dividing the search space, i.e. the set of all possible assignments, into several parts, nding an optimal assignment within each part, and then taking the best of them. Carefully selected branchings enable us to simplify the formula in some of the branches. Observe that the subsequent order of the steps is important. In each step, the algorithm always executes the applicable branching rule with the lowest possible number:

RULE 1: If there is a $(9^+; 1)$-, $(6^+; 2)$-, or $(4^+; 3^+)$-literal x, then we branch into $F[x]$ and $F[x]$. The correctness of this rule is clear. In the worst case, a $(4; 3)$-literal, by branching into $F[x]$, we may satisfy 4 clauses and by branching into $F[x]$, we may satisfy 3 clauses. We describe this situation by saying that we have a *branching vector* $(4; 3)$, which expresses the corresponding recurrence for the search tree size, solvable by standard methods (cf. [1,12]). Solving the corresponding recurrence for the branching tree size, we obtain here the *branching number* $1:2208$. This means that were we always to branch according to a $(4; 3)$-literal, the branching tree size would be bounded by $1:2208^k$. It is easy to check that branching vectors $(9; 1)$ and $(6; 2)$ yield better (i.e., smaller) branching numbers.

RULE 2: If there is a $(2; 1)$-literal x, such that $F = ffx; yg; fx; zgg [G$ and y occurs at least as often in F as z, then branch as follows: If both y and z are $(2; 1)$-literals, branch into $F[x]$ and $F[x]$. We can show a worst case branching vector of $(4; 5)$ in these situations. Otherwise, i.e., if one of y and z is not $(2; 1)$, then branch into $F[y]$ and $F[y]$. The correctness is again obvious. However, the complexity analysis (i.e., analysis of the branching vectors) is signi cantly harder in this case. Keep in mind that the formula is reduced, meaning that we may exclude all cases where a transformation rule would apply.

First, we distinguish according to the number of common occurrences of x and y: Assuming that there are three common occurrences we either have clauses $fx; yg$, $fx; yg$, clauses $fx; yg$, $fx; yg$, or clauses $fx; yg$, $fx; yg$, $fx; yg$. In the rst two cases, the Almost Common Clause Rule applies (cf. Section 2), and in the latter case, the rst of the Three Occurrence Rules applies. Analogously, assuming two common occurrences, either the Almost Common Clause Rule or the second of the Three Occurrence Rules applies. Hence, because the formula is reduced, we can neglect these cases.

It remains to consider only one common occurrence of x and y. We make the following observation: By satisfying y, we reduce literal x to occurrence two and the Resolution Rule applies, eliminating x and satisfying one additional clause. On the other hand, satisfying y leaves a unit occurrence of x and the Dominating 1-Clause Rule applies, eliminating x from the formula and satisfying the two x-clauses. Now we consider each possible occurrence pattern for literal y. If y occurs at least four times, it is a $(3^+; 1)$-, $(1; 3^+)$-, or a $(2^+; 2^+)$-literal, and using the given observation, in the worst cases we obtain branching vectors $(4; 3)$, $(2; 5)$, or $(3, 4)$. If y occurs only three times, it is a $(2, 1)$- or $(1, 2)$-literal. We then take a literal z into consideration as well. We know from the way in which y was chosen that the literal z is also of occurrence three. We consider all combinations of y and z, which are either $(2, 1)$- or $(1, 2)$, and also cover a possible common occurrence of y and z in one clause. Branching as specified, we in the worst case obtain a branching vector $(2, 6)$, namely when both y and z are $(1, 2)$ and there is no common clause of y and z. We omit the details here.

Summarizing, for RULE 2, the worst observed branching vector is $(2, 5)$, which corresponds to the branching number 1.2365.

RULE 3: If there is a $(3^+, 3^+)$- or $(4^+, 2)$-literal x, then branch into $F[x]$ and $F[x]$. Trivially we get the branching vectors $(3, 3)$ and $(4, 2)$, implying the branching numbers 1.2600 and 1.2721.

RULE 4: If there is a $(c, 1)$-literal x with $c \in \{3, 4, 5, 6, 7, 8\}$, then choose a literal y occurring in a clause $\{x, y\}$ and branch into $F[y]$ and $F[y]$. Again, this is clearly correct.

With regard to the complexity analysis, we observe that by satisfying y, a unit occurrence of x arises and the Dominating 1-Clause Rule applies, satisfying all x-clauses. Having reached RULE 4, we know that all literals in the formula occur at least four times, as the 3-occurrences are eliminated by RULE 2. We consider different possible cases for y, namely y being a $(3^+, 1)$-, $(1, 3^+)$-, or $(2^+, 2^+)$-literal, and we consider all possible numbers of common occurrences of x and y. Using the given observation, we can show a branching vector of $(1, 6)$ in the worst case, namely for a $(3, 1)$-literal x, a $(1, 3)$-literal y, and $\#_{xy} = 1$. This corresponds to the branching number 1.2852. Again, we omit the details.

RULE 5: By this stage, there remain only $(2, 2)$-, $(3, 2)$-, or $(2, 3)$-literals in the formula. RULE 5 deals with the case that there is a $(2, 2)$-literal x. Our branching rule now is more involved. We choose a literal y occurring in a clause $\{x, y\}$ and a literal z occurring in a clause $\{x, z\}$. For x having at least two common occurrences with y or z, we branch into $F[x]$ and $F[x]$. If this is not the case but y and z have at least two common occurrences, we branch into $F[y]$ and $F[y]$. It remains that $\#_{xy} = 1$, $\#_{xz} = 1$, and $\#_{yz} \leq 1$. If y and z have a common occurrence in a clause $\{y, z\}$, we branch into $F[y]$, $F[yz]$, and $F[yz]$. If not, i.e. there is no clause $\{y, z\}$, we branch into $F[yz]$, $F[yz]$, $F[yz]$, and $F[yz]$. It is easy to verify that we have covered all possible cases.

Regarding the complexity analysis, we first make use of the following: Whenever two literals being $(2, 2)$ or $(3, 2)$ have at least two common occurrences, we can take one of them and branch setting it true and false. In the worst case, this results in the branching vector $(2, 5)$ with branching number 1.2366.

Thus, we are only left with situations in which $\#_{xy} = 1$, $\#_{xz} = 1$, and $\#_{yz} \leq 1$. For these cases, we consider all arrangements of x, y and z possible, with x being $(2, 2)$, y being $(2^+, 2^+)$ and z being $(2^+, 2^+)$. We obtain “good” branching numbers of 1.2886 for vectors as, such as $(5, 6, 5, 6)$ in most cases by branching into $F[yz]$, $F[yz]$, $F[yz]$, and $F[yz]$. Only for a possible common occurrence of y and z in a clause $\{y, z\}$ would the branching number be worse. We avoid this by branching into $F[y]$, $F[yz]$, and $F[yz]$ instead. Here, we study in more detail what happens in the single subcases: Setting y true in the first subcase of the branching, we satisfy two y-clauses. By setting y false and z true in the second subcase, we directly eliminate two y- and two z-clauses. Consequently, the Dominating Unit Clause Rule now applies for x and satisfies two additional clauses. In total, we satisfy six clauses in the second subcase. Setting y and z false in the third subcase, we satisfy two y- and two z-clauses. In addition, there arise unit clauses for x and x such that the Complementary 1-Clause Rule and then the Resolution Rule apply, satisfying two additional clauses. Summarizing these considerations, the resulting branching vector is $(2,6,6)$ with branching number 1.3022.

For our purpose, this vector is still not good enough. However, we observe that in the first branch $\bar{x}$, is reduced to occurrence three, meaning that in this branch the next rule that will be applied will undoubtly be RULE 2. We recall that RULE 2 yields the branching vector $(2,5)$, and possibly even a better one. Combining these two steps, we obtain the branching vector $(4,7,6,6)$ and the branching number 1.2812.

Note that in RULE 5, we have the real worst case of the algorithm, namely for the situation of $\#_{\bar{x}\bar{y}} = \#_{\bar{x}\bar{z}} = \#_{\bar{y}\bar{z}} = 1$ and $\bar{y}$ and $\bar{z}$ having their common occurrence in a clause $\{y,z\}$. For this situation, we can find no branching rule improving the branching number 1.2886.

RULE 6: When this rule applies, all literals in the formula are either $(3,2)$ or $(2,3)$. We choose a $(3,2)$-literal x. The branching instruction is now primarily the same as in RULE 5 above. However, it is now possible that there is no literal z occurring in a clause $\{x,z\}$, as the two x-occurrences may be in unit clauses. In this case, i.e. for two x-unit clauses, we branch into $F[y]$ and $F[\bar{y}]$. Having two or more common occurrences for a pair of $\bar{x}$, $\bar{y}$, and $\bar{z}$, we branch as in RULE 5. For the remaining cases, i.e. $\#_{\bar{x}\bar{y}} = 1$, $\#_{\bar{x}\bar{z}} = 1$, and $\#_{\bar{y}\bar{z}} \geq 1$, we branch into $F[y]$, $F[\bar{y}z]$, and $F[\bar{y}\bar{z}]$.

The complexity analysis works analogously to RULE 5. For $\#_{\bar{x}\bar{y}} = 1$, $\#_{\bar{x}\bar{z}} = 1$, and $\#_{\bar{y}\bar{z}} \geq 1$ we test all possible arrangements of $\bar{x}$, $\bar{y}$, and $\bar{z}$ with x being $(3,2)$ and y and z being either $(3,2)$ or $(2,3)$. The worst case branching vector in these situations, when branching into $F[y]$, $F[\bar{y}z]$, and $F[\bar{y}\bar{z}]$, is $(2,9,5)$ and yields the branching number 1.2835. Again, we omit the details.

4 A Bound in the Formula Length

Compare Theorem 3 with the $O(1.1058^{|F|})$ time bound for MaxSat [1]. Observe that when the exponential bases are close to 1, even small improvements in the exponential base can mean significant progress.

Theorem 3 Max2Sat *can be solved in time* $O(1.0970^{|F|})$.

We sketch the proof of Theorem 3, presenting Algorithm B with the given running time. For the most part, it is equal to Algorithm A, sharing the branching instructions of RULEs 1 to 4. Taking up ideas given in [1], we replace RULE 5 and 6 with new branching instructions RULE $5'$, $6'$, $7'$, and $8'$.

For the rules known from Algorithm A, it remains to examine their branching vectors with respect to formula length. As the analysis is in essence the same as that of the proof for Theorem 1, we omit the details once again, while only stating that the worst case branching vector with respect to formula length for RULEs 1 to 4 is $(7; 8)$ (branching number $1:0970$), and continue with the new instructions:

RULE 5^0: Upon reaching this rule, all literals in the formula are of type $(2; 2)$, $(3; 2)$, or $(2; 3)$. RULE 5^0 deals with the case that there is a $(3; 2)$-literal x, which is not $(3; 2)[2; 2; 2][1; 2]$.

If x is a $(3; 2)[2; 2; 2][2; 2]$-literal, we branch into $F[x]$ and $F[x]$. Counting the literals eliminated in either branch, we easily obtain a branching vector of $(8; 7)$.

If x is $(3; 2)[2; 2; 2][1; 1]$-literal with clauses $fx; y_1g$, $fx; y_2g$, and $fx; y_3g$ in which some of y_1, y_2, and y_3 may be equal, we branch into $F[x]$ and $F[xy_1y_2y_3]$. This is correct, as should we want to satisfy more clauses by setting x to true than by setting x to false, all y_1, y_2 and y_3 must be falsi ed. We easily check that if all y_1, y_2, and y_3 are equal, we obtain a branching vector of $(10; 10)$. For at least two literals of y_1, y_2, and y_3 being distinct, we eliminate in the rst subcase eight literals, namely the literals in the satis ed x-clauses and the falsi ed x-literals. In the second subcase, we eliminate $\bar{x}$, having ve occurrences and two variables having at least four occurrences. This gives a branching vector of $(8; 13)$, corresponding to the branching number $1:0866$.

If x is ultimately a $(3; 2)[1; 2; 2][2; 2]$-literal with clauses $fx; z_1g$, $fx; z_2g$ in which z_1 and z_2 may be equal, we branch into $F[x]$ and $F[xz_1z_2]$. The correctness is shown as in the previous case. In the rst branch, we directly eliminate eight literals. In the second branch, we eliminate literal x having ve occurrences and at least one literal having four or ve occurrences. This gives a branching vector of $(7; 9)$, corresponding to the branching number $1:0910$.

By using these branching instructions we obtain for RULE 5^0 the worst case branching vector $(8; 7)$ in terms of formula length, namely for a $(3; 2)[2; 2; 2][2; 2]$-literal x. This corresponds to the branching number $1:0970$ and will turn out to be the overall worst case in our analysis of the algorithm.

RULE 6^0: Upon reaching this rule, all remaining literals in the formula are either $(2; 2)$, $(3; 2)[2; 2; 2][1; 2]$, or $(2; 3)[1; 2][2; 2; 2]$. RULE 6^0 deals with the case that there is a $(2; 2)[2; 2][1; 2]$-literal x, i.e. a $(2; 2)$-literal having a unit occurrence of x. As this rule is similar to RULE 5^0, we omit the details here and claim a worst case branching vector of $(5; 12)$ corresponding to the number $1:0908$.

RULE 7^0 eliminates all remaining $(3; 2)$-literals, namely those of type $(3; 2)[2; 2; 2][1; 2]$. We select literals y_1, y_2, y_3, and z from clauses $fx; y_1g$, $fx; y_2g$, $fx; y_3g$, and $fx; zg$. If there is a variable $\bar{y}$ which equals at least two of the variables $\bar{y}_1$, $\bar{y}_2$, $\bar{y}_3$, and $\bar{z}$, we branch into subcases $F[y]$ and $F[y]$. Otherwise, i.e. all variables $\bar{y}_1$, $\bar{y}_2$, $\bar{y}_3$, and $\bar{z}$ are distinct, we branch into subcases $F[y_1x]$, $F[y_1xy_2y_3z]$, and $F[y_1]$. The analysis of this rule is omitted here, as it is in large extent analogous to the nal RULE 8^0, which we will study in more detail.

RULE 8^0 applies to the $(2; 2)[2; 2][2; 2]$-literals, which are the only literals remaining in the formula. Consider clauses $fx; y_1g$, $fx; y_2g$, $fx; z_1g$, and $fx; z_2g$.

In the case where there is a variable y which equals two of the variables y_1, y_2, z_1, or z_2, i.e. y has two or more common occurrences with x, we branch into $F[y]$ and $F[y]$. We can easily see how to obtain a branching vector of $(8, 8)$ and the branching number 1.0906, as setting a value for y implies a value for x. Therefore, we proceed to the case of distinct variables y_1, y_2, z_1, and z_2.

First, we discuss the correctness of the subcases. The correctness of the subcases $F[y_1x]$, $F[y_1x]$, $F[y_1x]$ and $F[y_1x]$ is obvious. Now assume in the second branch that a partner of x, e.g. z_1, would be falsified. Then, in comparison to the first branch, we would lose the now falsified clause $\{x, z_1\}$, but could, in the best case, gain one additional x-clause. On the other hand, assume that y_2 would be satisfied. Then in the second branch, as compared with the first one, we can not gain any additional x-clause, but could lose some x-clauses. This shows that in the second branch, we can neglect the considered assignments, as they do not improve the result obtained in the first branch. Analogously, we obtain the additional assignments in the fourth branch and, therefore, branch into subcases $F[y_1x]$, $F[y_1xy_2z_1z_2]$, $F[y_1x]$, and $F[y_1xy_2z_1z_2]$. Knowing that all literals in the formula are $(2, 2)[2, 2][2, 2]$, we obtain the vector $(11, 20, 11, 20)$.

As this vector does not satisfy our purpose, we further observe that in branch $F[y_1x]$ and in branch $F[y_1x]$, there are undoubtly literals reduced to an occurrence of three or two. These literals are either eliminated due further reduction, or give rise to a RULE 2 branching in the next step. We check that the worst case branching vector in RULE 2 is $(7, 10)$. Combining these steps, we are now able to give a worst case branching vector for RULE $8'$ of $(18, 21, 20, 18, 21, 20)$, corresponding to the branching number 1.0958.

This completes our algorithm and its analysis in terms of formula length. Omitting some details, we have shown a worst case branching number of 1.0970 in all branching subcases, which justifies the claimed time bound.

For MaxSat in terms of the number of clauses, the upper time bound $O(1.3413^K |F|)$ is known [1]. Setting $|F| = 2K$ in Theorem 3, we obtain:

Corollary 4 Max2Sat *can be solved in time* $O(1.2035^K)$.

Using this algorithm we can also solve the Maximum Cut problem, as we can translate instances of the Maximum Cut problem into 2CNF formulas [11]. In fact, these formulas exhibit a special structure and we can modify and even simplify the shown algorithm, in order to obtain better bounds on formulas having this special structure. As shown in [8] on 2CNF formulas generated from Maximum Cut instances, Max2Sat can be solved in time $O(1.0718^{|F|})$ and $O(|F| + 1.2038^k)$, where k is the maximum number of satisfiable clauses in the formula. This implies the bounds for Maximum Cut shown in Theorem 5. Observe for part (2) that Maximum Cut, when restricted to graphs of vertex degree at most three, is NP-complete [7].

Theorem 5 *1. For a graph with n vertices and m edges the Maximum Cut problem is solvable in $O(1.3197^m)$ time.*

2. *If the graph has vertex degree at most three, then Maximum Cut can be solved in time* $O(1{:}5160^n)$. *If the graph has vertex degree at most four, then Maximum Cut can be solved in time in* $O(1{:}7417^n)$.
3. *We can compute in time* $O(m + n + k \; 1{:}7445^k k)$ *whether there is a maximum cut of size* k.

5 Experimental Results

Here we indicate the performance of our algorithms A (Section 3) and B (Section 4), and compare them to the two-phase heuristic algorithm for MaxSat presented by Borchers and Furman [3]. The tests were run on a Linux PC with an AMD K6 processor (233 MHz) and 32 MByte of main memory. All experiments are performed on random 2CNF-formulas generated using the MWFF package from Bart Selman [14]. We take di erent numbers of variables and clauses into consideration and, for each such pair, generate a set of 50 formulas. As results, we give the average for these sets of formulas. If at least one of the formulas in a set takes longer than 48 hours, we do not process the set and indicate this in the table by \not run". Our algorithms are implemented in JAVA. This gives credit to the growing importance of JAVA as a convenient and powerful programming language. Furthermore, our aim is to show how the algorithms limit the exponential growth in running time, being e ective independent of the programming language. The algorithm of Borchers and Furman is coded in C. Coding a simple program for Fibonacci recursion in C and JAVA and running it in the given environment, we found the C program to be faster by a factor of about nine. Due to the di erent performance of the programming languages, it is di cult to only compare running times. As a fair measure of performance we, therefore, also provide the size of the scanned branching tree, as it is responsible for the exponential growth of the running time. More precisely, for the branching tree size we count all inner nodes, where we branch towards at least two subcases.

There is almost no di erence in the performance between algorithms A and B; therefore they are not listed separately. This is plausible, as in the processing of random formulas, the \bad" case situations in whose handling our algorithms di er, are rare. On problems of small size, the 2-phase-EPDL (Extended Davis-Putnam-Loveland) algorithm of Borchers and Furman [3] has smaller running times despite its larger branching trees. One reason may also be the di erence in performance of JAVA and C. Nevertheless, with a growing problem size our algorithm does a better job in keeping the exponential growth of the branching tree small, which also results in signi cantly better running times, see Table 1.

In order to gain insight into the performance of our rules, we collected some statistics on the application of the transformation and branching rules. For algorithm B, we examine which rules apply how often during a run on random formulas. First, we consider the transformation rules. Note that at one point, several transformation rules could be applicable to a formula. Therefore, for judging the results, it is important to know the sequence in which the application of transformation rules is tested. We show the results in Table 2, with the rules

		Algorithm B		2-Phase-EDPL	
n	m	Tree	Time	Tree	Time
25	100	16	0.77	961	0.27
	200	108	1.78	37 092	1.93
	400	385	5.41	514 231	43.72
	800	752	12.59	2 498 559	9:16.51
50	100	6	0.70	69	0.48
	200	320	4.48	611 258	27.11
	400	18 411	3:45.80	not run	{
100	200	36	1.14	10 872	2.14
	400	91 039	23:50.09	not run	{
200	400	1 269	21.87	not run	{

Table 1. Comparison of average branching tree sizes (Tree) and average running times (Time), given in minutes:seconds, of our Algorithm B and the 2-phase-EDPL by Borchers and Furman. Tests are performed on 2CNF formulas with di erent numbers of variables (n) and clauses (m).

Variables	25				50		
Clauses	100	200	400	800	100	200	400
Search Tree Size	16	108	385	752	6	320	18 411
Almost Common Cl.	10	38	102	235	4	76	1 421
Pure Literals	26	82	111	99	34	658	11 476
Dominating 1-Clause	123	704	1 844	2 839	42	3387	134 425
Complementary 1-Clause	40	571	2 831	6 775	4	1173	128 030
Resolution	22	57	54	30	25	726	10 821
Three Occurrences 1	0	1	4	7	0	1	35
Three Occurrences 2	6	25	51	47	2	86	2 810

Table 2. Statistics about the application of transformation rules in algorithm B on random formulas.

being in the order in which they are applied. Considering the shown and additional data, we nd application pro les being characteristic for variable/clause ratios. We observe for formulas having a higher ratio, i.e. with fewer clauses for a xed number of variables, that the Dominating 1-Clause Rule is the rule which is applied most often. With lower ratio, i.e. when we have more clauses for the same number of variables, the Complementary 1-Clause Rule gains in importance.

Besides the transformation rules, we also study the frequency in which the single branching rules are applied. Recall that algorithm B has a list of eight di erent cases with corresponding branching rules. We show the results collected during runs on random formulas in Table 3. We observe that the most branching steps occur with RULE 1 or RULE 2. The other rules are used in less than one percent of the branchings. It is reasonable that in formulas with

Variables	25				50		
Clauses	100	200	400	800	100	200	400
Tree Size	15.26	107.4	384.4	751.18	5.26	319.36	18 410.38
RULE 1	10.62	102.32	381.76	749.42	0.88	217.36	18 300.02
RULE 2	4.02	3.54	1.18	0.32	4.34	100.12	97.54
RULE 3	0.5	0.9	0.94	0.64	0	1.44	11.14
RULE 4	0.08	0.44	0.26	0.36	0.04	0.34	1.22
RULE 5′	0.04	0.12	0.16	0.26	0	0.08	0.3
RULE 6′	0	0.06	0.1	0.16	0	0.02	0.16
RULE 7′	0	0.02	0	0	0	0	0
RULE 8′	0	0	0	0.02	0	0	0

Table 3. Statistics on the application of branching rules in algorithm B on random formulas having n variables and m clauses. Recall that each result is the average on 50 formulas to understand that we give non-integer values. Thereby we even see the application of very rare rules.

a high variable/clause ratio, i.e. fewer clauses, we have more variables with an occurrence of three. Therefore, the rule applied most while processing these formulas is RULE 2. As the variable/clause ratio shifts down, i.e. when we have more clauses for the same number of variables, there necessarily are more variables with a large number of occurrence in the formula. Consequently, RULE 1 becomes dominating.

Considering our statistics, we can roughly conclude: Some of the transformation rules are, in great part, responsible for the good practical performance of our algorithms, as they help to decrease the search tree size. The less frequent transformation rules and the rather complex set of branching rules, on the other hand, are mainly important for guaranteeing good theoretical upper bounds.

6 Open Questions

There remains the option of investigating exact algorithms for other versions of Max2Sat, for example, Max3Sat. Furthermore, n being the number of variables, can Max2Sat be solved in less than 2^n steps? Regarding Hirsch's recent theoretical results [10], it seems a promising idea to combine our algorithm with his, in order to improve the upper bounds for Max2Sat even further.

References

1. N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science, Chennai, India, Dec. 1999. Springer-Verlag.
2. R. Battiti and M. Protasi. Approximate algorithms and heuristics for MAX-SAT. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 77{148. Kluwer Academic Publishers, 1998.

3. B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299{306, 1999.
4. J. Cheriyan, W. H. Cunningham, L. Tuncel, and Y. Wang. A linear programming and rounding approach to Max 2-Sat. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:395{414, 1996.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
6. U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *3d IEEE Israel Symposium on the Theory of Computing and Systems*, pages 182{189, 1995.
7. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.
8. J. Gramm. Exact algorithms for Max2Sat: and their applications. Diplomarbeit, Universität Tübingen, 1999. Available through http://www-fs.informatik.uni-tuebingen.de/∼niedermr/publications/index.html.
9. J. Hastad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1{10, 1997.
10. E. A. Hirsch. A new algorithm for MAX-2-SAT. Technical Report TR99-036, ECCC Trier, 1999. To appear at *STACS 2000*.
11. M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335{354, 1999.
12. R. Niedermeier and P. Rossmanith. New upper bounds for MaxSat. In *Proceedings of the 26th International Conference on Automata, Languages, and Programming*, number 1644 in Lecture Notes in Computer Science, pages 575{584. Springer-Verlag, July 1999. Long version to appear in *Journal of Algorithms*.
13. V. Raman, B. Ravikumar, and S. S. Rao. A simpli ed NP-complete MAXSAT problem. *Information Processing Letters*, 65:1{6, 1998.
14. B. Selman. MWFF: Program for generating random Max*k*-Sat instances. Available from DIMACS, 1992.
15. M. Yannakakis. On the approximation of maximum satis ability. *Journal of Algorithms*, 17:475{502, 1994.

Dynamically Maintaining the Widest k-Dense Corridor

Subhas C. Nandy[1?], Tomohiro Harayama[2], and Tetsuo Asano[2]

[1] Indian Statistical Institute, Calcutta - 700 035, India
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan

Abstract. In this paper, we propose an improved algorithm for dynamically maintaining the *widest k-dense corridor* as proposed in [6]. Our algorithm maintains a data structure of size $O(n^2)$, where n is the number of points present on the floor at the current instant of time. For each insertion/deletion of points, the data structure can be updated in $O(n)$ time, and the widest k-dense corridor in the updated environment can be reported in $O(kn + n\log n)$ time.

1 Introduction

Given a set S of n points in the Euclidean plane a corridor C is de ned as an open region bounded by parallel straight lines ‘^{0} and ‘^{00} such that it intersects the convex hull of S [3]. The width of the corridor C is the perpendicular distance between the bounding lines ‘^{0} and ‘^{00}. The corridor is said to be k-dense if C contains k points in its interior. The widest k-dense corridor through S is a k-dense corridor of maximum width [1]. See Figure 1 for illustration.

The widest empty corridor problem was rst proposed by [3] in the context of robot motion planning where the objective was to nd an widest straight route avoiding obstacles. They also proposed an algorithm for this problem with time and space complexities $O(n^2)$ and $O(n)$ respectively. The widest k-dense corridor problem was introduced in [1] along with an algorithm of time and space complexities $O(n^2\log n)$ and (n^2) respectively. Here the underlying assumption is that the robot can pass through (or in other words, can tolerate collission with) a speci ed number (k) of obstacles. In [4], the space complexity of the widest k-dense corridor problem was improved to $O(n)$. In the same paper, they have suggested an $O(n\log n)$ time and $O(n^2)$ space algorithm for maintaining the widest empty corridor where the set of obstacles is dynamically changing. However, the dynamic problem for general k (0), was posed as an open problem. In [6], both the static and dynamic versions for the k-dense corridor problem are studied. The time and space complexities of their algorithm for the static version

[?] This work was done when the author was visiting School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan.

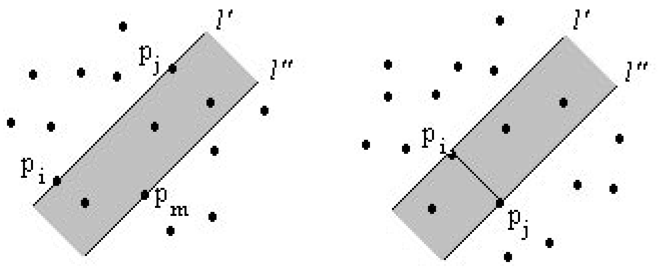

Fig. 1. Two types of corridors

of the problem are $O(n^2)$ and $O(n^2)$ respectively. For the dynamic version, their algorithm is the pioneering work. Maintaining an $O(n^2)$ size data structure they proposed an algorithm which reports the widest k-dense corridor after the insertion and deletion of a point. The time complexity of their algorithm is $O(K\log n)$, where $O(K)$ is the combinatorial complexity of (k)-*level* of an arrangement of n half-lines, each of them belongs to and touching the same side of a given line. They proved that the value of K is $O(kn)$ in the worse case.

In this paper, we improve the time complexity of the dynamic version of the widest k-dense corridor problem. Given $O(n^2)$ space for maintaining the data structure, our algorithm can update the data structure and can report the widest k-dense corridor in $O(K + n\log n)$ time. As it is an online algorithm, this reduction in the time complexity is de nitely important.

2 Geometric Preliminaries

Throughout the paper, we assume that the points in S are in general position, i.e., no three points in S are collinear, and the lines passing through each pair of points have distinct slope. Theorem 1, stated below, characterizes a widest corridor among the points S.

Theorem 1. *[1,3,4,6] Let C be the widest corridor with bounding lines $‘^{0}$ and $‘^{00}$. Then C must satisfy the following conditions :*

(A) $‘^{0}$ touches two distinct points p_i and p_j of S and $‘^{00}$ touches a single point p_m of S, or
(B) $‘^{0}$ and $‘^{00}$ contain points p_i and p_j respectively, such that $‘^{0}$ and $‘^{00}$ are perpendicular to the line through p_i and p_j.

From now onwards, a k-dense corridor satisfying conditions (A) and (B) will be referred to as *type-A* and *type-B* corridors respectively (see Figure 1).

2.1 Relevant Properties of Geometric Duality

We follow the same tradition [1,3,4,6] of using geometric duality for solving this problem. It maps (i) a point $p = (a, b)$ to the line $D(p) : y = ax - b$ in the dual plane, and (ii) a non-vertical line $\ell : y = mx - c$ to the point $D(\ell) = (m, c)$ in the dual plane. Needless to say, a point p is below (resp., on, above) a line ℓ in the primal plane if and only if $D(p)$ is above (resp., on, below) $D(\ell)$ in the dual plane. A line passing through two points p and q in the primal plane, corresponds to the point of intersection of the lines $D(p)$ and $D(q)$ in the dual plane, and vice versa.

For the k-dense vertical corridors, we can not apply geometric duality theory, and so we apply the vertical line sweep technique. We maintain a balanced binary leaf search tree, say $BB(\ )$ tree, with the existing set of points in the primal plane. Here each point in S appears at the leaf level, and is attached with the width of the widest k-dense vertical corridor with its left boundary passing through that point. At each non-leaf node, we attach the width of the widest vertical k-dense corridor in the subtree rooted at that node. It can be easily shown that for each insertion/deletion of a point, the necessary updates in this data structure and the reporting of widest k-dense vertical corridor can be done in $O(k + \log n)$ time. Below, we concentrate on studying the properties of the non-vertical corridors.

Consider the two bounding lines ℓ' and ℓ'' of a corridor C in the primal plane, which are mutually parallel. The corresponding two points, $D(\ell')$ and $D(\ell'')$ in the dual plane, will have the same x-coordinate. Thus a corridor C will be represented by a vertical line segment joining $D(\ell')$ and $D(\ell'')$ in the dual plane, and will be denoted by $D(C)$. The width of C is $\frac{|y(D(\ell'))-y(D(\ell''))|}{\sqrt{1+(x(D(\ell')))^2}}$, and will be referred as the *dual distance* between the points $D(\ell')$ and $D(\ell'')$. Here $x(p)$ and $y(p)$ denote the x- and y-coordinates of the point p respectively.

Let $H = \{h_i = D(p_i) \mid p_i \in S\}$ be the set of lines in the dual plane corresponding to the n points of S in the primal plane. Let p be a point inside the corridor C. In the dual plane, the points $D(\ell')$ and $D(\ell'')$ will lie in the opposite sides of the line $D(p)$. Now, we have the following observation, which is the direct implication of Theorem 1. The dual of a k-dense corridor is characterized in Observation 2.

Observation 1 *Let C be a corridor bounded by a pair of parallel lines ℓ', ℓ''.*

- *Now, if C is a type-A corridor, ℓ' passes through p_i and p_j, and ℓ'' passes through p_m. This implies that $D(\ell')$ corresponds to a vertex of $A(H)$, which is the point of intersection of h_i and h_j (denoted by $h_i \cap h_j$), and $D(\ell'')$ corresponds to a point on h_m satisfying $x(D(\ell')) = x(h_i \cap h_j)$.*
- *If C is a type-B corridor, ℓ' and ℓ'' pass through the two points p_i and p_j respectively. This implies, $D(\ell')$ and $D(\ell'')$ will correspond to the two points on h_i and h_j respectively, satisfying $x(D(\ell')) = x(D(\ell'')) = -(1/x(h_i \cap h_j))$.*

Thus, A non-vertical *type-A* corridor may uniquely correspond to a vertex of $A(H)$, and a non-vertical *type-B* corridor may also uniquely correspond to an edge of $A(H)$, on which its upper end point lies.

Observation 2 *A corridor C is said to be k-dense if and only if there are exactly k lines of H that intersect the vertical line segment $D(C)$, representing the dual of the corridor C, and will be commonly referred to as a k-stick.*

Thus, recognizing a widest k-dense non-vertical corridor in the primal plane is equivalent to nding a *k-stick* in the dual plane having maximum *dual length*.

3 Widest Non-vertical k-Dense Corridor

We now explain an appropriate scheme for maintaining the widest non-vertical k-dense corridor dynamically. Let $A(H)$ denote the arrangement of the set of lines H [2]. The number of vertices, edges and faces in $A(H)$ are all $O(n^2)$. In the dynamic scenario, we need to suggest an appropriate data structure which can be updated for insertion/deletion of points, and the widest k-dense corridor can be reported e ciently in the changed scenario. As the deletion is symmetric to insertion, we shall explain our method for insertion of a new point in S only.

3.1 Data Structures

We dynamically maintain the following data structure which stores the arrangement of the lines in H. It is de ned using the concept of *levels* as stated below.

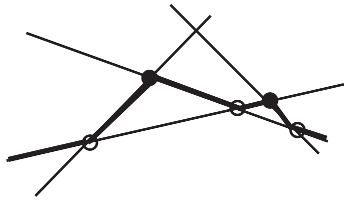

Fig. 2. Demonstration of levels in an arrangement of lines

De nition 1. [2] A point   in the dual plane is at level   (0 n) if there are exactly   lines in H that lie strictly below . The -level of $A(H)$ is the closure of a set of points on the lines of H whose levels are exactly   in $A(H)$, and is denoted as L (H). See Figure 2 for an illustration.

Clearly, L (H) is a polychain from $x = -1$ to $x = 1$, and is monotone increasing with respect to x-axis. The vertices on L (H) is precisely the union of vertices of $A(H)$ at levels -1 and . The edges of L (H) are the edges of $A(H)$ at level . In Figure 2, a demonstration of levels in the arrangement $A(H)$ is shown. Here the thick chain represents $L_1(H)$. Among the vertices of $L_1(H)$, those marked with empty (black) circles are appearing in level 0 (2) also. Each vertex of the arrangement $A(H)$ appears in two consecutive levels, and each edge of $A(H)$ appears in exactly one level. We shall store L (H), 0 n in a data structure as described below.

level-**structure**

It is an array of size n, called **primary structure**, whose -th element is composed of the following elds :

level-id : an integer containing the level-id .
left-prt : pointing to the left most node of the **secondary structure** T .
root-ptr : pointing the root node of the **secondary structure** T .
list-ptr : pointing to a linear link list, called **TEMP-list**, whose each element is a tuple $(';r)$ of pointers. The **TEMP-list** data structure will be explained after de ning the secondary structure.

The **secondary structure** at a particular level , denoted as T , is organized as a height balanced binary tree (AVL-tree). The nodes of this tree correspond to the vertices and edges at level in left to right order. In addition, each node is attached with the following information.

Two integer elds, called *LEN* and *MAX*, are attached with each node. The *LEN*- eld contains the *dual length* of the k-*stick* attached to it. Here we explicitly mention that, if a node corresponds to a vertex of the arrangement, it de nes at most one k-*stick*, but if it corresponds to an edge, more than one k-*sticks* may be de ned by that edge. In that case, the *LEN*- eld will contain the length of the one having maximum length among them. A node (corresponding to an edge) de ning no k-*stick* will contain a value 0 in its *LEN*- eld. The *MAX*- eld contains the maximum value of the *LEN* elds among all the nodes in the subtree rooted at that node. This actually indicates the widest one among all the k-dense corridors stored in the subtree rooted at that node.

Apart from its two child pointers, each node of the tree has three more pointers.
parent pointer : It helps in traversing the tree from a node towards its root. The parent pointer of the root node points to the corresponding element of the primary structure.
neighbor-pointer : It helps the constant time access of the in-order successor of a node.
self-indicator : As an element representing a vertex appears in the secondary structure (T) of two consecutive levels, each of them is connected with the other using this pointer.

By Observation 1 and succeeding discussions, a *type-A* k-dense corridor corresponds to a vertex of the arrangement. A vertex v 2 $A(H)$ appearing in levels, say and + 1, may correspond to at most two k-sticks (corresponding to two di erent *type-A* k-dense corridors), whose one end point is positioned at vertex v, and their other end points lie on some edge at levels $-k-1$ (if $-k-1>0$) and $+k+2$ (if $+k+2<n$) respectively, and are attached to the vertex v appearing in the corresponding levels. An edge e appearing at level stores at most one *k-stick* which is de ned by it and another edge in ($-k-1$)-th level and appears vertically below it.

TEMP-list : After the addition of a new point p in S, its dual line $h = D_S(p)$ is inserted in $A(H)$ to get an updated arrangement $A(H^0)$, where $H^0 = H$ h. This may cause rede ning the *k-sticks* of some vertices and edges of $A(H^0)$.

In order to store this information, we use a linear link list at each level of the primary structure. Each element of this list is a tuple ($`;r$). Here ‘ and $r$ points to two elements (vertex/edge) at level , and the tuple ($`;r$) represents a set of consecutive elements (vertices/edges) in T such that the *k-sticks* de ned by all the vertices and edges in that set has been rede ned due to the appearance of the new line h in the dual plane. Note that,

The list attached to a particular level, say , of the arrangement may contain more than one tuple after computing the *k-sticks* at all the vertices and edges of $A(H)$ a ected by the inclusion of h. In that case, the set of elements represented by two distinct tuples, say ($`_1;r_1$) and ($`_2;r_2$) in that list must be disjoint, i.e., $r_1 < `_2$. Moreover, the elements represented by $r_1$ and $`_2$ must not be consecutive in T .

L-list : We shall use another temporary linear link list (L) during the processing of an insertion/deletion of a line h in the arrangement $A(H)$. This contains the intersection points of h with the lines in H in a left to right order.

3.2 Updating the Primary Structure

We rst compute the leftmost intersection point of the newly inserted line h with the existing lines in H by comparing all the lines. Let the intersection point be and the corresponding line be h_i. In order to nd the edge e 2 $A(H)$ on which lies, we traverse along the line h_i from its left unbounded edge towards right using the *neighbor-pointers* and *self-indicators*.

Next, we use *parent pointers* from the edge e upto the root of T and nally, the parent pointer of the root points to the primary structure record corresponding to the level . The level of the left unbounded edge e on the newly inserted line h in the updated structure $A(H^0)$ will be (= or (+ 1)) depending on whether h intersects e from below or from above.

We replace the old primary structure by a new array of size $n+1$, and insert a new level corresponding to the left unbounded edge e in appropriate place.

Moreover, the *list_ptr* for all the levels are initialized to NULL. It is easy to see that the updating of the primary structure requires $O(n)$ time.

Our updating of the secondary structure will be guided by two pointers, $P1$ and $P2$, which will initially contain the edges at the T_{-1} and T_{+1} which are just below and above respectively.

3.3 Updating the Secondary Structure

Let the level of e (the unbounded portion of h to the left of) be in $A(H^0)$, and the edge e $(2\ A(H))$, which is intersected by h, be at level (= $-$ 1 or + 1). In this subsection, we describe the creation of the new edges and vertices generated due to the inclusion of h in $A(H)$.

The portions of h to the left and right of are denoted as e and e^0 respectively, and the portions of e to the left and right of the point by e_{left} and e_{right} respectively. Note that, the vertex appears in both the levels and . Next, we do the following changes in the secondary structure for the inclusion of the new vertex and its adjacent edges in the *label*-structure. Refer to Figure 3.

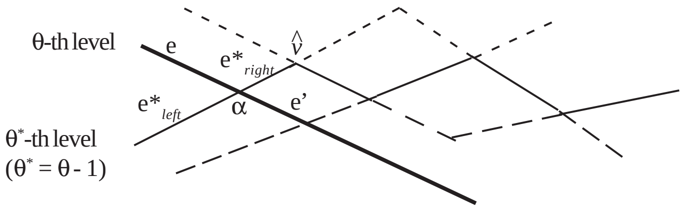

Fig. 3. Processing a new vertex of $A(H^0)$

e and the vertex are added in T .
e_{left} remains in its previous level , so e is replaced by e_{left} in T .
e_{right} goes to level . So, rst of all e_{right} is added to T .
Let $\hat{v}$ be the vertex at the right end of e (recently modi ed to e_{left}) in T . The tree T is split into two height balanced trees, say T_R and T_L, where T_R contains all the elements (vertices and edges) in that level to the right of $\hat{v}$ including itself, and T_L contains all the elements in the same level to the left of e_{left} and including itself. This requires $O(\log n)$ time [5].
Next we concatenate T_R to the right of e_{right} in T . The *neighbor-pointer* of e_{right} is immediately set to point $\hat{v}$, and the *parent-pointers* of the a ected nodes are appropriately adjusted. This can be done in $O(\log n)$ time [5].
Finally, the vertex is added in T_L as the right most element, and T_L is renamed as T . Note that a portion of e^0 will be the right neighbor of the

vertex in T . Now, if we have already considered all the n newly generated vertices, the right end of e^0 will be unbounded. In that case, e^0 is added in T as the rightmost element, and its *neighbor pointer* and *parent pointer* are appropriately set. Otherwise, the right end of e is yet to be de ned, and its addition in T is deferred until the detection of the next intersection.

In the former case, the updating of the secondary structure is complete. But in the later case, we proceed with e^0, the portion of h to the right of . First of all, we set e to e^0. Now, (i) if $=\ -1$, then $P2$ is set to e_{right} and $P1$ needs to be set to an appropriate edge in level -2, and (ii) if $=\ +1$, then $P1$ is set to e_{right} and $P2$ needs to be set to an appropriate edge in level $+2$. Finally, the current level is set to , and proceed to detect next intersection.

Now two important things need to be mentioned.

For all newly created edges/vertices, we set the width of the k-dense corridor to 0. They will be computed afresh after the update of the *level*-structure.
During this traversal, we create L with all the newly created edges and vertices on h in a left-to-right order. The edges are attached with their corresponding levels. As the newly created vertices appear in two consecutive levels, they show their lower levels in the L list.

Lemma 1. *The time required for constructing $A(H^0)$ from the existing $A(H)$ is $O(n \log n)$ in the worst case.* ⊓⊔

3.4 Computing the New k-Dense Corridors

We now describe the method of computing all the *k-sticks* which intersect the newly inserted line h. The L list contains the pieces of the line h separated by the vertices in $(A(H^0) - A(H))$, which will guide our search process. We process edges in the L list one by one from left to right. For each edge $e\ 2\ L$, we locate all the vertices and edges of $A(H^0)$ whose corresponding *k-sticks* intersect e.

We proceed with an array of pointers P of size $2k + 3$, indexed by $-(k + 1) ::: 0 ::: (k + 1)$. Initially, $P(0)$ points to the leftmost edge in the L list. If its level in the *level*-structure is then $P(-1) ::: P(-k - 1)$ will point to the leftmost edges at levels $(\ -k-1) ::: (\ -1)$, and $P(1) ::: P(k + 1)$ will point to the leftmost edges at levels $(\ +1) ::: (\ +k+1)$ in the *level*-structure.

While processing an edge $e\ 2\ L$ (not the left-most edge), $P(0)$ points to the edge e; $P(-1) ::: P(-k - 1)$ point to $k + 1$ edges below the left end vertex of e and $P(1) ::: P(k + 1)$ point to $k + 1$ edges above the left end vertex of e. At the end of the execution of edge e, if e is not right unbounded, we set $P(0)$ to the next edge of e in L and proceed. Otherwise, our search stops.

In order to evaluate all the *k-sticks* intersecting e and having its bottom end at level i ($i =\ -k-1; :::;\ $), we need to consider the pair of levels $(i; i + k + 1)$.

Consider a x-monotone polygon bounded below (resp. above) by the x-monotone chain of edges and vertices at level i (resp. $i + k + 1$) and by two vertical lines at the end points of e (see Figure 4). This can easily be detected using the pointers $P(i -\quad)$ and $P(i + k + 1 -\quad)$. The *LEN*- elds of all the edges and vertices in the above two x-monotone chains are initialized to zero. We draw vertical lines at each vertex of the upper and lower chains, which split the polygon into a number of vertical trapezoids.

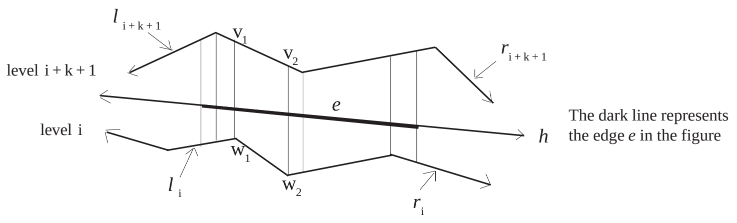

Fig. 4. Computation of *type-A* and *type-B* corridors while processing an edge $e\ 2\ L$

Each of the vertical lines drawn from the convex vertices de nes a *type-A k-stick*. Its *dual distance* is put in the corresponding node of T_i or T_{i+k+1}.

In order to compute the *type-B k-sticks*, consider each of the vertical trapezoids from left to right. Let $\quad = v_1v_2w_2w_1$ be such a trapezoid whose v_1w_1 and v_2w_2 are two vertical sides, I denote the x-range of $\quad$. Let v_1v_2 be a portion of an edge $e\quad$, which in turn lies on a line $h\quad 2\ H^0$, and w_1w_2 lies on $h\quad 2\ H^0$. We compute $-1{=}x(h \top h\quad)$ and check whether it lies in I. If so, the vertical line at $x = -1{=}x(h \top h\quad)$, bounded by v_1v_2 and w_1w_2, indicates a *type-B k-stick* corresponding to the edge $e\quad$. We compute its *dual distance*; this newly computed *k-stick* replaces the current one attached with $e\quad$ provided the dual length of the newly computed *k-stick* is greater than the *LEN*- eld attached with $e\quad$ in the data structure T_i.

Let $‘_i$ and r_i (resp. $‘_{i+k+1}$ and r_{i+k+1}) denote the edges at level i (resp. $i+k+1$), which are intersected by the vertical lines at the left and right end points of the edge $e(2\ L)$ respectively. Note that, the de nition of *k-sticks* for the edges and vertices of the i-th level between $‘_i$ and r_i may have changed due to the presence of $e\ 2\ A(H^0)$. So, we need to store the tuple $(‘_i; r_i)$ in the *TEMP*-list attached to level i of the primary structure. But, before storing it, we need to check the last element stored in that list, say $(‘\quad; r\quad)$. If the neighbor pointer of $r\quad$ points to $‘_i$ in T_i (the secondary structure at level i), then $(‘\quad; r_i)$ is a continuous set of elements in level i which are a ected due to the insertion of h. So, the element $(‘\quad; r\quad)$ is updated to $(‘\quad; r_i)$; otherwise, $(‘\quad; r\quad)$ is added in the *TEMP*-list. We store $(‘_{i+k+1}; r_{i+k+1})$ in the *TEMP*-list attached to level $i + k + 1$ of the primary structure in a similar way.

Next, we may proceed by setting $P(0)$ to the next edge e^0 of L list. Each of the pointers $P(i); i = -k-1; \ldots; k+1$, excepting $P(0)$, need to point to an edge either at level $(i + \;\; - 1)$ or at level $(i + \;\; + 1)$ which lies just below or above the current edge pointed by $P(i)$ in the *level*-structure, depending on whether e^0 lies at level $\;\; - 1$ or $\;\; + 1$ in the level structure. From the adjacencies of vertices and edges in the *level*-structure, this can be done in constant time for each $P(i)$.

Theorem 2. *The time required for computing the k-sticks intersecting the line h in $A(H^0)$ is $O(nk)$.*

Proof : Follows from the above discussions, and the fact that the complexity of the k-levels of n half-lines lying above (below) the newly inserted line h in the arrangement $A(H^0)$ is $O(nk)$. [6] □

3.5 Location of Widest k-Dense Corridor

The *TEMP*-list for a level, say , created in the earlier subsection, is used to update the *MAX*- eld of the nodes of the tree T , by considering its elements in a sequential manner from left to right. Let the tuple $(\ell; r)$ be an entry of the *TEMP-list* at the level . Let q be the common predecessor of the set of nodes represented by the tuple $(\ell; r)$. Let P_{IN} be a path from the root of T to the node q, and P_L and P_R be two paths from q to ℓ and q to r respectively. In T , the *MAX*- elds of all the nodes in the interval $(\ell; r)$, and the set of nodes in P_{IN}, P_L and P_R may be changed. So, they need to be inspected in order to update the *MAX*- elds of the nodes in T . Now we have the following lemma.

Lemma 2. *For each entry $(\ell; r)$ of the TEMP-list of a level, say , the number of nodes of T which need to be visited to update the MAX- elds is* $O(\log n + \;)$, *where is the number of consecutive vertices and edges of the arrangement at level represented by $(\ell; r)$.* □

In order to count the total number of elements attached to the *TEMP-list* at all the levels let us consider a $n \;\; n$ square grid M whose rows represent the n levels of the arrangement and its each column represents an edge on h in left to right order (See Figure 5). Consider the shaded portion of the grid; observe that its i-th column spans from row $\;\; - k - 1$ to $\;\; + k + 1$, where is the level of e_i in $A(H^0)$. This corresponds to the levels which are a ected by e_i. The shaded region is bounded by two x-monotone chains. Now, let us de ne a *horizontal strip* as a set of consecutive cells on a row which belong to the shaded portion of the grid. A horizontal strip which spans from the rst to the last column of the grid, is referred to as *long* strip. The strips which are not *long*, are called *short* strips. Note that, each strip attached to a row represents an element of the *TEMP-list* attached to the corresponding level. It is easy to observe that the number of such *short strips* is $O(n)$, and the number of such *long strips* may be at most $2k - 3$ in the worst case. Now we have the following lemma.

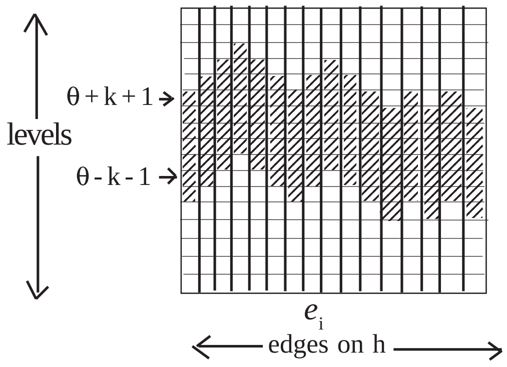

Fig. 5. Grid M estimating the number of elements in the *TEMP* list

Lemma 3. *In order to update the MAX- eld of the nodes of the secondary structure, the tree traversal time for an entry of any of the TEMP-lists is* $O(\ +\log n)$ *if the corresponding strip is short, and is* $O(\ )$ *if the strip is long, where , the length of the strip, is the number of nodes represented by the corresponding entry of the TEMP-list.*

Proof : For the *short* strips, the result follows from Lemma 2. For the *long* strips, all the nodes of the corresponding tree is a ected. So, subsumes $O(\log n)$. ⊓⊔

Finally, the roots of the trees at all levels need to be inspected to determine the widest k-dense corridor.

3.6 Complexity

Theorem 3. *Addition of a new point requires*

(a) $O(n \log n)$ *time for updating the data structure.*
(b) $O(nk)$ *time to compute the k-dense corridors containing that point.*
(c) $O(nk + n \log n)$ *time to traverse the trees attached to di erent levels of the secondary structure for reporting the widest k-dense corridor.* ⊓⊔

As we are preserving all the vertices and edges of the arrangement of the dual lines in our proposed *level*-structure, the space complexity is $O(n^2)$, where n is the number of points on the floor at the current instant of time.

References

1. S. Chattopadhyay and P. P. Das, *The k-dense corridor problems*, Pattern Recognition Letters, vol. 11, pp. 463-469, 1990.

2. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer, Berlin, 1987.
3. M. Houle and A. Maciel, *Finding the widest empty corridor through a set of points*, Report SOCS-88.11, McGill University, Montreal, Quebec, 1988.
4. R. Janardan and F. P. Preparata, *Widest-corridor problems*, Nordic J. Comput., vol. 1, pp. 231-245, 1994.
5. E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms : Theory and Practice*, Prentice-Hall, N.J., 1977.
6. C. -S. Shin, S. Y. Shin and K. -Y. Chwa, *The widest k-dense corridor problems*, Information Processing Letters, vol. 68, pp. 25-31, 1998.

Reconstruction of Discrete Sets from Three or More X-Rays

Elena Barcucci[1], Sara Brunetti[1], Alberto Del Lungo[2], and Maurice Nivat[3]

[1] Dipartimento di Sistemi e Informatica, Universita di Firenze,
Via Lombroso 6/17, 50134, Firenze, Italy,
[barcucci,brunetti]@dsi.unifi.it
[2] Dipartimento di Matematica, Universita di Siena,
Via del Capitano 15, 53100, Siena, Italy,
dellungo@unisi.it
[3] LIAFA, Institut Blaise Pascal, Universite "Denis Diderot",
2 Place Jussieu 75251, Paris Cedex 05, France,
Nivat@liafa.jussieu.fr

Abstract. The problem of reconstructing a discrete set from its X-rays in a nite number of prescribed directions is NP-complete when the number of prescribed directions is greater than two. In this paper, we consider an interesting subclass of discrete sets having some connectivity and convexity properties and we provide a polynomial-time algorithm for reconstructing a discrete set of this class from its X-rays in directions $(1, 0), (0, 1)$ and $(1, 1)$. This algorithm can be easily extended to contexts having more than three X-rays.

keywords: algorithms, combinatorial problems, discrete tomography, discrete sets, X-rays.

1 Introduction

A *discrete set* is a nite subset of the integer lattice $\mathbb{Z}^2$ and can be represented by a binary matrix or a set of unitary squares. A *direction* is a vector of the Euclidean plane. If u is a direction, we denote the line through the origin parallel to u by l_u. Let F be a discrete set; the *X-ray of F in direction u* is the function X_uF, de ned as: $X_uF(x) = jF \setminus (x + l_u)j$ for $x \; 2 \; u^?$, where $u^?$ is the orthogonal complement of u (see Fig. 1). The function X_uF is the projection of F on $u^?$ counted with multiplicity. The inverse problem of reconstructing a discrete set from its X-rays is of fundamental importance in elds such as: image processing [17], statistical data security [14], biplane angiography [16], graph theory [1] and reconstructing crystalline structures from X-rays taken by an electron microscope [15]. An overview on the problems in discrete tomography and a study of the complexity can be found in [12] and [7]. Many authors have studied the problem of determining a discrete set from its X-rays in both horizontal and vertical directions. Some polynomial algorithms that reconstruct some special sets having some convexity and/or connectivity properties, such as horizontally and vertically convex polyominoes [3,4], have been determined.

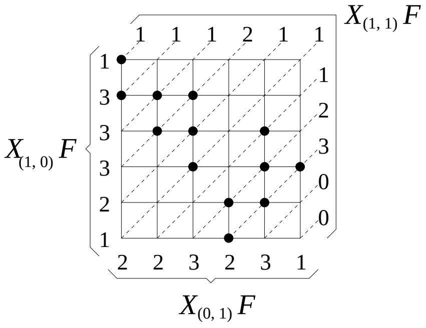

Fig. 1. X-rays in the directions $u_1 = (1; 0); \; u_2 = (0; 1)$ and $u_3 = (1; 1)$.

In this paper, we study the reconstruction problem with respect to three directions: $(1; 0); \; (0; 1)$ and $(1; 1)$. We denote the X-rays in these directions by $H; V$ and D, respectively. The basic question is to determining if, given $H \; 2 \; \mathbb{N}^m; V \; 2 \; \mathbb{N}^n$ and $D \; 2 \; \mathbb{N}^{n+m-1}$, a discrete set F whose X-rays are $(H; V; D)$ exists. Let $fu_1; : : : ; u_k g$ be a nite set of prescribed directions. The general problem can be formulated as follows:
Consistency$(u_1; : : : ; u_k)$
Instance: k vectors $X_1; : : : ; X_k$:
Question: is there F such that $X_{u_i}F = X_i$ for $i = 1; : : : ; k$?
Gardner, Gritzmann and Prangenberg [8] proved that Consistency$((1; 0); (0; 1); (1; 1))$ is NP-complete in the strong sense. Then, by means of this result and two lemmas, they proved that the problem Consistency$(u_1; : : : ; u_k)$ is NP-complete in the strong sense, for $k \quad 3$.

In this paper, we determine a class of discrete sets for which the problem is solvable in polynomial time. These sets are hex-connected and convex in the horizontal, vertical and diagonal directions. They can be represented by a set of hexagonal cells. By exploiting the new geometric properties of these structures, we can provide an algorithm which nds a solution in $O(n^5)$, where we assume $n = m$. The algorithm can be easily extended to contexts having more than three X-rays and can reconstruct some discrete sets that are convex in the directions of the X-rays. We wish to point out that the question of determining when planar convex bodies can be reconstructed from their X-rays was raised by Hammer [6,11] in 1963. The discrete analogue of this question raised by Gritzmann [10] in 1997 is an open problem. We believe that our algorithm can be considered to be an initial approach to this problem in so far as it reconstructs a discrete set which is convex in the directions of the X-rays.

2 De nitions and Preliminaries

We now wish to examine an interesting class of discrete sets for which the problem can be solved in polynomial time. We introduce some de nitions which allow us to characterize this class. Let us take the triangular lattice made up of directions: $(1;0)$; $(0;1)$ and $(1;1)$ into consideration. A point $Q = (i;j)$ of this lattice has 6 neighbours and can be represented by a hexagonal cell (see Fig. 2).

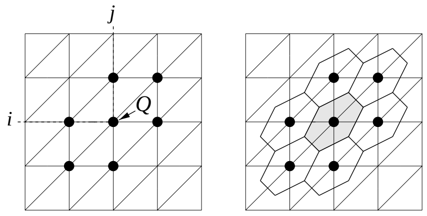

Fig. 2. The 6-neighbours of $Q = (i;j)$.

De nition 1. *If F is a discrete set, a* 6-path from P to Q *in F is a sequence $P_1;\ldots;P_s$ of points in F, where $P = P_1$; $Q = P_s$ and P_t is a 6-neighbour of P_{t-1}, for $t = 2;\ldots;s$.*

It can be noted that the sequence with $s = 1$ is also a 6-path.

De nition 2. *F is* hex-connected *if, for each pair of F points, there is a 6-path in F that connects them.*

Finally,

De nition 3. *A hex-connected set is* horizontally, vertically and diagonally convex *if all its rows, columns and diagonals are hex-connected.*

We denote the class of hex-connected and horizontally, vertically and diagonally convex sets by $\mathcal{F}$. An element of this class corresponds to a convex polyomino with hexagonal cells. Fig. 3 shows a hex-connected discrete set and a discrete set of $\mathcal{F}$ with its corresponding hexagonal convex polyomino. This class of hexagonal polyominoes was studied by some authors in enumerative combinatorics [5,18].

3 The Reconstruction Algorithm

Let us now consider the reconstruction problem applied to class $\mathcal{F}$. Given $H = (h_1;\ldots;h_m)$; $V = (v_1;\ldots;v_n)$ and $D = (d_1;\ldots;d_{n+m-1})$, we want to establish the existence of a set $F \in \mathcal{F}$ such that $X_{(1;0)}F = H$; $X_{(0;1)}F = V$; $X_{(1;1)}F = D$. A necessary condition for the existence of this set is:

$$\sum_{i=1}^{m} h_i = \sum_{j=1}^{n} v_j = \sum_{k=1}^{n+m-1} d_k. \qquad (3.1)$$

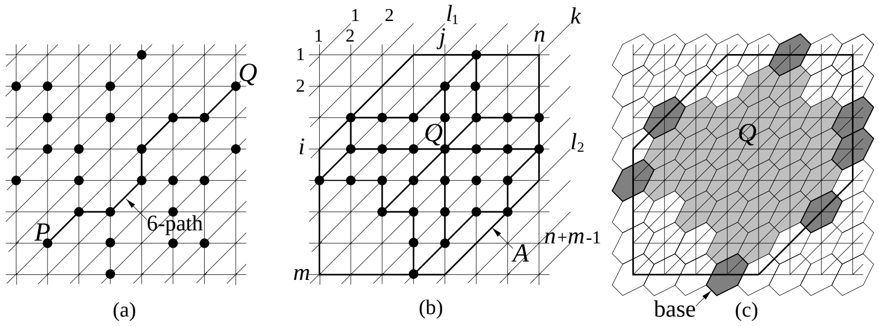

Fig. 3. a) A hex-connected set. b) A set of $\mathcal{F}$. c) Its corresponding hexagonal convex polyomino.

Without loss of generality, we can assume that $h_i \neq 0$ for $i = 1, \ldots, m$, and $v_j \neq 0$ for $j = 1, \ldots, n$. From this assumption and the definition of $\mathcal{F}$, it follows that there are two integers l_1 and l_2 such that:

$$\begin{aligned} &1 \leq l_1 \leq l_2 \leq n + m - 1; \\ &d_k \neq 0 \text{ for } k = l_1, \ldots, l_2; \\ &d_k = 0 \text{ for } k = 1, \ldots, l_1 - 1, l_2 + 1, \ldots, n + m - 1; \end{aligned} \tag{3.2}$$

this in turn means that F is contained in the discrete hexagon:

$$A = \{(i, j) \in \mathbb{N}^2 : 1 \leq i \leq m,\ 1 \leq j \leq n,\ l_1 \leq i + j - 1 \leq l_2\}.$$

A is the smallest discrete hexagon containing F. We wish to point out that the X-ray vector components are numbered starting from the left-upper corner (see Fig. 3b)). We call *bases* of F the points belonging to the boundary of A (see Fig. 3c)). Our aim is to determine which points of A belong to F. Let $Q = (i, j) \in A$; this point defines the following six zones (see Fig. 4):

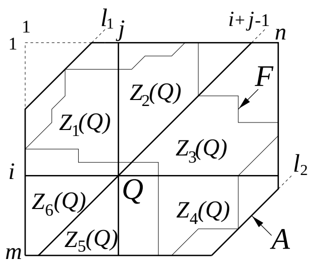

Fig. 4. The six zones determined by $Q = (i, j)$.

$$Z_1(Q) = f(r;c)\ 2\ A : r \quad i \text{ and } c \quad jg;$$
$$Z_2(Q) = f(r;c)\ 2\ A : j \quad c \text{ and } r + c \quad i + jg;$$
$$Z_3(Q) = f(r;c)\ 2\ A : r \quad i \text{ and } i + j \quad r + cg;$$
$$Z_4(Q) = f(r;c)\ 2\ A : i \quad r \text{ and } j \quad cg;$$
$$Z_5(Q) = f(r;c)\ 2\ A : c \quad j \text{ and } i + j \quad r + cg;$$
$$Z_6(Q) = f(r;c)\ 2\ A : i \quad r \text{ and } r + c \quad i + jg$$

Each zone contains Q. Moreover, from the de nition of the class F , it follows that, if Q does not belong to F, there are two consecutive zones which do not contain any point of F. For example, the point Q in Fig. 4 does not belong to F, and the intersection between F and $Z_5(Q)\ [\ Z_6(Q)$ is the empty set. By setting $Z_k(Q) = Z_k(Q)\ [\ Z_{k+1}(Q)$, with $k = 1;:::;5$, and $Z_6(Q) = Z_6(Q)\ [\ Z_1(Q)$, we obtain:

Property 1. Let Q be a point of the smallest discrete hexagon A containing a discrete set F of F . The point $Q\ 2\ F$ if and only if $Z_k(Q)\ \backslash\ F\ 6\!\!=\ ;$, for each $1 \quad k \quad 6$.

We can determine some F points just by referring to the geometry of A. Let $I_1 = (l_2 - n + 1;n);\ I_2 = (l_1;1);\ J_1 = (1;l_1);\ J_2 = (m;l_2 - m + 1);\ K_1 = (m;1);\ K_2 = (1;n)$. These points are the vertices of A as shown in Fig. 5. Let

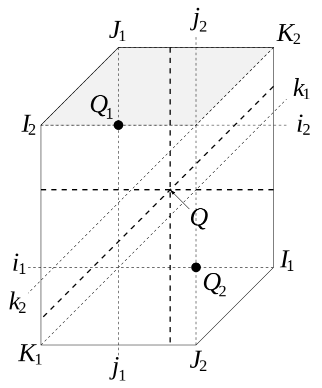

Fig. 5. The points Q_1 and Q_2 belonging to every discrete set of F contained in hexagon A.

$i_t;j_t$ and k_t be the row, column and diagonal index containing $I_t;\ J_t$ and K_t, respectively, with $t = 1;2$. Note that, $i_2 = j_1 = l_1$, $i_1 = l_2 - n + 1$, $j_2 = l_2 - m + 1$, $k_1 = m$ and $k_2 = n$. The hexagon illustrated in Fig. 5 is such that: $i_1 > i_2;\ j_1 < j_2$ and $k_1 > k_2$. Since A is the smallest discrete hexagon containing F, then the sides I_2J_1 and J_1K_2 contain at least a base of F. So, if $Q = (i;j)\ 2\ A$ is such that $i \quad i_2$ or $k \quad k_2$, with $k = i + j - 1$, then $Z_1(Q)\ \backslash\ F\ 6\!\!=\ ;$ (see Fig. 5). We proceed in the same way for the other ve pairs of consecutive sides:

$$\begin{aligned}
&i \text{ [illegible] } i_1 \text{ or } j \text{ [illegible] } j_1 \Rightarrow Z_2(Q) \cap F \neq \emptyset,\\
&j \text{ [illegible] } j_2 \text{ or } k \text{ [illegible] } k_2 \Rightarrow Z_3(Q) \cap F \neq \emptyset,\\
&i \text{ [illegible] } i_1 \text{ or } k \text{ [illegible] } k_1 \Rightarrow Z_4(Q) \cap F \neq \emptyset,\\
&i \text{ [illegible] } i_2 \text{ or } j \text{ [illegible] } j_2 \Rightarrow Z_5(Q) \cap F \neq \emptyset,\\
&j \text{ [illegible] } j_1 \text{ or } k \text{ [illegible] } k_1 \Rightarrow Z_6(Q) \cap F \neq \emptyset,
\end{aligned}$$

where $k = i + j - 1$. The points $Q_1 = (i_2, j_1)$ and $Q_2 = (i_1, j_2)$ verify these six inequalities and so, by Property 1, Q_1 and Q_2 belong to F. Fig. 6 illustrates

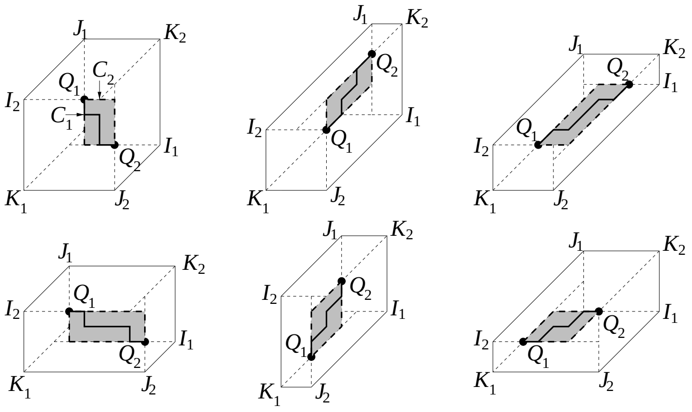

Fig. 6. The six con gurations of hexagon A.

the six allowed con gurations of A and the points Q_1 and Q_2 belonging to each discrete set of F contained in A. We can divide these con gurations into three groups:

a. if i_2 [illegible] i_1, j_1 [illegible] j_2, then $Q_1 = (i_2, j_1)$ and $Q_2 = (i_1, j_2)$ (see Fig. 6a));
b. if j_2 [illegible] j_1, k_2 [illegible] k_1, then $Q_1 = (k_1 - j_2 + 1, j_2)$ and $Q_2 = (k_2 - j_1 + 1, j_1)$ (see Fig. 6b));
c. if i_1 [illegible] i_2, k_1 [illegible] k_2, then $Q_1 = (i_2, k_1 - i_2 + 1)$ and $Q_2 = (i_1, k_2 - i_1 + 1)$ (see Fig. 6c)).

We refer to these con gurations as case a, case b and case c. We notice that, if $i_1 = i_2$, $j_1 = j_2$ and $k_1 = k_2$, we nd one point ($Q_1 = Q_2$) of F, and if $i_1 = i_2$ or $j_1 = j_2$ or $k_1 = k_2$, we nd more than two F points.
Let us now determine a 6-path from Q_1 to Q_2 made up of F points. In case a, $Q_1 = (i_2, j_1)$ and the two points $P_1 = (i_2 + 1, j_1)$ and $P_2 = (i_2, j_1 + 1)$ adjacent to Q_1 are such that:

$$Z_k(P_1) \cap F \neq \emptyset, \text{ for } k = 1,2,3,4,6, \quad \text{and} \quad Z_k(P_2) \cap F \neq \emptyset, \text{ for } k = 1,3,4,5,6,$$

(see Fig. 6a)). From Property 1, we can deduce that:

if $Z_5(P_1) \cap F \neq \emptyset$, then $P_1 \in F$, and if $Z_2(P_2) \cap F \neq \emptyset$, then $P_2 \in F$.

We prove that $P_1 \in F$ or $P_2 \in F$. Let us onsider the following cumulated sums of the row, column and diagonal X-rays:

$$\begin{aligned} H_0 &= 0;\ H_k = \textstyle\sum_{i=1}^{k} h_i; \quad k = 1, \ldots, m; \\ V_0 &= 0;\ V_k = \textstyle\sum_{i=1}^{k} v_i; \quad k = 1, \ldots, n; \\ D_{l_1-1} &= 0;\ D_k = \textstyle\sum_{i=1}^{k} d_i; \quad k = l_1, \ldots, l_2; \end{aligned}$$

and denote the common total sums of the row, column and diagonal X-rays by S. We have that:

Lemma 1. *Let $Q = (i, j)$ be a point of hexagon A containing a discrete set F of $\mathcal{F}$.*

– *If $H_i \geq S - D_{i+j-1}$, then $Z_1(Q) \cap F \neq \emptyset$.*
– *If $H_i \geq V_{j-1}$, then $Z_2(Q) \cap F \neq \emptyset$.*
– *If $S - D_{i+j-2} \geq V_{j-1}$, then $Z_3(Q) \cap F \neq \emptyset$.*
– *If $S - D_{i+j-2} \geq H_{i-1}$, then $Z_4(Q) \cap F \neq \emptyset$.*
– *If $V_j \geq H_{i-1}$, then $Z_5(Q) \cap F \neq \emptyset$.*
– *If $V_j \geq S - D_{i+j-1}$, then $Z_6(Q) \cap F \neq \emptyset$.*

Proof. We denote the first j columns and the first $i-1$ rows of the set F by C_j and R_{i-1}, respectively. If $Z_5(Q) \cap F = \emptyset$, then $C_j \subset R_{i-1}$ and so $V_j < H_{i-1}$ (see Fig 7). Therefore, if $V_j \geq H_{i-1}$, then $Z_5(Q) \cap F \neq \emptyset$. We obtain the other

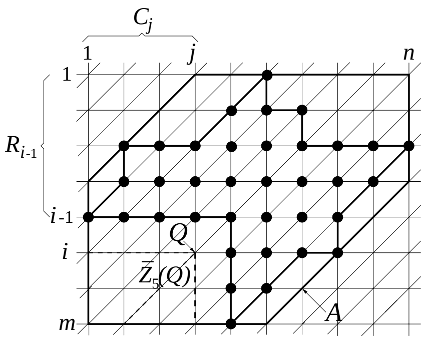

Fig. 7. A point $Q \notin F$ and $Z_5(Q) \cap F = \emptyset$.

statements in the same way.

Theorem 1. *There exists a 6-path from Q_1 to Q_2 made up of F points.*

Proof. By lemma 1 and the previous discussion:

if $V_{j_1} \le H_{i_2}$, then $P_1 = (i_2 + 1, j_1) \in F$ and if $H_{i_2} \le V_{j_1}$, then $P_2 = (i_2, j_1 + 1) \in F$

Since $V_{j_1} \le H_{i_2}$ or $H_{i_2} \le V_{j_1}$, we have $P_1 \in F$ or $P_2 \in F$. We can repeat the same operation on the two points adjacent to the point $P_k \in F$ (with $k = 1$ or $k = 2$) determined in the previous step. We wish to point out that if $V_{j_1} = H_{i_2}$, then $P_1 \in F$ and $P_2 \in F$. In this case, $Z_k(i_2 + 1, j_1 + 1) \cap F \neq \emptyset$, for each $1 \le k \le 6$. So $(i_2 + 1, j_1 + 1) \in F$ and we repeat the operation on its two adjacent points. We perform this procedure until it determines a point $P \in F$ which belongs to the row or the column containing $Q_2 = (i_1, j_2)$ (i.e., $P = (i_1, j)$ or $P = (i, j_2)$). Every point Q between P and $Q_2 = (i_1, j_2)$ is such that $Z_k(Q) \cap F \neq \emptyset$, for each $1 \le k \le 6$ and so $Q \in F$. By means of this procedure, we are able to find a 6-path from Q_1 to Q_2 made up of F points.

For example, if $H = (1, 4, 5, 7, 9, 7, 5, 3, 2)$ and $V = (1, 3, 3, 4, 6, 4, 6, 5, 4, 3, 3, 1)$, we obtain the 6-path from Q_1 to Q_2 shown in Fig. 8. We treat the other two cases in the same way. From Lemma 1, it follows that we have to use the cumulated sums V_j and $S - D_{i+j-1}$ in case b, and H_i and $S - D_{i+j-1}$ in case c.

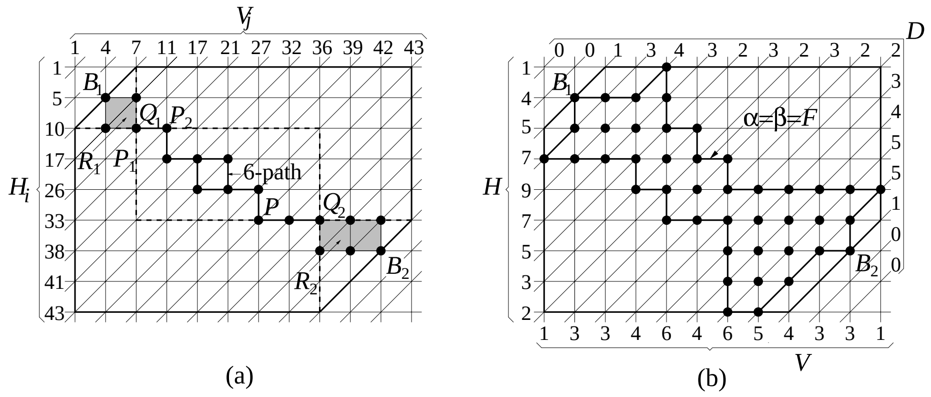

Fig. 8. a) The 6-path from Q_1 to Q_2 made up of F points. b) The discrete set $\alpha = \beta = F\ 2F$.

Let us now take the bases into consideration. Each side of hexagon A contains at least one base of F. Let B_1 and B_2 be a base of the sides I_2J_1 and a base of I_1J_2, respectively. In case a, B_i and Q_i, with $i = 1$ or $i = 2$, define a discrete rectangle R_i such that: if $Q \in R_i$, then $Z_k(Q) \cap F \neq \emptyset$, for each $1 \le k \le 6$ and so $Q \in F$ (see Fig. 8). Notice that R_i can degenerate into a discrete segment when B_i and Q_i belong to the same row or column. Therefore, we obtain a hex-connected set made up of F points and connecting two opposite sides of A. Unfortunately, we do not usually know the positions of the bases and so our algorithm chooses a pair of base-points (B_1, B_2) belonging to two opposite sides

of A, and then attempts to construct a discrete set F of F whose X-rays are equal to $H; V$ and D. If the reconstruction attempt fails, the algorithm chooses a di erent position of the base-points $(B_1; B_2)$ and repeats the reconstruction operations. More precisely, the algorithm determines Q_1, Q_2 and the 6-path from Q_1 to Q_2; after that it chooses:
- a point B_1 2 I_2J_1 and a point B_2 2 I_1J_2 in case a, or
- a point B_1 2 K_1J_2 and a point B_2 2 K_2J_1 in case b, or
- a point B_1 2 I_2K_1 and a point B_2 2 I_1K_2 in case c,
and then reconstructs the rectangle R_i de ned by Q_i and B_i, with $i = 1; 2$. I then uses the same reconstruction procedure described in the algorithm de ned in [3], that is, it performs the *lling operations* in the directions $(1; 0)$; $(0; 1)$ $(1; 1)$ and, if necessary, links our problem to the 2-Satis ability problem which can be solved in linear time [2].

We now describe the main steps of this reconstruction procedure. We call any set such that F, a *kernel*, and we call any set , such that F A, a *shell*. Assuming that is the reconstructed hex-connected set from B_1 to B_2 and is hexagon A, we perform the lling operations that expand and reduce . These operations take advantage of both the convexity constraint and vectors $H; V; D$, and they are used iteratively until 6 or and are invariant with respect to the lling operations.
If we obtain 6 , there is no discrete set of F containing B_1 and B_2 and having X-rays $H; V; D$. Therefore, the algorithm chooses another pair of base-points and performs the lling operations again.
If $=$, then $= F$ and so there is at least one solution to the problem (the algorithm reconstructs one of them). For example, by performing the lling operations on the hex-connected set from B_1 to B_2 in Fig 8, we obtain $=$, and is a discrete set having X-rays $H; V; D$.
Finally, if we obtain , then $-$ is a set of \indeterminate" points and we are not yet able to say that a set F having X-rays $H; V; D$ exists. Therefore, we have to perform another operation to establish the existence of F. At rst, is a hex-connected set from B_1 to B_2, where B_1 to B_2 belong to two opposite sides of A; therefore by performing the lling operations, we obtain:

- has at least one point in each diagonal of A in case a;
- has at least one point in each row of A in case b;
- has at least one point in each column of A in case c.

Assume that we have case b: there is at least one point of in each row of A, and so by the properties of the lling operations (see [3]), the length of the i-th row of is smaller than $2h_i$ for each 1 i m. If we are able to prove that:

I) the length of the j-th column of is equal to, or less than, $2v_j$ for each 1 j n;
II) the length of the k-th diagonal of is equal to, or less than, $2d_k$ for each 1 k $n + m - 1$,

there is a polynomial transformation of our reconstruction problem to the 2-Satis ability problem. We rst prove (I) and (II), and we then outline the main

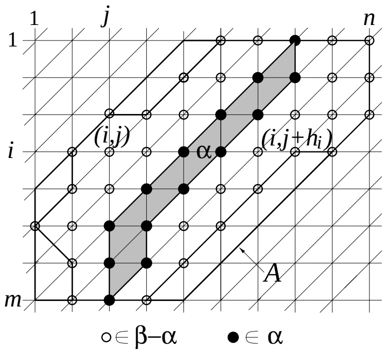

Fig. 9. The kernel, the shell and the set of the indeterminate points.

idea of the reduction to 2-Satis ability problem de ned in [3]. By the properties of lling operations, the indeterminate points follow each other into two sequences, one on the left side of   and the other on its right side. If $(i; j)$ is an indeterminate point of the left sequence, then $(i; j + h_i)$ belongs to the right sequence (see Fig. 9) and these ponts are related to each other; let us assume that there is a discrete set F of F having X-rays equal to $H; V; D$, then:

{ if $(i; j)$ 2 F, then $(i; j + h_i)$ @2 F,
{ if $(i; j)$ @2 F, then $(i; j + h_i)$ 2 F.

As a result, the number of indeterminate points belonging to F is equal to the number of indeterminate points not belonging to F. This means that, in order to the conditions given by horizontal X-ray be satis ed, half of the indeterminate points have to be in F. If there is at least a j such that j-th column of   is larger than $2v_j$, the number of its indeterminate points belonging to F has to be less than the number of its indeterminate points not belonging to F. Therefore, less than half of the indeterminate points are in F. We got a contradiction and so satis es (I). By proceeding in the same way, we prove that   satis es (II). Consequently, we can reduce our problem to a 2-Satis ability problem. We prove the same result for the cases a and c in the same way. Therefore, the algorithm solves the problem for all $H; V; D$ instances.

We can summarize the reconstruction algorithm as follows.

```
Input: Three vectors $H$ 2 $\mathbb{N}^m$, $V$ 2 $\mathbb{N}^n$ and $D$ 2 $\mathbb{N}^{n+m-1}$;
Output: a discrete set of $F$ such that $X_{(1;0)}F = H; X_{(0;1)}F = V; X_{(1;1)}F = D$
or a message that there is no a set like this;
1. check if $H; V$ and $D$ satisfy conditions (3.1)and (3.3);
2. compute the points $Q_1$ and $Q_2$;
3. compute the cumulated sums $H_i; V_j; D_k$, for $i = 1; : : : ; m; j = 1; : : : ; n$ and
$k = 1; : : : ; n + m - 1$;
4. compute the 6-path from $Q_1$ to $Q_2$;
5. repeat
```

```
        5.1. choose a pair of base-points $(B_1; B_2)$ belonging to two
        opposite sides of $A$;
        5.2. compute the rectangles $R_1$ and $R_2$;
        5.3.   $:= R_1 [ R_2 [$ the 6-path from $Q_1$ to $Q_2$ ;
        5.4.   $:= A$;
        5.5. repeat
                    5.5.1. perform the  lling operations;
             until   $6$   or   $;$   are invariants;
        5.7. if   $=$   then $F =$   is a solution;
        5.8. if         then reduce our problem to 2SAT;
   until there is a discrete set of $F$ having X-rays equal to $H; V; D$ or all the
   base-point pairs have been examined.
```

We now examine the complexity of the algorithm described. Determining the hex-connected set from B_1 to B_2 (i.e., Q_1, Q_2, the 6-path from Q_1 to Q_2, and the rectangles R_1 and R_2) involves a computational cost of $O(nm)$. In [13], the author proposes a simple procedure for performing the lling operations whose computational cost is $O(nm(n + m))$. This procedure gives a kernel and a shell invariant with respect to the lling operations. If we obtain   , the algorithm transforms our problem into a 2-Satis ability problem and solves it in $O(nm)$ time. In case of failure, that is, when there is no discrete set of F containing B_1 and B_2 and having X-rays $H; V; D$, the algorithm chooses another pair of base-points and performs the lling operations again. At most, it has to check all the possible base-point pairs, that is $O((n + m)^2)$ pairs. Consequently, the algorithm decides if there is a discrete set of F having X-rays $H; V; D$; if so, the algorithm reconstructs one of them in $O(nm(n + m)^3)$ time.

Theorem 2. *Consistency$((1; 0); (0; 1); (1; 1))$ on F can be solved in $O(nm(n + m)^3)$ time.*

Remark 1. The algorithm can be easily extended to contexts having more than three X-rays and can reconstruct discrete sets convex in the directions of the X-rays. This means that Consistency$((1; 0); (0; 1); (1; 1); u_4; \ldots; u_k)$ on the class of connected sets, which are convex in all the directions $(1; 0); (0; 1); (1; 1); u_4; \ldots; u_k$, is solvable in polynomial time.

References

1. R. P. Anstee, \Invariant sets of arcs in network flow problems," *Disc. Appl. Math.* **13**, 1-7 (1986).
2. B. Aspvall, M.F. Plass and R.E. Tarjan,\ A linear-time algorithm for testing the truth of certain quanti ed Boolean formulas", *Inf. Proc. Lett.*, **8**, 3 (1979).
3. E. Barcucci, A. Del Lungo, M. Nivat and R. Pinzani,\ Reconstructing convex polyominoes from their horizontal and vertical projections", *Theor. Comp. Sci.*, **155** (1996) 321-347.
4. M. Chrobak, C. Dürr, \Reconstructing hv-Convex Polyominoes from Orthogonal Projections", *Inf. Proc. Lett.* **(69) 6**, (1999) 283-289.

5. A. Denise, C. Dürr and F.Ibn-majdoub-Hassani, \Enumeration et generation aleatoire de polyominos convexes en reseau hexagonal", *Proc. of the* 9^{th} *FPSAC*, Vienna, 222-235 (1997).
6. R. J. Gardner, *Geometric Tomography*, Cambridge University Press, Cambridge, UK, 1995, p.51.
7. R. J. Gardner and P. Gritzmann, \Uniqueness and Complexity in Discrete Tomography", in *discrete tomography: foundations, algorithms and applications*, editors G.T. Herman and A. Kuba, Birkhauser, Boston, MA, USA, (1999) 85-111.
8. R. J. Gardner, P. Gritzmann and D. Prangenberg, \On the computational complexity of reconstructing lattice sets from their X-rays," *Disc. Math.* 202, (1999) 45-71.
9. M. R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, New York, (1979) 224.
10. P. Gritzmann, \Open problems," in *Discrete Tomography: Algorithms and Complexity*, Dagstuhl Seminar report 165 (1997) 18.
11. P. C. Hammer, \Problem 2," in *Proc. Simp. Pure Math. vol VII: Convexity*, Amer. Math. Soc., Providence, RI, (1963) 498-499.
12. G.T. Herman and A. Kuba, \Discrete Tomography: A Historical Overview", in *discrete tomography: foundations, algorithms and applications*, editors G.T. Herman and A. Kuba, Birkhauser, Boston, MA, USA, (1999) 3-29.
13. M. Gebala :The reconstruction of convex polyominoes from horizontal and vertical projections, *private comunication*.
14. R. W. Irving and M. R. Jerrum, \Three-dimensional statistical data security problems," *SIAM Journal of Computing* 23, 170-184 (1994).
15. C. Kisielowski, P. Schwander, F. H. Baumann, M. Seibt, Y. Kim and A. Ourmazd, \An approach to quantitative high-resolution transmission electron microscopy of crystalline materials," *Ultramicroscopy* 58, 131-155 (1995).
16. G. P. M. Prause and D. G. W. Onnasch, \Binary reconstruction of the heart chambers from biplane angiographic image sequence," *IEEE Trans. Medical Imaging* 15, 532-559 (1996).
17. A. R. Shliferstein and Y. T. Chien, \Switching components and the ambiguity problem in the reconstruction of pictures from their projections," *Pattern Recognition* 10, 327-340 (1978).
18. X. G. Viennot, A Survey of polyomino enumeration, *Proc. Series formelles et combinatoire algebrique*, eds. P. Leroux et C. Reutenauer, Publications du LACIM 11, Universite du Quebec a Montreal (1992).

Modi ed Binary Searching for Static Tables

Donatella Merlini, Renzo Sprugnoli, and M. Cecilia Verri

Dipartimento di Sistemi e Informatica - Universita di Firenze
Via C. Lombroso 6/17 - 50134 Firenze - Italy
`fmerlini, sprugnoli, verrig@dsi.unifi.it`

Abstract. We consider a new method to retrieve keys in a static table. The keys of the table are stored in such a way that a binary search can be performed more e cently. An analysis of the method is performed and empirical evidence is given that it actually works.

Keywords: binary searching, static dictionary problem.

1 Introduction

The present paper was motivated by some obvious observations concerning binary searching when applied to static sets of strings; by following the directions indicated by these observations, we obtained some improvements that reduce execution time for binary searching of 30 − 50%, and also more for large tables.

Let us consider the set of the zodiac names

$$S = f\texttt{capricorn}; \texttt{acquarius}; \texttt{pisces}; \texttt{aries}; \texttt{taurus}; \texttt{gemini}; \texttt{cancer}; \texttt{leo};$$

$$\texttt{virgo}; \texttt{libra}; \texttt{scorpio}; \texttt{sagittarius}g$$

and consider the problem of nding out if identi er x belongs or does not belong to $S:$ To do this, we can build a lexicographically ordered table T with the names in S and then use binary searching to establish whether $x\ 2\ S;$ or not (see Table 1(a)). What is ne in binary searching is that, if the program is properly realized, the number of character comparisons is taken to a minimum. For instance, the case of `aries` is as follows: 1 comparison (`a` against `l`) is used to compare `aries` and the median element `leo`; 1 comparison (`a` against `c`) is used to distinguish `aries` from `cancer`; 2 comparisons (`ar` against `ac`) are used for `aries` against `acquarius`. After that, the searched string is compared to the whole item `aries`.

The problem here is that we do not know in advance how many characters have to be used in each string-to-string comparison; therefore, every string comparison requires a (comparatively) high number of housekeeping instructions, which override the small number of character comparisons and make the matching procedure relatively time consuming.

The aim of this paper is to show that the housekeeping instructions can be almost eliminated by arranging the elements in a suitable way. First of all we have to determine the minimal number of characters able to distinguish the elements

Table 1. Two table's structures for zodiac names

	(a)	(b): s=1
1	acquarius	1 $\boxed{\text{a}}$cquarius
2	aries	1 $\boxed{\text{c}}$ancer
3	cancer	2 l$\boxed{\text{e}}$o
4	capricorn	1 $\boxed{\text{l}}$ibra
5	gemini	1 $\boxed{\text{p}}$isces
6	leo	5 capr$\boxed{\text{i}}$corn
7	libra	1 $\boxed{\text{s}}$agittarius
8	pisces	1 $\boxed{\text{t}}$aurus
9	sagittarius	2 s$\boxed{\text{c}}$orpio
10	scorpio	1 $\boxed{\text{a}}$ries
11	taurus	1 $\boxed{\text{g}}$emini
12	virgo	1 $\boxed{\text{v}}$irgo

of the set: in the previous example the first three characters are sufficient to univocally distinguish strings in S and this is not casual. In fact, if we have a set S containing n strings on the alphabet with j $j =$ $;$ what we are expecting is that $d$log ne characters actually distinguish the elements in $S.$ The problem is that in general these characters can be in different positions and we have to determine, string by string, where they are. What we wish to show here is that for any given set S, it is relatively easy to find out whether the elements in S can be distinguished using a small amount of characters. In that case, we are then able to organize the set S in a table T such that a modified binary searching can be performed with a minimum quantity of housekeeping instructions, thus improving the traditional searching procedure. Presently, we are also investigating how modified binary search compares to the method proposed by Bentley and Sedgewick [1]. To be more precise, in Section 2 we show that a pre-processing algorithm exists which determines (if any) the optimal organization of the table for the given set S; the time for this pre-processing phase is in the order of $n(\log n)^2.$ Then, in Section 3, we find the probability that the pre-processing algorithm gives a positive result, i.e., it finds the optimal table organization for set $S.$ Finally, in Section 4, an actual implementation of our method is discussed and empirical results are presented showing the improvement achieved over traditional binary searching.

2 The Pre-processing for Modified Binary Search

Let us consider a set S of n *elements* or *keys*, each u characters long (some characters may be blank). We wish to store these elements in a table T in which binary searching can be performed with a minimum quantity of housekeeping instructions. To this purpose we need a vector $V[1 \ldots n]$ containing, for every $i = 1, 2, \ldots, n$, the position from which the comparison, relative to the element

$T[i]$, has to begin. A possible Pascal-like implementation of this Modified Binary Searching (MBS) method can be the following:

```
function MBS(str:string): integer;
var $a; b; k; p$: integer; found:boolean;
begin $a := 1; b := n$; found:=false;
    while $a \le b$ do
        $k := \lfloor (a + b)/2 \rfloor$;
        $p$:=Compare(str, $T[k]; V[k]; s$);
        if $p = 0$
            then found:=true; $a := b + 1$
            else if $p < 0$
                then $b := k - 1$
                else $a := k + 1$        od;
    if found and $T[k]$ = str then MBS:=$k$ else MBS:=0
end;
```

Here, the function *Compare(A,B,i,s)* compares the two strings A and B according to s characters beginning from position i; the result is -1, 0 or 1 according to the fact that the s characters in A are lexicographically smaller, equal or greater than the corresponding characters in B.

The procedure, we are going to describe to build table T and vector V, will be called *PerfectDivision(S,n,s)*, where S and n are as specified above and s is the number of consecutive characters used to distinguish the various elements (in most cases $1 \le s \le 4$ is sufficient). It returns a table T of dimension n; the *optimal table*, containing S's elements and a vector V of dimension n such that, for all i from 1 to n; $V[i]$ indicates the position from which the comparison has to begin. More precisely, the elements in table T are arranged in such a way that, if we binary search for the element $x = x_1 \ldots x_u$ and compare it with element $T[i] = T[i]_1 \ldots T[i]_u$ we have only to compare characters $x_{V[i]} \ldots x_{V[i]+s-1}$ with characters $T[i]_{V[i]} \ldots T[i]_{V[i]+s-1}.$

The procedure consists of two main steps, the first of which tries to find out the element in S which determines the subdivision of the keys into two subtables. The second step recursively calls *PerfectDivision* to actually construct the two subtables.

We begin by giving the definition of a *selector*:

Definition 1 *Let S be a set containing n u-length strings. The pair $(i; \sigma^{[s]} = a_1a_2 \ldots a_s)$; $i \in [1 \ldots u-s+1]$; is an $s^{[i]}$-selector for S iff :*

1) $\exists! w \in S : w_i w_{i+1} \ldots w_{i+s-1} = a_1a_2 \ldots a_s$;
2) $\exists \lfloor (n-1)/2 \rfloor$ keys $y \in S : y_i y_{i+1} \ldots y_{i+s-1} < a_1a_2 \ldots a_s$;
3) $\exists \lceil (n-1)/2 \rceil$ keys $y \in S : y_i y_{i+1} \ldots y_{i+s-1} > a_1a_2 \ldots a_s.$

Theorem 1 *Let S be a set containing n u-length strings. If $n = 1$ or $n = 2$ then S always admits a selector.*

Proof: If $S = \{x\}$ then it admits the $1^{[1]}$-selector $(1, x_1)$. If $S = \{x, y\}$, it admits a $1^{[p]}$-selector (p, z_p) with $p = \min\{i \in [1 \ldots u] : x_i \neq y_i\}$ and $z_p = \min\{x_p, y_p\}$ (such a p exists since x and y are distinct elements). We observe that if S admits a $1^{[p]}$-selector then it also admits an $s^{[q]}$-selector $\forall p + 1 - s \quad q \quad p.$ ■

More in general, for $n > 2$ we can give an integer function *FindSelector(*var *T, a, b, s)* to determine whether or not the elements from position a to position b in a table T containing $b - a + 1$ u-length strings, admit an s-selector; after a call to this function, if the selector has been found and $k = \lfloor (a + b)/2 \rfloor$ is the index of the median element, then the elements in T are arranged in such a way that:

1. the median element $T[k]$ contains $^{[s]}$ starting at position i;
2. the elements $T[a], \ldots, T[k - 1]$ are less than $T[k]$ when we compare the characters from position i to position $i + s - 1$ with $^{[s]}$;
3. the elements $T[k + 1], \ldots, T[b]$ are greater than $T[k]$ when we compare the characters from position i to position $i + s - 1$ with $^{[s]}$.

The function returns the position $i = sel$ at which the selector begins, if any, 0 otherwise. This value is stored in vector V from procedure *PerfectDivision* after *FindSelector* is called.

FindSelector calls, in turn, a procedure, *Select(*var *T, a, b, sel, s)*, and a boolean function, *Equal(x, y, sel, s)*, which operate in the following way:

– procedure *Select* arranges the elements in table T, from position a to position b, by comparing, for each element, the s characters beginning at position sel; the arrangement is such that the three conditions 1., 2. and 3. above are satis ed. As is well known, the selection problem to nd the median element can be solved in linear time; however, in our algorithm we decided to use a heapsort procedure to completely sort the elements in T. This is slower, but safer[1] than existing selection algorithms, and the ordering could be used at later stages;
– function *Equal* compares the s characters of the strings x, y, beginning at position sel, and returns *true* if the compared characters are equal, *false* otherwise. It corresponds to *Compare(x,y,sel,s)* = 0 above.

```
Function FindSelector(var T,a,b,s): integer;
var $sel, i$: integer;
begin
$sel$ := 1;
if $b - a$   2 then
  FindSelector:=0;
  $i := \lfloor (a + b)/2 \rfloor$;
```

[1] The algorithm of Rivest and Tarjan [3, vol. 3, pag. 216], can only be applied to large sets; the algorithm of selection by tail recursion [2, pag. 230] is linear on the average but is quadratic in the worst case.

```
    while $sel$   $(u - s + 1)$ do
        Select(T, a, b, sel, s);
        if $Equal(T[i]; T[i+1]; sel; s)$ or $Equal(T[i]; T[i-1]; sel; s)$ then
            $sel := sel + 1$
        else
            FindSelector:=$sel$;
            $sel := u + 1$ $f$to force exit from the loop$g$

    od
else if $b - a = 1$ then
    while $sel$   $(u - s + 1)$ do
        Select(T,a,b,sel,s);
        if $Equal(T[a]; T[b]; sel; s)$ then
            $sel := sel + 1$
        else
            FindSelector:=1;
            $sel := u + 1$ $f$to force exit from the loop$g$

    od
    FindSelector:=$sel$
else
    FindSelector:=$sel$

end $f$FindSelector$g$;

Procedure PerfectDivision(var T,a,b,s);
var $k; i$: integer;
begin
$k$ :=FindSelector(T, a, b, s);
if $k = 0$ then
        fail
else
        $i := b(a + b)=2c$;
        $V[i] := k$;
        if $a < i$ then PerfectDivision(T, a, i-1, s)   ;
        if $i < b$ then PerfectDivision(T, i+1, b, s)

end $f$PerfectDivision$g$;
```

If we want to apply *PerfectDivision* to a set S of n u-length strings, we rst store S's elements in a table T, hence call *PerfectDivision(T,1,n,s)* by choosing an appropriate value for s (in practice, we can start with $s = 1$ and then increase its value if the procedure fails).

This procedure applied to the set of zodiac names by using $s = 1$ returns the table T and the vector V as in Table 1(b). Let us take into consideration again the case of `aries`: we start with the median element and since $V[6] = 5$ we have to compare the fth character of `aries`, `s`, with the fth character of `capricorn`, `i`; `s>i` and we proceed with the new median element `scorpio`. Since $V[9] = 2$ we compare `r` with `c`; in the next step the median element is `gemini` and we compare the rst character of `aries` with the rst one of `gemini`. After that, the searched string is compared to the whole string `aries` and the procedure ends with success.

For the set of the rst twenty numbers names' the procedure fails by using $s = 1$ and returns the situation depicted in Table 2 by choosing $s = 2$:

Table 2. Table T and vector V when $s = 2$ for the rst twenty numbers' names

	V	T
1	1	$\boxed{\text{te}}$n
2	3	th$\boxed{\text{re}}$e
3	5	seve$\boxed{\text{n}}$
4	5	seve$\boxed{\text{nt}}$een
5	2	n$\boxed{\text{in}}$e
6	1	$\boxed{}$ve
7	1	$\boxed{\text{on}}$e
8	1	$\boxed{\text{si}}$x
9	1	$\boxed{\text{tw}}$o
10	4	nin$\boxed{\text{et}}$een
11	1	$\boxed{\text{th}}$irteen
12	2	f$\boxed{\text{if}}$teen
13	5	eigh$\boxed{\text{t}}$
14	5	eigh$\boxed{\text{te}}$en
15	2	s$\boxed{\text{ix}}$teen
16	3	tw$\boxed{\text{el}}$ve
17	3	tw$\boxed{\text{en}}$ty
18	3	el$\boxed{\text{ev}}$en
19	4	fou$\boxed{\text{r}}$
20	4	fou$\boxed{\text{rt}}$een

The complexity of *PerfectDivision* is given by the following Theorem:

Theorem 2 *If S is a set of words having length n and we set $d = u - s + 1$; then the complexity of the procedure PerfectDivision is at most $\ln 2dn \log_2^2 n$ if we use Heapsort as a selection algorithm, and $O(dn \log_2 n)$ on the average if we use a linear selection algorithm.*

Proof: The most expensive part of *PerfectDivision* is the selection phase, while all the rest is performed in constant time. If we use HeapSort as an in-place sorting procedure, the time is $An \log_2 n$; where A $2 \ln 2$: This procedure is executed at most d times. Let C_n be the total time for finding a complete division for our set of n elements; clearly we have:

$$C_n \le Adn\log_2 n + 2C_{n/2}.$$

There are general methods, see [4], to find a valuation for C_n; however, let us suppose $n = 2^k$, for some k, and set $c_k = C_{2^k}$; then we have $c_k = Ad2^k k + 2c_{k-1}$. By unfolding this recurrence, we find:

$$c_k = Ad2^k k + 2Ad2^{k-1}(k-1) + 4Ad2^{k-2}(k-2) + \dots =$$

$$= Ad2^k(k + (k-1) + \cdots + 1) = Ad2^{k-1}k(k+1).$$

Returning to C_n :

$$C_n \le Ad\frac{n}{2}\log_2 n(\log_2 n + 1) = O(dn(\log_2 n)^2).$$

Obviously, if we use a linear selection algorithm the time is reduced by a factor $O(\log_2 n)$. Finally, we observe that $d \le \log_2 n$ and this justifies our statement in the Introduction. ■

The next section is devoted to the problem of evaluating the probability that a perfect subdivision for the table T exists.

3 The Analysis

We now perform an analysis of the pre-processing phase of modified binary searching. Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_\sigma\}$ be a finite alphabet and let its elements be ordered in some way, for instance $\sigma_1 < \sigma_2 < \dots < \sigma_\sigma$. In real situations, Σ can be the Latin alphabet, $\sigma = 26$ or $\sigma = 27$ (if we also consider the space) and its ordering is the usual lexicographical order; so, on a computer, Σ is a subset of ASCII codes and if we have $s = 3$ then we should consider triples of letters. We can now abstract and define $A = \Sigma^s$, for the specific value of s, with the order induced by the order in Σ. Then we obscure s and set $A = \{a_1, a_2, \dots, a_r\}$, where $r = \sigma^s$. Finally, we observe that if S is any set of strings, $S \subseteq \Sigma^u$, for a given starting index the subwords of the words in S beginning at that position and composed by s consecutive characters are a multiset S over A. What we are looking for, is a suitable division of this multiset.

Therefore, let us consider the multiset S, with $|S| = n$, over our abstract alphabet A. Our problem is to find the probability that an element $a_m \in A$ exists such that: i) $a_m \in S$, but has no duplicate in S; ii) there are exactly $\lfloor(n+1)/2\rfloor - 1$ elements $a_i \in S$ such that $a_i < a_m$ (i.e., these elements constitute a multiset over $\{a_1, a_2, \dots, a_{m-1}\}$); iii) there are exactly $\lceil(n+1)/2\rceil$ elements a_j in S such that $a_j > a_m$ (i.e., these elements constitute a multiset over $\{a_{m+1}, \dots, a_r\}$). In a more general way we can define the following three (m, p)-*separation conditions* for a multiset S as above, relative to an element $a_m \in A$ (see also Definition 1):

i) $a_m \in S$ and has no duplicate in S;
ii) there are $(p-1)$ elements in S preceding a_m;
iii) there are $(n-p)$ elements in S following $a_m.$

Our first important result is the following:

Theorem 3 *Let* $(n;p;r;m)$ *be the probability that a multiset* S *on* A *(*S *and* A *as above) contains a specific element* a_m *for which the* $(m;p)$*-separation conditions are satisfied; then*

$$(n;p;r;m) = \frac{p}{r^n}\binom{n}{p}(m-1)^{p-1}(r-m)^{n-p}.$$

Proof: Let us count the total number $P(n;p;r;m)$ of the multisets satisfying the three $(m;p)$-separation conditions for a given element $a_m \in A$ $(1 \leq m \leq r)$. If we imagine that the elements in S are sorted according to their order sequence in A, the first $p-1$ elements must belong to the subset $\{a_1; a_2; \ldots; a_{m-1}\}$, and therefore $(m-1)^{p-1}$ such submultisets exist. In the same way, the last $n-p$ elements in S must belong to $\{a_{m+1}; \ldots; a_r\}$ and therefore $(r-m)^{n-p}$ such submultisets exist. Since every first part can be combined with each second part, and the pth character must equal a_m by hypothesis, there exists a total of $(m-1)^{p-1}(r-m)^{n-p}$ ordered multisets of the type described. The original multisets, however, are not sorted, and the elements may assume any position in S. The different combinations are counted by a simple trinomial coefficient, which reduces to a binomial coefficient:

$$\binom{n}{p-1;1;n-p} = \frac{n!}{(p-1)!\,1!\,(n-p)!} = p\frac{n!}{p!\,(n-p)!} = p\binom{n}{p}$$

We conclude that the total number of multisets we are looking for is:

$$P(n;p;r;m) = p\binom{n}{p}(m-1)^{p-1}(r-m)^{n-p}.$$

Finally, the corresponding probability is found by dividing this expression by r^n; the total number of multisets with n elements. ■

An immediate corollary of this result gives us the probability that, given S; an element a_m exists that satisfies the $(m;p)$-separation conditions:

Theorem 4 *Given a multiset* S *as above, the probability* $(n;p;r)$ *that an element* $a_m \in A$ *exists for which the* $(m;p)$*-separation conditions hold true is:*

$$(n;p;r) = \frac{p}{r^n}\binom{n}{p}\sum_{m=1}(m-1)^{p-1}(r-m)^{n-p}$$

Proof: Obviously, any $a_m \in A$ could satisfy the separation conditions, each one in an independent way of any other (a single multiset can contain several elements satisfying the conditions). Therefore, the probability is the sum of the single probabilities. ■

The formula of this theorem is not very appealing and does not give an intuitive idea of how the probability $\Pi(n;p;r)$ varies with p; in particular, what it is when $p = \lfloor (n+1)/2 \rfloor$; the case we are interested in. In order to arrive at a more intelligible formula, we are going to approximate it by obtaining its asymptotic value. To do that, we must suppose $n < r$; and this hypothesis will be understood in the sequel. In real situations, where $r = |A| = 26^s$ or $r = |A| = 27^s$; we simply have to choose a selector of s character with:

- $s = 1$ for tables up to $n = 22$ elements;
- $s = 2$ for tables up to $n = 570$ elements;
- $s = 3$ for tables up to $n = 15{,}000$ elements;
- $s = 4$ for tables up to $n = 400{,}000$ elements.

These numbers correspond to $n/r \approx 0.85$ if $|A| = 26^s$ and to $n/r \approx 0.8$ if $|A| = 27^s$. Obviously, a selector with $s > 4$ can result in a worsening with respect to traditional binary searching. On the other hand, very large tables should reside in secondary storage.

A curious fact is that the probabilities $\Pi(n;p;r)$ are almost independent of p; as we are now going to show, the dependence on p can only be appreciated when p is very near to 1 or very near to n.

Theorem 5 *The probability* $\Pi(n;p;r)$ *of the previous theorem has the following asymptotic approximations* $(n < r)$*:*

$$\Pi(n;p;r) = \left(1 - \frac{1}{r}\right)^n \left(1 + O\left(\frac{n^3}{r^3}\right)\right) \qquad \text{when} \quad 2 < p < n-2; \tag{3.1}$$

$$\Pi(n;2;r) = \Pi(n;n-1;r) = \left(1 - \frac{1}{r}\right)^n \left(1 - \frac{n(n-1)}{12(r-1)^2} + O\left(\frac{n^3}{r^3}\right)\right); \tag{3.2}$$

$$\Pi(n;1;r) = \Pi(n;n;r) =$$
$$= \left(1 - \frac{1}{r}\right)^n \left(1 - \frac{n}{2(r-1)} - \frac{n(n-1)}{12(r-1)^2} + O\left(\frac{n^3}{r^3}\right)\right). \tag{3.3}$$

Proof: Let us apply the Euler-McLaurin summation formula to approximate the sum $\sum_{m=1}^{r}(m-1)^{p-1}(r-m)^{n-p}$. By writing m as the continuous variable x, we have:

$$\sum_{m=1}^{r}(m-1)^{p-1}(r-m)^{n-p} = \sum_{m=1}^{r-1}(m-1)^{p-1}(r-m)^{n-p} =$$

$$= \int_1^r (x-1)^{p-1}(r-x)^{n-p}dx + B_1[f(x)]_1^r + \frac{B_2}{2!}[f'(x)]_1^r +$$

Here, $f(x) = (x-1)^{p-1}(r-x)^{n-p}$, and we immediately observe that for $p > 1$ we have $[f(x)]_1^r = 0$. In the same way, the first $(p-1)$ derivatives are 0 and, in order to find suitable approximations for $(n;p;r)$, we are reduced to the three cases $p = 1; p = 2$ and $p > 2$. By symmetry, we also have $(n;p;r) = (n;n-p+1;r)$.

Since $f(x)$ is a simple polynomial in x, the integral can be evaluated as follows:

$$\int_1^r (x-1)^{p-1}(r-x)^{n-p}dx = \int_0^{r-1} y^{p-1}(r-1-y)^{n-p}dy =$$

$$= \int_0^{r-1} y^{p-1}\left(\sum_{k=0}^{n-p}\binom{n-p}{k}(r-1)^k(-y)^{n-p-k}\right)dy =$$

$$= \sum_{k=0}^{n-p}\binom{n-p}{k}(r-1)^k(-1)^{n-p-k}\int_0^{r-1} y^{n-k-1}dy =$$

$$= \sum_{k=0}^{n-p}\binom{n-p}{k}(r-1)^k(-1)^{n-p-k}\frac{(r-1)^{n-k}}{n-k} = (r-1)^n\sum_{k=0}^{n-p}\binom{n-p}{k}\frac{(-1)^k}{p+k} =$$

$$= (r-1)^n\frac{1}{p\binom{n-p+p}{n-p}} = \frac{(r-1)^n}{p}\binom{n}{p}^{-1}$$

The last sum in this derivation is well-known and represents the inverse of a binomial coefficient (see Knuth [3, vol. 1, pag. 71]). Therefore we obtain formula (3.1) for $2 < p < n-2$: For $p = 2$ we have a contribution from the first derivative:

$$[f'(x)]_1^r = \left[(r-x)^{n-2} - (x-1)(n-2)(r-x)^{n-3}\right]_1^r = -(r-1)^{n-2}$$

and therefore we have formula (3.2). Finally, for $p = 1$ we also have a contribution from $[f(x)]_1^r$; and find formula (3.3). ■

Table 3 illustrates the probabilities $(n;p;r)$ for $n = 12$ and $r = 20$. We used a computer program for simulating the problem, and in the table we show the probabilities found by simulation, the exact probabilities given by Theorem 4, and the approximate probabilities as computed by formulas (3.1), (3.2) and (3.3).

Formula (3.1), in the short version $(n;p;r) = (1-1/r)^n$; allows us to obtain an estimate of the probability of finding a division with $p = \lfloor(n+1)/2\rfloor$ for a given set S of identifiers or words over some language. In general, several positions d of the identifiers are available for a division (function *FindSelector* finds the first one), and this increases the probability of success. Usually, if s is the length of the selector and u the identifiers' length, we have $d = u - s + 1$; and in general:

Table 3. Simulation, exact and approximate probabilities

SIMULATION FOR MODIFIED BINARY SEARCHING

Number of alphabet elements (r): 20
Number of multiset elements (n): 12
Number of simulation trials: 30000

p	simulat.	exact	approx.
1	0.72176667	0.72739722	0.72746538
2	0.51906667	0.52409881	0.52389482
3	0.54113333	0.54015736	0.54036009
4	0.53856667	0.54042599	0.54036009
5	0.53563333	0.54036133	0.54036009
6	0.54176667	0.54035984	0.54036009
7	0.54300000	0.54035984	0.54036009
8	0.54173333	0.54036133	0.54036009
9	0.53923333	0.54042599	0.54036009
10	0.53963333	0.54015736	0.54036009
11	0.52596667	0.52409881	0.52389482
12	0.72970000	0.72739722	0.72746538

Theorem 6 *Let S be a multiset of identi ers over A, with $|S| = n$ and $|A| = r$. If d positions in the elements of S are available for division, then the probability that an alement $a_m \in S$ exists satisfying the (m,p)-separation conditions with $p = \lfloor (n+1)/2 \rfloor$ is:*

$$(n;r;d) = 1-(1- \quad (n;p;r))^d \quad 1- \quad 1- \quad 1-\frac{1}{r} \quad ^{n \quad d} : \tag{3.4}$$

This quantity can be approximated by:

$$(n;r;d) \quad 1- \quad \frac{n}{r} \quad ^d \exp \quad -\frac{d(n-1)}{2r} \quad : \tag{3.5}$$

Proof: We can consider the d positions as independent of each other, and therefore equation (3.4) follows immediately. To prove (3.5) we need some computations. First of all we have:

$$1-\frac{1}{r} \quad ^n = \exp \quad n\ln \quad 1-\frac{1}{r} \quad = \exp \quad n \quad -\frac{1}{r}-\frac{1}{2r^2}-\frac{1}{3r^3}- \qquad =$$

$$= \exp \quad -\frac{n}{r}-\frac{n}{2r^2}-\frac{n}{3r^3}- \qquad = e^{-n/r}e^{-n/2r^2}e^{-n/3r^3}$$

By expanding the exponentials, we nd:

$$1- \quad 1-\frac{1}{r} \quad ^n \qquad 1-e^{-n/r}e^{-n/2r^2}e^{-n/3r^3} \qquad =$$

$$= 1 - \quad 1 - \frac{n}{r} + \frac{n^2}{2r^2} - \frac{n^3}{6r^3} + \qquad 1 - \frac{n}{2r^2} + \qquad 1 - \frac{n}{3r^3} + \qquad =$$

$$= \frac{n}{r} \quad 1 - \frac{n-1}{2r} + \frac{n^2 - 3n + 2}{6r^2} + O \quad \frac{n^3}{r^3} \qquad :$$

This formula can be written as:

$$1 - \quad 1 - \frac{1}{r} \quad ^{n} = \frac{n}{r} \exp \quad -\frac{n-1}{2r} \qquad 1 + O \quad \frac{n^2}{r^2}$$

and therefore:

$$1 - \quad 1 - \frac{1}{r} \quad ^{n \quad d} \qquad \frac{n}{r} \quad ^{d} \exp \quad -\frac{d(n-1)}{2r}$$

which immediately gives equation (3.5). ■

Having found a division for S, we are not yet nished, because the procedure has to be recursively applied to the subsets obtained from S (see procedure *Perfect-Division*). Since n=2 is the approximate size of these two subsets, the probability of nding a division for each of them is:

$$\frac{n}{2}; r; d \qquad 1 - \quad \frac{n}{2r} \quad ^{d} \exp \quad -\frac{d(n-2)}{4r} \qquad ;$$

in fact, except that in very particular cases, the number of possible positions, at which division can take place, is not diminished. The joint probability that both subsets can be divided is $(n{=}2; r; d)^2$ and the probability of obtaining a complete division of $S;$ so that modi ed binary searching is applicable, is:

$$(n; r; d) = \quad (n; r; d) \quad (n{=}2; r; d)^2 \quad (n{=}4; r; d)^4 \qquad ;$$

the product extended up to $(n{=}2^k; r; d)^{2^k}$ such that $n{=}2^k$ $\quad 3:$ In fact, as observed in Theorem 1, a division for tables with 1 or 2 elements is always possible (i.e., $\quad = 1$).

The probability $(n; r; d)$ is the quantity we are mainly interested in. Obviously, $(n; r; d) < \quad (n; r; d)$ but, fortunately, we can show that these probabilities are almost equal, at least in most important cases.

Theorem 7 *If* $(n; r; d)$ *is su ciently near to* 1; *then* $(n; r; d) \qquad (n; r; d):$

Proof: First of all, let us develop $(n; r; d)$:

$$(n; r; d)$$

$$1 - \quad \frac{n}{r} \quad ^{d} \exp \quad -\frac{d(n-1)}{2r} \qquad 1 - 2 \quad \frac{n}{2r} \quad ^{d} \exp \quad -\frac{d(n-2)}{4r}$$

$$1 - \quad \frac{n}{r} \quad ^{d} \exp \quad -\frac{d(n-1)}{2r} \quad - 2 \quad \frac{n}{2r} \quad ^{d} \exp \quad -\frac{d(n-2)}{4r} \quad -$$

$$-4 \quad \frac{n}{4r}^{d} \exp \quad -\frac{d(n-4)}{8r} \quad -$$

What we now wish to show is that the rst term (after 1) dominates all the others, and therefore $(n; r; d) \quad (n; r; d):$ Let us consider the ratio between the rst and the second term after the 1 :

$$\frac{n}{r}^{d} \exp \quad -\frac{d(n-1)}{2r} \quad \frac{1}{2} \quad \frac{2r}{n}^{d} \exp \quad \frac{d(n-2)}{4r} \quad =$$

$$= 2^{d-1} \exp \quad \frac{dn - 2d - 2dn + 2d}{4r} \quad = 2^{d-1} \exp \quad -\frac{dn}{4r} \quad :$$

This quantity shows that the rst term is much larger than the second, as claimed. ■

4 Experimental Results

In order to verify the e ectiveness of our modi ed binary searching method, we devised some experiments comparing traditional and modi ed binary search programs. We used Pascal as a programming language, well aware that this might not be the best choice. However, our aim was also to show how better is our method when realised without any particular trick in a high level language. When the table is large, we need two or more characters for the selector s; our implementation obviously performs one-character-at-a-time comparisons; since the characters to be compared against s are consecutive, a machine language realization would instead perform a multi-byte loading and comparing, further reducing execution time. Presently, we are developing a C version of our programs to compare them to the most sophisticated realisations of traditional binary searching and to other, more recent approaches to string retrieval, such as the method of Bentley and Sedgewick [1].

We used the following program for traditional binary searching:

```
function BS (str : string) : integer;
var a, b, k : integer; found : boolean;
begin
    a := 1; b := n; found := false;
    while a<=b do begin
        k:=(a+b) div 2;
        if str = T[k]
            then begin a := b+1; found := true end
        else if str < T[k] then b:=k-1 else a:=k+1
    end;
    if found then BS := k else BS := 0
end;
```

For the modi ed binary searching, the program is:

```
function MBS1 (str : string): integer;
var a, b, j, k : integer; found : boolean;
begin
    a := 1; b := n; found := false;
    while a <= b do begin
        k := (a+b) div 2; j := V[k];
        if str[j] = T[k,j]
            then begin a := b+1; found := true end
            else if str[j] < T[k,j]
                then b := k - 1
                else a := k + 1 end;
    if found and (str = T[k])
        then MBS1 := k else MBS1 := 0
end;
```

This program is to be used when the selector length is 1; for a selector length of 2 we simply perform two cascade if's, and three for a selector length of 3.

We performed 10 blocks of 20,000 searches of all the strings in a table, randomly chosen in a dictionary of 1,524 English words. For small tables (selector length equal to 1) we obtained the average times in the rst part of Table 4 (the time unit is inessential; only relative times are of importance). For larger tables with selector length equal to 2 or 3 we obtained the times in parts two and three of Table 4.

Table 4. Times for selector of length 1, 2 and 3

n	MBS	BS	gain (%)
10	29.7	56.2	47.2
11	33.9	63.6	46.7
12	36.3	70.9	48.8
13	38.9	80.4	51.6
14	44.5	86.3	48.4
15	47.1	93.8	49.8
16	52.0	103.1	49.6
17	54.4	113.7	52.2
18	58.3	123.1	52.6
20	65.9	139.6	52.8
22	73.3	159.5	54.0
24	81.3	178.8	54.5
26	87.3	197.8	55.9
28	94.4	215.9	56.3
30	101.6	232.8	56.4
32	110.0	254.8	56.8

n	MBS	BS	gain (%)
10	33.0	54.8	39.8
20	69.9	141.1	50.5
30	108.3	234.4	53.8
40	150.5	345.4	56.4
60	235.1	568.5	58.6
80	326.8	825.0	60.4
100	421.8	1084.2	61.1
120	519.6	1340.6	61.2
140	619.5	1625.3	61.9
160	713.4	1919.7	62.8
180	806.5	2214.3	63.6
200	913.3	2510.2	63.6
225	1032.6	2877.0	64.1
250	1146.3	3248.7	64.7
275	1263.1	3648.8	65.4
300	1408.8	4060.6	65.3

n	MBS	BS	gain (%)
50	100.0	228.4	56.2
100	222.8	541.8	58.9
150	346.0	887.6	61.0
200	472.4	1256.6	62.4
250	607.6	1625.4	62.6
300	751.2	2035.4	63.1
350	883.4	2452.8	64.0
400	1032.8	2864.8	63.9
450	1151.4	3283.4	64.9
500	1310.4	3702.0	64.6
550	1447.8	4147.0	65.1
600	1592.8	4610.2	65.5
700	1874.0	5534.2	66.1
800	2202.6	6461.4	65.9
900	2491.4	7377.8	66.2
1000	2760.6	8303.8	66.8

5 Conclusions

We have considered a variant of binary searching, which avoids most of the housekeeping instructions related to string comparisons. This requires a suitable pre-processing phase for the table to be searched, and therefore only applies to the static case. What we have shown is:

1. The variant is considerably faster than traditional binary searching; we give empirical evidence of this fact, by comparing actual programs performing both kinds of binary searching.
2. The pre-processing phase is fast, because it runs in time $O(n(\log_2 n)^2)$, and produces with a very high probability the optimal arrangement of the table elements. In any case, an almost optimal arrangement can always be found.

In our opinion, an important aspect of our method is that it can be e ciently realised in a high level language; as is well-known, this is not always possible for other kinds of fast retrieval methods for static tables, such as perfect hashing. This makes our modi ed binary searching procedure attractive for actual implemenattion in real systems.

References

1. J.L. Bentley, R. Sedgewick: Fast Algorithms for Sorting and Searching Strings. Proceedings of 8th Annual ACM-SIAM Symp. On Discrete Algorithms. (1997) 360{369
2. G. H. Gonnet, R. Baeza-Yates: Algorithms and Data Structures, 2nd edition. Addison Wesley (1991)
3. D. E. Knuth: The Art of Computer Programming. Vol. 1-3. Addison-Wesley (1973)
4. R. Sedgewick, P. Flajolet: An Introduction to the Analysis of Algorithms. Addison-Wesley (1996)

An Efficient Algorithm for the Approximate Median Selection Problem

Sebastiano Battiato[1], Domenico Cantone[1], Dario Catalano[1], Gianluca Cincotti[1], and Micha Hofri[2]

[1] Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I–95125 Catania, Italy
fbattiato,cantone,catalano,cincottig@cs.unict.it
[2] Department of Computer Science, WPI
100 Institute Road, Worcester MA 01609-2280, USA
hofri@cs.wpi.edu

Abstract. We present an efficient algorithm for the approximate median selection problem. The algorithm works *in-place*; it is fast and easy to implement. For a large array it returns, with high probability, a very close estimate of the true median. The running time is linear in the length n of the input. The algorithm performs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average. We present analytical results of the performance of the algorithm, as well as experimental illustrations of its precision.

Keywords: Approximation algorithms, in-place algorithms, median selection, analysis of algorithms.

1. Introduction

In this paper we present an efficient algorithm for the *in-place* approximate median selection problem. There are several works in the literature treating the exact median selection problem (cf. [BFP*73], [DZ99], [FJ80], [FR75], [Hoa61], [HPM97]). Various in-place median finding algorithms have been proposed. Traditionally, the "comparison cost model" is adopted, where the only factor considered in the algorithm cost is the number of key-comparisons. The best upper bound on this cost found so far is nearly $3n$ comparisons in the worst case (cf. [DZ99]). However, this bound and the nearly-as-efficient ones share the unfortunate feature that their nice asymptotic behaviour is "paid for" by extremely involved implementations.

The algorithm described here approximates the median with high precision and lends itself to an immediate implementation. Moreover, it is quite fast: we show that it needs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average and fewer than $\frac{3}{2}n$ comparisons and $\frac{1}{2}n$ exchanges in the *worst-case*. In addition to its sequential efficiency, it is very easily parallelizable due to the low level of data contention it creates.

The usefulness of such an algorithm is evident for all applications where it is sufficient to find an approximate median, for example in some heapsort variants (cf. [Ros97], [Kat96]), or for median-filtering in image representation. In addition, the analysis of its precision is of independent interest.

We note that the procedure *pseudomed* in [BB96, x7.5] is similar to performing just one iteration of the algorithm we present (using quintets instead of triplets), as an aid in deriving a (precise) selection procedure.

In a companion paper we show how to extend our method to approximate general k-selection.

All the works mentioned above—as well as ours—assume the selection is from values stored in an array in main memory. The algorithm has an additional property which, as we found recently, has led to its being discovered before, albeit for solving a rather different problem. As is apparent on reading the algorithms presented in Section 2, it is possible to perform the selection in this way "on the fly," without keeping all the values in storage. At the extreme case, if the values are read in one-by-one, the algorithm only uses $4\log_3 n$ positions (including $b\log_3 nc$ loop variables). This way of performing the algorithm is described in [RB90], in the context of estimating the median of an unknown distribution. The authors show there that the value thus selected is a consistent estimator of the desired parameter. They need pay no attention (and indeed do not) to the relation between the value the algorithm selects and the actual sample median. The last relation is the center point of interest for us. Curiously, Weide notes in [Wei78] that this approach provides an approximation of the sample median, though no analysis of the bias is provided. See [HM95] for further discussion of the method of Rousseeuw and Bassett, and numerous other treatments of the statistical problem of low-storage quantile (and in particular median) estimation.

In Section 2 we present the algorithm. Section 3 provides analysis of its run-time. In Section 4, to show the soundness of the method, we present a probabilistic analysis of the precision of its median selection. Since it is hard to glean the shape of the distribution function from the analytical results, we provide computational evidence to support the conjecture that the distribution is asymptotically normal. In Section 5 we illustrate the algorithm with a few experimental results, which also demonstrate its robustness. Section 6 concludes the paper with suggested directions for additional research.

An extended version of this paper is available by anonymous `ftp` from `ftp://ftp.cs.wpi.edu/pub/techreports/99-26.ps.gz`.

2. The Algorithm

It is convenient to distinguish two cases:

2.1 The Size of the Input Is a Power of 3: $n = 3^r$

Let $n = 3^r$ be the size of the input array, with an integer r. The algorithm proceeds in r stages. At each stage it divides the input into subsets of three elements, and calculates the median of each such triplet. The "local medians" survive to the next stage. The algorithm continues recursively, using the local results to compute the approximate median of the initial set. To incur the fewest number of exchanges we do not move the chosen elements from their original triplets. This adds some index manipulation operations, but is typically advantageous. (While the order of the elements is disturbed, the contents of the array is unchanged).

Approximate Median Algorithm (1)

```
Triplet_Adjust($A$, $i$, Step)
    Let $j = i + Step$ and $k = i + 2 \cdot Step$; this procedure moves the median of a triplet of
    terms at locations $i, j, k$ to the middle position.
    if ($A[i] < A[j]$)
      then
            if ($A[k] < A[i]$) then Swap($A[i], A[j]$);
            else if ($A[k] < A[j]$) then Swap($A[j], A[k]$);
      else
            if ($A[i] < A[k]$) then Swap($A[i], A[j]$);
            else if ($A[k] > A[j]$) then Swap($A[j], A[k]$);

Approximate_Median($A$, $r$)
    This procedure returns the approximate median of the array $A[0, \ldots, 3^r - 1]$.
    Step=1;   Size=$3^r$;
    repeat $r$ times
            $i$=(Step−1)/2;
            while $i$ < Size do
                  Triplet_Adjust($A$, $i$, Step);
                  $i$=$i$+(3 Step);
            end while;
            Step = 3 Step;
     end repeat;
     return $A[(Size - 1)/2]$;
```

Fig. 1. Pseudo-code for the approximate median algorithm, $n = 3^r, r \in \mathbb{N}$.

In Fig. 1 we show pseudo-code for the algorithm. The procedure *Triplet_Adjust* finds the median of triplets with elements that are indexed by two parameters: one, i, denotes the position of the leftmost element of triplet in the array. The second parameter, *Step*, is the relative distance between the triplet elements. This approach requires that when the procedure returns, the median of the triplet is in the middle position, possibly following an exchange. The *Approximate_Median* algorithm simply consists of successive calls to the procedure.

2.2 The Extension of the Algorithm to Arbitrary-Size Input

The method described in the previous subsection can be generalized to array sizes which are not powers of 3. The basic idea is similar. Let n be the input size at the current stage, where

$$n = 3 \cdot t + k; \qquad k \in \{0, 1, 2\}.$$

We divide the input into $(t-1)$ triplets and a $(3+k)$-tuple. The $(t-1)$ triplets are processed by the same *Triplet_Adjust* procedure described above. The last tuple is sorted (using an adaptation of selection-sort) and the median is extracted. The algorithm continues iteratively using the results of each stage as input for a new one. This is done until the number of local medians falls below a small fixed threshold. We then sort the remaining elements and obtain the median. To symmetrize the algorithm, the array is scanned from left to right during the first iteration, then from right to left on the second one, and so on, changing the scanning sense at each iteration. This should reduce the perturbation due to the different way in which the medians from the $(3 + k)$-tuples are selected and improve the precision of the algorithm. Note that we chose to select the second element out of four as the median (2 out of 1..4). We show pseudo-code for the general case algorithm in Fig. 2.

```
                    Approximate Median Algorithm (2)

Selection_Sort ($A$, Left, Size, Step)
    This procedure sorts Size elements of the array $A$ located at positions Left, Left + Step,
    Left + 2  Step; : : : ; Left + (Size − 1)  Step.

    for ($i$ = Left ; $i$ < Left + (Size − 1)  Step; $i$ = $i$ + Step)
            $Min = i$;
            for ($j$ = $i$ + Step; $j$ < Left + Size  Step; $j$ = $j$ + Step)
                    if ($A[j] < A[min]$) then $min = j$;
            end for;
            Swap($A[i]; A[min]$);
    end for;

Approximate_Median_AnyN (A, Size)
      This procedure returns the approximate median of the array $A$[0; :::; Size − 1].
      LeftToRight = $False$;   Left = 0;   Step = 1;
      while (Size > Threshold) do
            LeftToRight = Not (LeftToRight);
            $Rem$ = (Size mod 3);
            if (LeftToRight) then $i$ = Left;
                             else $i$ = Left + (3 + $Rem$)  Step;
            repeat (Size=3 − 1) times
                    Triplet_Adjust ($A$, $i$, Step);
                    $i = i$ + 3  Step;
            end repeat;
            if (LeftToRight) then Left = Left + Step;
                             else $i$ = Left;
                                  Left = Left + (1 + $Rem$)  Step;
            Selection_Sort ($A$, $i$, 3 + $Rem$, Step);
            if ($Rem$ = 2) then
                        if (LeftToRight) then Swap($A[i$ + Step], $A[i$ + 2  Step])
                                         else Swap($A[i$ + 2  Step], $A[i$ + 3  Step]);
            Step = 3  Step; Size = Size=3;
      end while;
      Selection_Sort ($A$, Left, Size, Step);
      return $A$[Left + Step  b(Size − 1)=2c];
```

Fig. 2. Pseudo-code for the approximate median algorithm, any n 2 ℕ.

Note: The reason we use a terminating tuple of size 4 or 5, rather than 1 or 2, is to keep the equal spacing of elements surviving one stage to the next.

The procedure *Selection_Sort* takes as input four parameters: the array A, its size and two integers, *Left* and *Step*. At each iteration *Left* points to the leftmost element of the array which is in the current input, and *Step* is the distance between any two successive elements in this input.

There are several alternatives to this approach for arbitrary-sized input. An attractive one is described in [RB90], but it requires *additional storage* of approximately $4 \log_3 n$ memory locations.

3. Run-Time Analysis: Counting Moves and Comparisons

Most of the work of the algorithm is spent in *Triplet_Adjust*, comparing values and exchanging elements within triplets to locate their medians. We compute now the number of comparisons and exchanges performed by the algorithm *Approximate_Median*.

Like all reasonable median-searching algorithms, ours has running-time which is linear in the array size. It is distinguished by the simplicity of its code, and hence it is extremely efficient. We consider first the algorithm described in Fig. 1.

Let $n = 3^r$, $r\ 2\ \mathbb{N}$, be the size of a randomly-ordered input array. We have the following elementary results:

Theorem 1. *Given an input of size n, the algorithm Approximate_Median performs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average.* ⊓⊔

Proof: Consider first the *Triplet_Adjust* subroutine. In the following table we show the number of comparisons and exchanges, C_3 and E_3, for each permutation of three distinct elements:

A[i]	A[i+Step]	A[i+2*Step]	Comparisons	Exchanges
1	2	3	3	0
1	3	2	3	1
2	1	3	2	1
2	3	1	2	1
3	1	2	3	1
3	2	1	3	0

Clearly, assuming all orders equally likely, we find $\Pr(C_3 = 2) = 1 - \Pr(C_3 = 3) = 1=3$, and similarly $\Pr(E_3 = 0) = 1 - \Pr(E_3 = 1) = 1=3$, with expected values $E[E_3] = 2=3$ and $E[C_3] = 8=3$.

To find the work of the entire algorithm with an input of size n, we multiply the above by $T(n)$, the number of times the subroutine *Triplet_Adjust* is executed. This number is deterministic. We have $T(1) = 0$ and $T(n) = \frac{n}{3} + T(\frac{n}{3})$, for $n > 1$; for n which is a power of 3 the solution is immediate: $T(n) = \frac{1}{2}(n - 1)$.

Let $_n$ be the number of possible inputs of size n and let $_n$ be the total number of comparisons performed by the algorithm on all inputs of size n.

The average number of comparisons for all inputs of size n is:

$$C(n) = \frac{\ _n}{\ _n} = \frac{16\quad T(n)\quad n!=3!}{n!} = \frac{4}{3}(n-1):$$

To get $\ _n$ we count all the triplets considered for all the $\ _n$ inputs, i.e. $n!\quad T(n)$; for each triplet we consider the cost over its 3! permutations (the factor 16 is the cost for the 3! permutations of each triplet[1]).

The average number of exchanges can be shown analogously, since two out of three permutations require an exchange.

By picking the "worst" rows in the table given in the proof of Theorem 1, it is straightforward to verify also the following:

Theorem 2. *Given an input of size $n = 3^r$, the algorithm Approximate_Median performs fewer than $\frac{3}{2}n$ comparisons and $\frac{1}{2}n$ exchanges in the worst-case.* ⊓⊔

For an input size which is a power of 3, the algorithm of Fig. 2 performs nearly the same operations as the simpler algorithm – in particular, it makes the same key-comparisons, and selects the same elements. For $\log_3 n \not\in \mathbb{N}$, their performance only differs on one tuple per iteration, hence the leading term (and its coefficient) in the asymptotic expression for the costs is the same as in the simpler case.

The non-local algorithm described in [RB90] performs exactly the same number of comparisons as above but always moves the selected median. The overall run-time cost is very similar to our procedure.

4. Probabilistic Performance Analysis

4.1 Range of Selection

It is obvious that not all the input array elements can be selected by the algorithm — *e.g.*, the smallest one is discarded in the first stage. Let us consider first the algorithm of Fig. 1 (*i.e.* when n is a power of 3). Let $v(n)$ be the number of elements from the lower end (alternatively – upper end, since the *Approximate_Median* algorithm has bilateral symmetry) of the input which *cannot be selected* out of an array of n elements. It is easy to verify (by observing the tree built with the algorithm) that $v(n)$ obeys the following recursive inequality:

$$v(3) = 1 \quad ; \qquad v(n) \qquad 2v(n=3) + 1: \tag{1}$$

Moreover, when $n = 3^r$, the equality holds. The solution of the recursive *equation*,

$$fv(3) = 1; \quad v(n) = 2v(n=3) + 1g$$

[1] Alternatively, we can use the following recurrence: $C(1) = 0$ and $C(n) = \frac{n}{3}\quad c + C(\frac{n}{3})$, for $n > 1$, where $c = \frac{16}{6}$ is the average number of comparisons of *Triplet_Adjust* (because all the 3! permutations are equally likely).

is the following function

$$v(n) = n^{\log_3 2} - 1 = 2^{\log_3 n} - 1.$$

Let x be the output of the algorithm over an input of n elements. From the definition of $v(n)$ it follows that

$$v(n) < rank(x) < n - v(n) + 1. \qquad (2)$$

The second algorithm behaves very similarly (they perform the same operations when $n = 3^r$) and the range function $v(n)$ obeys the same recurrence.

Unfortunately not many entries get thus excluded. The range of possible selection, as a fraction of the entire set of numbers, increases promptly with n. This is simplest to illustrate with n that is a power of 3. Since $v(n)$ can be written as $2^{\log_3 n} - 1$, the ratio $v(n)/n$ is approximately $(2/3)^{\log_3 n}$. Thus, for $n = 3^3 = 27$, where the smallest (and largest) 7 numbers cannot be selected, 52% of the range is excluded; the comparable restriction is 17.3% for $n = 3^6 = 729$ and only 1.73% for $n = 3^{12} = 531441$.

The true state of affairs, as we now proceed to show, is much better: while the possible range of choice is wide, the algorithm zeroes in, with overwhelming probability, on a very small neighborhood of the true median.

4.2 Probabilities of Selection

The most telling characteristic of the algorithms is their precision, which can be expressed via the probability function

$$P(z) = \Pr[zn < rank(x) < (1 - z)n + 1], \qquad (3)$$

for $0 \le z \le 1/2$, which describes the closeness of the selected value to the true median.

The purpose of the following analysis is to show the behavior of this distribution. We consider n which is a power of 3.

Definition 1. *Let $q^{(r)}_{a,b}$ be the number of permutations, out of the $n! = 3^r!$ possible ones, in which the entry which is the a^{th} smallest in the set is: (1) selected, and (2) becomes the b^{th} smallest in the next set, which has $\frac{n}{3} = 3^{r-1}$ entries.*

It turns out that this quite narrow look at the selection process is all we need to characterize it completely.

It can be shown that

$$q^{(r)}_{a,b} = 2n(a-1)!(n-a)! \binom{\frac{n}{3}-1}{b-1} 3^{a-b-1} \sum_i \binom{b-1}{i} \binom{\frac{n}{3}-b}{a-2b-i} \frac{1}{9^i} \qquad (4)$$

(for details, see [BCC*99]).

It can also be seen that $q^{(r)}_{a,b}$ is nonzero for $0 \le a - 2b \le \frac{n}{3} - 1$ only. The sum is expressible as a Jacobi polynomial, $\left(\frac{8}{9}\right)^{a-2b} P^{(u,v)}_{a-2b}\left(\frac{5}{4}\right)$, where $u = 3b - a - 1, v = \frac{n}{3} + b - a$, and a simpler closed form is unlikely.

Let $p^{(r)}_{a;b}$ be the probability that item a gets to be the b^{th} smallest among those selected for the next stage. Since the $n! = 3^r!$ permutations are assumed to be equally likely, we have $p^{(r)}_{a;b} = q^{(r)}_{a;b}/n!$:

$$
\begin{aligned}
p^{(r)}_{a;b} &= \frac{2}{3}\frac{3^{-b}\binom{\frac{n}{3}-1}{b-1}}{3^{-a}\binom{n-1}{a-1}} \sum_i \binom{b-1}{i}\binom{\frac{n}{3}-b}{a-2b-i}\frac{1}{9^i} \\
&= \frac{2}{3}\frac{3^{-b}\binom{\frac{n}{3}-1}{b-1}}{3^{-a}\binom{n-1}{a-1}} [z^{a-2b}](1+\frac{z}{9})^{b-1}(1+z)^{\frac{n}{3}-b}. \qquad (5)
\end{aligned}
$$

This allows us to calculate the center of our interest: The probability $P^{(r)}_a$, of starting with an array of the first $n = 3^r$ natural numbers, and having the element a ultimately chosen as approximate median. It is given by

$$
P^{(r)}_a = \sum_{b_r} p^{(r)}_{a;b_r} P^{(r-1)}_{b_r} = \sum_{b_r;b_{r-1};\cdots;b_3} p^{(r)}_{a;b_r} p^{(r-1)}_{b_r;b_{r-1}} \cdots p^{(2)}_{b_3;2}, \qquad (6)
$$

where $2^{j-1} \le b_j \le 3^{j-1} - 2^{j-1} + 1$, for $j = 3, 4, \ldots, r$.

Some telescopic cancellation occurs when the explicit expression for $p^{(r)}_{a;b}$ is used here, and we get

$$
P^{(r)}_a = \left(\frac{2}{3}\right)^r \frac{3^{a-1}}{\binom{n-1}{a-1}} \sum_{b_r;b_{r-1};\cdots;b_3} \prod_{j=2} \sum_{i_j \ge 0} \binom{b_j-1}{i_j}\binom{3^{j-1}-b_j}{b_{j+1}-2b_j-i_j}\frac{1}{9^{i_j}}. \qquad (7)
$$

As above, each b_j takes values in the range $[2^{j-1} \ldots 3^{j-1} - 2^{j-1} + 1]$, $b_2 = 2$ and $b_{r+1} \equiv a$ (we could let all b_j take all positive values, and the binomial coefficients would produce nonzero values for the required range only). The probability $P^{(r)}_a$ is nonzero for $v(n) < a < n - v(n) + 1$ only.

This distribution has so far resisted our attempts to provide an analytic characterization of its behavior. In particular, while the examples below suggest very strongly that as the input array grows, it approaches the normal distribution, this is not easy to show analytically. (See Section 4.4 of [BCC*99] for an approach to gain further information about the large-sample distribution.)

4.3 Examples

We computed $P^{(r)}_a$ for several values of r. Results for a small array ($r = 3, n = 27$) are shown in Fig.3. By comparison, with a larger array ($r = 5, n = 243$, Fig. 4) we notice the relative concentration of the likely range of selection around the true median. In terms of these probabilities the relation (3) is:

$$
\sum_{\lfloor zn \rfloor < a < \lceil (1-z)n \rceil + 1} P^{(r)}_a \qquad (8)
$$

where $0 \le z < \frac{1}{2}$.

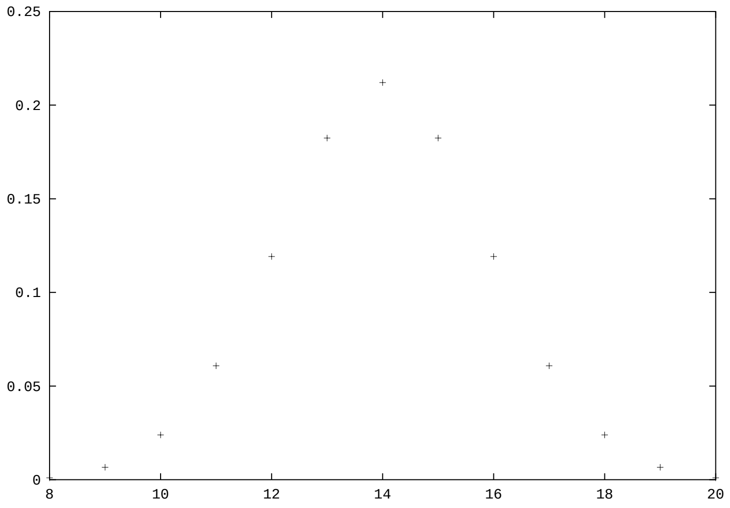

Fig. 3. Plot of the median probability distribution for n=27.

We chose to present the effectiveness of the algorithm by computing directly from equation (7) the statistics of the absolute value of the bias of the returned approximate median, $D_n \quad |X_n - M_d(n)|$ (where $M_d(n)$ is the true median, $(n+1)/2$). We compute its mean (Avg.) and standard deviation, denoted by $_d$.

A measure of the improvement of the selection effectiveness with increasing (initial) array size n is seen from the variance ratio $_d/M_d(n)$. This ratio may be viewed as a measure of the expected relative error of the approximate median selection algorithm.

Numerical computations produced the numbers in Table 1; note the trend in the rightmost column. (This trend is the basis for the approach examined in Section 4.4 of [BCC*99].)

n	$r = \log_3 n$	Avg.	$_d$	$_d/\sqrt{n}$
9	2	0.428571	0.494872	0.164957
27	3	1.475971	1.184262	0.227911
81	4	3.617240	2.782263	0.309140
243	5	8.096189	6.194667	0.397388
729	6	17.377167	13.282273	0.491958
2187	7	36.427027	27.826992	0.595034

Table 1. Statistics of the median selection as function of array size.

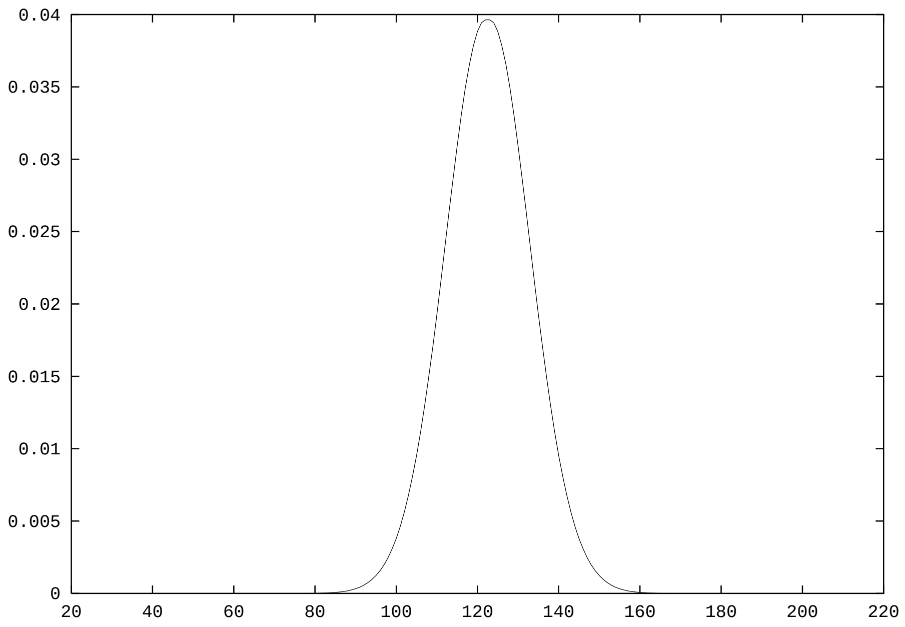

Fig. 4. Plot of the median probability distribution for n=243.

5. Experimental Results

In this section we present empirical results, demonstrating the effectiveness of the algorithms – also for the cases which our analysis does not handle directly. Our implementation is in standard C (GNU C compiler v2.7). All the experiments were carried out on a PC Pentium II 350Mhz with the Linux (Red Hat distribution) operating system. The lists were permuted using the pseudo-random number generator suggested by Park and Miller in 1988 and updated in 1993 [PM88]. The algorithm was run on random arrays of sizes that were powers of 3, $n = 3^r$, with $r \in \{3, 4, \dots, 11\}$, and others. The entry keys were always the integers $1, \dots, n$.

The following tables present results of such runs. They report the statistics of D_n, the absolute value of the bias of the approximate median. For each returned result we compute its distance from the correct media $\frac{n+1}{2}$. The units we use in the tables are "normalized" values of D_n, denoted by $d_\%$; these are percentiles of the possible range of error of the algorithm: $d_\% = 100 \frac{D_n}{(n-1)/2}$. The extremes are $d_\% = 0$ when the true median is returned – and it would have been 100 if it were possible to return the smallest (or largest) elements. (But relation (2) shows that $d_\%$ can get arbitrarily close to 100 as n increases, but never quite reach it). Moreover, the probability distributions of the last section suggest, as illustrated in Figure 4, that such deviations are extremely unlikely. Table 2 shows selected results, using a threshold value of 8. All experiments used a sample size of 5000, throughout.

n	Avg.		Avg. + 2	Rng(95%)	(Min-Max)
50	10.27	7.89	26.05	24.49	0.00–44.90
100	8.45	6.63	21.70	20.20	0.00–48.48
3^5	6.74	5.13	17.00	16.53	0.00–38.02
500	5.80	4.41	14.63	14.03	0.00–29.06
3^6	4.83	3.71	12.26	12.09	0.00–24.73
1000	4.70	3.66	12.02	11.61	0.00–23.62
3^7	3.32	2.54	8.41	8.05	0.00–16.83
5000	2.71	2.10	6.91	6.72	0.00–17.20
3^8	2.31	1.75	5.81	5.67	0.00–11.95
10000	2.53	1.86	6.24	6.04	0.00–11.38
3^9	1.58	1.18	3.94	3.86	0.00–6.78

Table 2. Simulation results for $d_{\%}$, fractional bias of approximate median. Sample size = 5000.

The columns are: n – array size; Avg. – the average of $d_{\%}$ over the sample;   – the sample standard-error of $d_{\%}$; Rng. (95%) – the size of an interval symmetric around Avg. that contains 95% of the returned values; the last column gives the extremes of $d_{\%}$ that were observed. In the rows that correspond to those of Table 1, the agreement of the Avg. and   columns is excellent (the relative differences are under 0:5%). Columns 4 and 5 suggest the closeness of the median distribution to the Gaussian, as shown above.

All the entries show the critical dependence of the quality of the selection on the size of the initial array. In the following table we report the data for different values of n with sample size of 5000, varying the threshold.

	t=8			t=26			t=80		
n	Avg.		Rng. (95%)	Avg.		Rng. (95%)	Avg.		Rng. (95%)
100	8.63	6.71	22.22	6.80	5.31	16.16	4.64	3.61	12.12
500	5.80	4.41	14.43	4.40	3.30	10.82	3.22	2.40	7.62
1000	4.79	3.67	12.01	3.80	2.88	9.41	2.98	2.27	7.41
10000	2.54	1.87	6.05	1.67	1.28	4.14	1.40	1.06	3.44

Table 3. Quality of selection as a function of threshold value.

As expected, increasing the threshold—the maximal size of an array which is sorted, to produce the exact median of the remaining terms—provides better selection, at the cost of rather larger processing time. For large n, threshold values beyond 30 provide marginal additional benefit. Settling on a correct trade-off here is a critical step in tuning the algorithm for any specific application.

Finally we tested for the relative merit of using quintets rather than triplets when selecting for the median. In this case $n = 1000$, Threshold=8, and sample size=5000.

	Avg.		Avg. + 2	Rng(95%)	(Min-Max)
Triplets	4.70	3.66	12.02	11.61	0.00–23.62
Quintets	3.60	2.74	9.08	9.01	0.00–16.42

Table 4. Comparing selection via triplets and quintets.

6. Conclusion

We have presented an approximate median finding algorithm, and an analysis of its characteristics. Both can be extended. In particular, the algorithm can be adapted to select an approximate k^{th}-element – for any k 2 [1; n]. The analysis of Section 4 can be extended to show how to compute with the exact probabilities, as given in equation (7). Also, the limiting distribution of the bias D with respect to the true median – while we know it is extremely close to a gaussian distribution, we have no efficient representation for it yet.

Acknowledgments

The derivations of Section 4.4 in [BCC*99] are largely due to Svante Janson. The references [HM95], [RB90], and [Wei78] were provided by Reza Modaress. We are grateful for their contributions.

The authors wish also to thank an anonymous referee for some useful comments.

References

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer algorithms*. Addison Wesley, Reading, MA 1974.

[BB96] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, Englewood Cliffs, NJ 1996.

[BCC*99] S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri. *An Efficient Algorithm for the Approximate Median Selection Problem*. Technical Report WPI-CS-TR-99-26, Worcester Polytechnic Institute, October 1999. Available from `ftp://ftp.cs.wpi.edu/pub/techreports/99-26.ps.gz`.

[BFP*73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and Systems Sciences*, 7(4):448–461, 1973.

[CS87] S. Carlsson, M. Sundstrom. Linear-time In-place Selection in Less than 3n Comparisons - Division of Computer Science, Lulea University of Technology, S-971 87 LULEA, Sweden.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[CM89] W. Cunto and J.I. Munro. Average case selection. *Journal of the ACM*, 36(2):270–279, 1989.

[Dra67] Alvin W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, 1967.

[DZ99] D. Dor, and U. Zwick. Selecting the Median. *SIAM Jour. Comp.*, 28(5):1722–1758, 1999.

[FJ80] G.N. Frederickson, D.B. Johnson. Generalized Selection and Ranking. *Proceedings STOC-SIGACT*, Los Angeles CA, 12:420–428, 1980.

[Fre90] G.N. Frederickson. The Information Theory Bound is Tight for selection in a heap. *Proceedings STOC-SIGACT*, Baltimore MD, 22:26–33, 1990.

[FR75] R.W. Floyd, R.L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[Hoa61] C.A.R. Hoare. Algorithm 63(partition) and algorithm 65(find). *Communications of the ACM*, 4(7):321–322, 1961.

[Hof95] M. Hofri, *Analysis of Algorithms: Computational Methods & Mathematical Tools*, Oxford University Press, New York (1995).

[HM95] C. Hurley and Reza Modarres: Low-Storage quantile estimation, *Computational Statistics*, 10:311–325, 1995.

[HPM97] P. Kirschenhofer, C. Martinez, and H. Prodinger. Analysis of Hoare's Find algorithm with median-of-three partition. *Random Structures and Algorithms*, 10:143–156, 1997.

[Kat96] J. Katajainen. *The Ultimate Heapsort*, DIKU Report 96/42, Department of Computer Science, Univ. of Copenhagen, 1996.

[Knu98] D.E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, 2nd Ed. 1999.

[LW88] T.W. Lai, and D. Wood. Implicit Selection. *Scandinavian Workshop on Algorithm Theory (SWAT88)*:18–23, LNCS 38 Springer-Verlag, 1988.

[Meh84] K. Mehlhorn. *Sorting and Searching, Data Structures and Algorithms*, volume 1. Springer-Verlag, 1984.

[Noz73] A. Nozaky. Two Entropies of a Generalized Sorting Problems. *Journal of Computer and Systems Sciences*, 7(5):615–621, 1973.

[PM88] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988. Updated in *Communications of the ACM*, 36(7):108–110, 1993.

[Ros97] L. Rosaz.*Improving Katajainen's Ultimate Heapsort*, Technical Report N.1115, Laboratoire de Recherche en Informatique, Université de Paris Sud, Orsay, 1997.

[RB90] P.J. Rousseeuw and G.W. Bassett: The remedian: A robust averaging method for large data sets. *Jour. Amer. Statist. Assoc*, 409:97–104, 1990.

[SJ99] Savante Janson – Private communication, June 1999.

[SPP76] A. Schonhage, M. Paterson, and N. Pippenger. Finding the median. *Journal of Computer and Systems Sciences*, 13:184–199, 1976.

[Wei78] B. W. Weide. Space efficient on-line selection algorithm. *Proceedings of the 11th symposium of Computer Science and Statistics, on the interface*, 308–311. (1978).

Extending the Implicit Computational Complexity Approach to the Sub-elementary Time-Space Classes

Emanuele Covino, Giovanni Pani, and Salvatore Caporaso

Dipartimento di Informatica dell'Universita di Bari
(covino|pani|caporaso)@di.uniba.it

Abstract. A resource-free characterization of some complexity classes is given by means of the *predicative recursion* and *constructive diagonalization* schemes, and of restrictions to substitution. Among other classes, we predicatively harmonize in the same hierarchy ptimef, the class E of the elementary functions, and classes dtimespacef$(n^p; n^q)$.
Keywords: time-space classes, implicit computational complexity, elementary functions.

1 Introduction

Position of the problem. The standard de nition of a complexity class involves the de nition of a bound imposed on time and/or space resources used by a Turing Machine during its computation; a di erent approach characterizes complexity classes by means of limited recursive operators. The rst characterization of this type of a small complexity class was given by Cobham [8], who showed that the polytime functions are exactly those functions generated by *bounded recursion on notation*; however, a *smash* initial function 2^{jxjjyj} was used to provide space enough.
Leivant [12] and Bellantoni & Cook [2] gave the characterizations of ptimef; several other complexity classes have been characterized by means of unlimited operators (see [13,5] for pspacef, [4] for ptime, pspace (languages), PH and its elements, [1,3] for NP, [14] for pspacef and the class of the elementary functions, [7] for the de nition of a time-space hierarchy between ptimef and pspacef). All these approaches have been dubbed *Implicit Computational Complexity*: they share the idea that no explicitly bounded schemes are needed to characterize a great number of classes of functions and that, in order to do this, it su ces to distinguish between *safe* and *unsafe* variables (or, following Simmons [17], between *dormant* and *normal* ones) in the recursion schemes. This distinction yields many forms of *predicative recurrence*, in which the being-de ned function cannot be used as counter into the de ning one.

Statement of the result. We de ne a *safe recursion* scheme srec on a ternary word algebra, such that $f(x; y; za) = h(f(x; y; z); y; za)$, where $x; y; z$ are, respectively, the auxiliary variable, the parameter and the recursion variable; no

other type of variables can be used, and the identi cation of z with x is not allowed. We also de ne a *constructive diagonalization* scheme cdiag, such that $f(n) = fe(n)g(n)$, where fmg is the standard Klenee's notation for the function coded by m and e is a given enumerator.
Starting from a characterization T_1 of lintimef, and from an assignment of fundamental sequences $_n$ for ordinals $< _0$ (see [16] for further details on ordinals), we de ne the hierarchy $fT\ g_{<\ 0}$ as follows:

1. at each successor ordinal, T_{+1} is the class of all functions obtained by one application of safe recursion to functions in T ;
2. at each limit ordinal , T is the class of all functions obtained by one application of constructive diagonalization in an enumerator $e\ 2\ T_{1}$, such that $fe(n)g\ 2\ T_{n}$.

Given an ordinal in Cantor normal form, $B\ (n)$ is the max$(2; n)^{clps(;n)}$, where $clps(;n)$ is the result of replacing $!$ by n in . We have that:

1. for all nite k, T_k = dtimef(n^k);
2. for all $! < \ < _0$, dtimef$(B\ (n))$ T dtimef$(B\ (n + O(1)))$.

Thus, $S_{<!} T$ = ptimef and $S_{< _0} T = E$, the elementary functions.
In analogy with $fT\ g_{< _0}$ we de ne a hierarchy $fS\ g_{< _0}$ of *not space-increasing* functions and, by means of a restricted form of substitution, we de ne a time-space hierarchy $fTS_{qp}g_{qp<!}$, such that
3. TS_{qp} = dtimespacef$(n^p; n^q)$.

2 Costants, Basic Functions, and De nition Schemes

2.1 Recursion Free Functions

T is the ternary alphabet $f0; 1; 2g$. $p; q; \ldots; s; \ldots$ are the word 0 or words over T not beginning with 0; is the empty word. B is the binary alphabet $f1; 2g$. $U; V; \ldots; Y$ are words over B. $a; b; a_1; \ldots$ are letters of T or B.
The *i-th component* $(s)_i$ of a word s of the form $Y_n0Y_{n-1}0\ldots0Y_20Y_1$ is Y_i. The rationale of this de nition is that ternary words are actually handled as tuples of binary words, with the zeroes playing the role of commas. jsj is the length of the word s.
We denote with $x; y; z$ variables used as, respectively, auxiliary, parameter and principal in the construction of the current function. $f(s; t; r)$ is the result of assigning words $s; t; r$ to $x; y; z$. By a notation like $f(x; y; z)$ we always allow some among the indicated variables to be absent.

De nition 1. Given i 1, $a = 1; 2$ and $u = x; y; z$, the *initial functions* are the following unary functions:

1. the *identity* i(u), which returns the value s assigned to u; sometimes we write s instead of i(s);

2. the *constructor* $\mathrm{c}_i^a(u)$, which, when s is assigned to u, adds the digit a at the right of the last digit of $(s)_i$, ; it leaves s unchanged if $(s)_i$ is a single letter;
3. the *destructor* $\mathrm{d}_i(s)$, which, when s is assigned to u: (a) erases the rightmost digit of $(s)_i$, if $(s)_i$ is not a single digit; (b) returns 0, otherwise. Constructors and destructors leave s unchanged if it has less than i components.

Example 1. $\mathrm{c}_1^1(12022) = 120221;\quad \mathrm{d}_2(1010) = 100;\quad \mathrm{d}_2(100) = 100.$

Definition 2. Given $i \geq 1$ and $b = 1, 2$, we have the following *simple schemes*:

1. $f = \mathrm{idt}_x(g)$ is the result of the *identification* of x as y in g;
2. $f = \mathrm{idt}_z(g)$ is the result of the *identification* of z as y in g;
3. $f = \mathrm{asg}_u(s, g)$ is the result of the *assignment* of s to the variable u in g;
4. $f = \mathrm{branch}_i^b(g, h)$ is defined by *branching* in g and h if for all s, t, r we have $f(s, t, r) =$ *if* the rightmost digit of $(s)_i$ is b *then* $g(s, t, r)$ *else* $h(s, t, r)$.

Example 2. $f = \mathrm{idt}_x(g)$ implies $f(t, r) = g(t, t, r)$. Similarly, $f = \mathrm{idt}_z(g)$ implies $f(s, t) = g(s, t, t)$. Let s be the word 110212, and $f = \mathrm{branch}_2^1(g, h)$; we have $f(s, t, r) = g(s, t, r)$, since the rightmost digit of $(s)_2$ is 1.

Definition 3. A *modifier* is the sequence composition of n constructors and m destructors.

Definition 4. Class T_0 is the closure of modifiers under branch_i^b and composition.

Definition 5. 1. The *rate of growth* $rog(g)$ of a modifier g is $n - m$, where n and m are respectively the number of constructors and destructors occurring in g.
2. For all $f \in T_0$ built-up by means of some branchings from modifiers $g_1, \ldots, g_k$, we have $rog(f) := \max_{i \leq k} rog(g_i)$.

Definition 6. Class S_0 is the class of functions in T_0 with non-positive rate of growth, that is $S_0 = \{f \in T_0 | rog(f) \leq 0\}$.

Notice that all functions in T_0 are unary, and they modify their inputs according to the result of some test performed over a fixed number of digits. Functions in S_0, or their iteration, cannot return values longer than their input.

2.2 Safe Recursion and Diagonalization

Definition 7. $f = \mathrm{srec}(g, h)$ is defined by *safe recursion* in the *basis function* $g(x, y)$ and in the *step function* $h(x, y, z)$ if for all s, t, r we have

$$\begin{aligned} f(s, t, a) &= g(s, t) \\ f(s, t, ra) &= h(f(s, t, r), t, ra). \end{aligned}$$

In particular, f =iter(h) is de ned by *iteration* of function $h(x)$ if for all $s; r$ we have

$$\begin{aligned} f(s; a) &= s \\ f(s; ra) &= h(f(s; r)): \end{aligned}$$

We write $h^{jrj}(s)$ for iter$(h)(s; r)$ (i.e. the jrj-th iteration of h on s).

De nition 8. f =cdiag(e) is de ned by *constructive diagonalization* in the *enumerator* e if for all $s; t; r$ we have

$$f(s; t; r) = fe(r)g(s; t; r)$$

where fmg denotes the function coded by m.

De nition 9. f =cmp($h; g$) is de ned by *composition* of h and g if $f(u) = g(h(u))$, with h or g in T_0.

De nition 10. Class T_1 (resp. S_1) is the closure under simple schemes and cmp of functions obtained by one application of iter to T_0 (resp. S_0).

Note that since identi cation of z as x is not allowed (see de nition 2), the step function cannot assign the previous value of the function being de ned by srec to the recursion variable. Thus, we obtain that z is a *dormant* variable, according to the Simmons' approach (see [17]), or a *safe* one, following Bellantoni&Cook: we always know in advance the number of recursive calls of the step function, and this number will never be a ected by the previous values of f.

Example 3. By a sequence of srec's we now de ne a sequence of functions g_n which, at m, *compute in unary* m^n, that is, such that $jg_n(a; t)j \quad jtj^n$. We then use the fact that the generation of the g_n's is uniform in n to de ne by cdiag a function $f_!$ which computes in unary m^m. That is:

$$g_1 := \mathrm{cmp}(\mathrm{iter}(\mathrm{c}_1^1); \mathrm{d}_1); \quad f_{n+1} := \mathrm{srec}(g_n; g_n); \; g_{n+1} := \mathrm{idt}_z(f_{n+1}).$$

We have

$$\begin{aligned} g_1(s; a) &= g_n(s; t) \\ g_1(s; ra) &= \mathrm{c}_1^1(g_1(s; r)) \end{aligned} \; ; \quad \begin{aligned} f_{n+1}(s; t; a) &= g_n(s; t) \\ f_{n+1}(s; t; ra) &= g_n(f_{n+1}(s; t; r); t): \end{aligned}$$

By induction on n and r one sees that we have $jf_{n+1}(s; t; r)j = jsj + jtj^n jrj$ and, therefore, $jg_n(a; t)j \quad jtj^n$. Assume now de ned a function $e \; 2 \; T_1$ such that $e(r) =^d g_{jrj}{}^e$. If we de ne $f_! :=$ cdiag(e), we have $jf_!(a; t; t)j = g_{jtj}(a; t) = jtj^{jtj}$.

2.3 Standard Computability

As model of computation we use the *push-down* Turing machine, and we give the de nition of inclusion between classes of TM's and classes of functions.

De nition 11. A *binary push-down* Turing machine M is de ned as follows:

{ M has k push-down tapes over the alphabet B and $m + 1$ states (0 denotes the nal state, 1 the initial state);
{ the description of M consists of m rows of the type

$$R_i = (i; j(i); i_1; j_1; I_1; i_2; j_2; I_2; i_3);$$

(one for each non- nal state) where:
(i) $i; i_1; i_2; i_3$ are states, with $i \neq 0$;
(ii) $j(i); j_1; j_2$ are tapes, with j a given function;
(iii) $I_1; I_2$ are de ned on the set of instructions $fpop; push$ 1$; push$ 2g
{ each row of M should be intended as

```
if the current state is $i$ then
          if $top(j(i)) = 1$    then enter $i_1$
                                     apply $I_1$ to $j_1$;
          if $top(j(i)) = 2$    then enter $i_2$
                                     apply $I_2$ to $j_2$;
          if $(j(i))$ is empty then enter $i_3$.
```

Given a push-down TM M with k tapes, $T_i = X$ means that the content of tape T_i is the binary word X.
Note that the previous model and the ordinary Turing machine model are equivalent, with respect to the order of time needed to compute a given function. In fact, let M be an ordinary TM, with n tapes unlimited to the left, alphabet B and a xed set of states. M can be simulated by a push-down TM N with $2n$ tapes and the same number of states, which stores the content of the i-th tape of M at the left of the observed symbol in its tape $2i$-1, and the part at the right in its tape $2i$. If M moves left on tape i, N pops a symbol from tape $2i$-1 and pushes it into tape $2i$; similarly, if M moves right on tape i, N pops a symbol from tape $2i$ and pushes it into tape $2i$-1.

De nition 12. 1. A push-down Turing machine M, by input $s = X_1 0 ::: 0 X_n$, *standard computes* $q = Y_1 0 ::: 0 Y_m$ ($M(s) =_{sc} q$) if starts with $T_i = X_i$ (1 i n) and stops with $T_j = Y_j$ (1 j m).
2. M *standard computes* the function f ($M =_{sc} f$) if $f(s) = q$ implies that $M(s) =_{sc} q$.

It is natural to observe that the number of tapes of the Turing machine which standard computes a function must be independent from the number of components of its input; following the previous de nition, a new Turing machine should be de ned for each possible number of components of the input of f. However, when we are de ning a TM that standard computes a function f, we need a number of tapes that depends only on the maximum number of components of the input that f can manipulate or check with a constructor, a destructor or a branching.

De nition 13. 1. Given f 2 T_1, we de ne the *number of components* of f (denoted with $\#(f)$) as $maxfij$d$_i$ or a c$_i^a$ or a branch$_i^b$ occurs in fg.

2. Given a function f, we de ne the *length* of f (denoted with $lh(f)$) as the number of destructors, destructors and de ning schemes occurring in its construction.

De nition 14. Given the class of time-bounded push-down TM's dtime($p(n)$) and given C a class of functions, we say that

1. dtime($p(n)$) C if for all M 2dtime($p(n)$) there exists a function f 2 C such that $M(s) =_{sc} f(s)$;
2. C dtime($p(n)$) if for all f 2 C there exists a push-down TM M with $\#(f)$ tapes such that, for all $s; t; r$, M returns $f(s; t; r)$ in time $p(jsj + jtj + jrj)$.

Let U be the nite alphabet that we use to write our functions; the code of a function f is obtained by concatenation of the codes for the letters of U which compose f; the *arity* associated with each letter ensures unique parsing.

De nition 15. The code dLe of the i-th letter L of U is $2^{i+1}1$. If the arity of L 2 U is n then $dE_ne{:}{:}{:}dE_1edLe$ codes the expression $LE_1{:}{:}{:}E_n$. We write $hE_1; {:}{:}{:}; E_ni$ for $dE_ne{:}{:}{:}dE_1e$.

In the same way we can de ne the code of a push-down TM.

De nition 16. Let M be a binary push-down TM with k tapes and $m + 1$ states.

1. The code of the row $R_i = (i; j(i); i_1; j_1; l_1; i_2; j_2; l_2; i_3)$ (1 i m) is the word
$$hi; j(i); i_1; j_1; l_1; i_2; j_2; l_2; i_3i:$$
2. The code of M is the word $hR_1; {:}{:}{:}; R_mi$.
3. An instantaneous description of M is coded by the word $X_1; {:}{:}{:}; X_k state$, where each X_i encodes the i-th tape of M, and *state* encodes the current state.

Lemma 1. T_1 =dtimef(n).

Proof. We rst show (by induction on the construction of f) that each function f 2 T_1 can be computed by a push-down tm in time $lh(f)n$. Base. f 2 T_0. The result follows from the de nition of initial functions and cmp (see de nition 4). Step. Case 1. f =iter(g), with g 2 T_0. We have $f(s; r) = g^{jrj}(s)$. A push-down tm M_f with $\#(f) + 1$ tapes can be de ned as follows: with $(s)_i$ on tape i (1 i $\#(f)$) it computes g (in time $lh(g)$) and, after jrj repetitions, it stops returning the nal result. Thus, M_f standard computes $f(s; r)$, within time $jrjlh(g)$.
Case 2. Let f be de ned by branching or cmp. The result follows by direct simulation of the schemes.
In order to prove the second inclusion, we show that the behaviour of an m-tape tm M (with linear time bound cn) can be simulated by a function in T_1. Indeed

a function nxt_M can be de ned in T_0, which uses two components for each tape of M, one for the part at the left of the observed symbol, one for the part at the right (read in reverse order); the internal state is stored in the $(2m + 1)$-*th* component. $nxt_M(s)$ has the form *if state*$[i](s)$ *and top*$[b](j(s))$ *then* E_{ib}, where: (a) *state*$[i](s)$ is a test which is true i the state of M is coded by i; (b) *top*$[b](j(s))$ is a test which is true i the observed character on tape $j(s)$ is b; (c) E_{ib} is a sequence of modi ers which update the code of the state and part of the tape, according to the de nition of M. By means of $c - 1$ cmp's we de ne in T_0 the function nxt^c_M, which applies c times the functions nxt_M to the word that encodes an istantaneous description of M. De ne now in T_1

$$linsim_M(x; a) = x$$
$$linsim_M(x; za) = nxt^c_M(linsim_M(x; z))$$

We have that $linsim_M(s; t)$ iterates $nxt_M(s)$ for $cjtj$ times, returning the code of the ID of M which contains the nal result.

3 The Hierarchies

3.1 Ordinals

In this section we de ne a hierarchy of functions $fT\ g\ _{<\ _0}$, starting from T_1, by means of closures under safe recursions and diagonalizations, such that $T\ _{+1}$ is de ned by one application of safe recursion to T , and T , for the limit ordinal , is obtained by diagonalization on classes $T\ _{n}$.
In the rest of this paper greek small letters are ordinal numbers, with , limit ordinals; $_n$ is the n-th element of the fundamental sequence assigned to . Recalling the de nition of the standard assignment of fundamental sequences for all $\ _0$ (cfr. [16], page 78), we introduce a slightly modi ed assignment.

$$_n = \begin{cases} n & \text{if } < !^2 \\ !\ ^n & \text{if Cantor normal form for is } ! \\ !\ n & \text{if Cantor normal form for is } !\ ^{+1} \\ +\ (!\ )_n & \text{if Cantor normal form for is } +\ ! \end{cases}$$

We now de ne a hierarchy of functions $B\ (n) := \max(2; n)^{clps(\ ;n)}$, where $clps(\ ; n)$ is the result of replacing $!$ by n in the Cantor normal form of . By simple computation one can see that $B_m(n) = n^m$, $B_{!c}(n) = n^{cn}$, $B_{!^c}(n) = n^{n^c}$, $B_{!_c}(n) = n^{n^{:::n}}$ (c times).

De nition 17. Given $1\ \ \ _0$ and a limit ordinal,

1. $T\ _{+1}$ is the closure under the simple schemes and cmp of the class of functions obtained by one application of srec to T .
2. T is the closure under the simple schemes and cmp of the class of functions obtained by cdiag(e), where $e\ 2\ T\ _1$, $fe(r)g\ 2\ T\ _{|r|}$.

Example 4. In the example 3 we have de ned a sequence of functions g_n, with n 1; according to de nition 17 one can easily verify that g_n 2 T_n, and that $f_!$ 2 $T_!$.

De nition 18. Given 1 $<$ $_0$ and a limit ordinal,

1. S $_{+1}$ is the closure under the simple schemes of the class of functions obtained by one application of srec to S .
2. S is the closure under the simple schemes of the class of functions obtained by cdiag(e), where e 2 S $_1$, $fe(r)g$ 2 S $_{|r|}$.

De nition 19. f = wsbst($h; g$) is de ned by *weak substitution* of h in g if $f(x; y; z) = g(h(x; y; z); y; z)$.

De nition 20. For all positive $p; q$, TS_{qp} is the class of functions de ned by weak substitution of h in g, with h 2 T_q, g 2 S_p and $q < p$.

Theorem 1. 1. For all nite k, T_k = dtimef(n^k).
2. For all ! $<$ $<$ $_0$, dtimef(B (n)) T dtimef(B ($n + 4$)).
3. TS_{qp} = dtimespacef($n^p; n^q$).

Proof. 1. Let f 2 T_k. In lemma 2 a TM which interprets the function f is de ned, and its runtime is proved to be in dtimef(n^k). Let M be a TM in dtimef(n^k). By lemma 3 we have that the iteration n^k times of nxt_M can be de ned in T_k; this function, starting from the code of the initial con guration of M, returns the code of its nal con guration.
2. The rst inclusion follows by lemma 3, and the second by lemma 2
3. The two inclusions follow by lemma 6 and lemma 5

The following results immediatly follow from theorem 1.

Corollary 1. 1. dtimef(n^n) $T_!$ dtimef($(n + 4)^{(n+4)}$).
2. $S_{<!} T$ = ptimef.
3. $S_{<_0} T = E$.

4 Proofs

4.1 Simulation of Functions by TM's

Lemma 2. For all ! we have T dtimef(B ($n + 4$)).
For all nite m, we have T_m dtimef($B_m(n)$).

Proof. In what follows we will de ne a $TM\ INT$, which interprets the input $dfe; s; t; r$, returning the value of f applied to its arguments. Given a function $f\ 2\ T$, we need to know, while designing INT, the number of tapes it uses. In order to do this, de ne d such that (a) $jsj + jtj + jrj \quad d$ or (b) no cdiag occurs in f and $\#(g) \quad d$, for each $g\ 2\ T_1$ occurring in the de nition of f. This means that the parts of the arguments which can be modi ed (and thus the number of tapes of INT) depend on d. At the end of this proof we reduce INT to a two-tape TM with a logarithmic increment in the time bound.

The interpreter INT uses the following stacks:

(a) $T^x; T^y; T^z$, to store the values of $x; y; z$ during the computation; each of them consists of d tapes, one for each of the modi able parts of the value assigned to x; their initial values are, respectively, $s; t; r$;

(b) T^u, to store the value of the principal variable of the current recursion;

(c) T_f, to store (the codes of) some sub-functions of f; its initial value is the code of f;

INT repeats, until T_f is not empty, the following cycle:

- it pops a function k from the top of T_f, and un-nests the outermost sub-function j of k;
- according to the form of j, it carries-out di erent actions on the stacks;
- if the form of j is iter(g) with $g\ 2\ T_0$, it calls an interpreter $ITER$ for T_1 which simulates g on T^x for jtj times, where t is on the top of T^y;
- in all other cases, it pushes into T_f an information of the form $j\ MARK\ k$, where $MARK$ informs about the outermost scheme used to de ne j.

Thus we de ne

```
$INT(dfe; s; t; r)$:=
$T_f := dfe;\ T^x := s;\ T^y := t;\ T^z := r;$
$while$ $T_f$ not empty $do$ $A :=$ last record(s) of $T_f$;
$case$
 $A = CMP(g; h)$           $then$ push $g\,h$ into $T_f$
 $A = ASG_X(p; g)$         $then$ push $g$ into $T^f$; copy $p$ into $T^X$
 $A = IDT_X(h)$            $then$ push $h$ in $T_f$; copy last record of $T^X$ into $T^y$
 $A = BRANCH_i^b(g; h)$    $then$ pop $T_f$;
                                  if $top((T^x)_i) = b$ then push $g$ into $T_f$
                                  else push $h$ into $T_f$
 $A = DIAG(h)$             $then$ push $DG\ h$ into $T_f$; copy last record of $T^x$ into $T^u$
 $A = DG$                  $then$ pop $T_f$; pop last record of $T^x$ and push it into $T_f$;
                                  pop last record of $T^u$ and push it into $T^x$
 $A = SREC(g; h)$          $then$ push $A\ RC\ g$ into $T_f$; copy last record of $T^z$ into $T^u$;
                                  push last digit of $T^u$ into $T^z$
 $A = SREC(g; h)\ RC$      $then$ if $T^u = T^z$ then pop $T_f$; pop $T^u$; pop $T^z$
                                  else push $h$ into $T_f$; push last digit of $T^u$ into $T^z$
 $A = ITER(g)$             $then$ call $ITER$.
end $case$;
end $while$.
```

We now show that for all $f; s; t; r$ respecting the imposed condition over d we have

$$f\ 2\ T \quad !\ INT(dfe; s; t; r) = f(s; t; r) \text{ within time } jsj + jdfejB \ \ (jtj + jrj + 1 \ \ )$$

where 1 = 0 if $< !$ and 1 = 1 otherwise. The result follows, since every function $f\ 2\ T$ is then computed in dtimef(B (n + 1)) by the composition of the constant-time tm writing the code of f with INT.
De ne $m := jsj;\ n := jtj + jrj;\ c := jdfej$. We show that, for all $f\ 2\ T$, INT moves within $m + cB$ (n + 1) steps from an istantaneous description of the form

$$T_f = Zdfe;\ T^x = s_0 s;\ T^y = t_0 t;\ T^z = r_0 r;\ T^u = q;$$

to a new istantaneous description of the form

$$T_f = Z;\ T^x = s_0 f(s; t; r);\ T^y = t_0 t;\ T^z = r_0 r;\ T^u = q:$$

Induction on the construction of f. Basis. Let $f\ 2\ T_1$. We have 1 = 0. The complexity of $ITER$ is obviously bounded by $m + cn$.
Step. Case 1. f =srec($g; h$). We have = + 1; let r be the word $a_{jrj} ::: a_1$. By the inductive hypothesis, INT needs time $m + jdgejB$ (n + 1) to produce the istantaneous description

$$T_f = ZdfeRC;\ T^x = s_0\, g(s; t; a_1);\ T^y = t_0 t;\ T^z = r_0 r a_1;\ T^u = qr:$$

If $jrj > 1$ then INT puts $T_f := Z\ dfe\ RC\ dhe$ and $T^z := r_0 r a_2 a_1$, and calls itself in order to compute h and the next value of f. By the inductive hypothesis we have that INT needs time $jg(s; t; a_1)j + jdhejB$ (n + 1) to produce an istantaneous description of the form

$$T_f = ZdfeRC;\ T^u = qr;$$

$$T^x = s_0\ (h(g(s; t; a_1); t; a_2 a_1));\ T^y = t_0 t;\ T^z = r_0 r a_2 a_1:$$

After jrj simulations of h we obtain the promised istantaneous description within an amount of time

$$m + jrj \max(jdgej; jdhej)B \ \ (n + 1 \ \ ) \quad m + jrjcB \ \ (n + 1 \ \ ) \quad m + cB \ \ (n + 1 \ \ )$$

where, since 2, in these evaluations we may compensate the quadratic amount of time needed to copy r and its digits with the di erence between c and max($jdgej; jdhej$).
Case 2. f =cdiag(h) $2\ T$. We have $h\ 2\ T_1$ and (recall that $_1$ = 1 when $< !^2$)

$$B_{\ 1}(n + 1) + B_{\ n}(n + 1) \quad B_{\ n+1}(n + 1) = B \ (n + 1): \tag{1}$$

INT computes $h(r)$, understands from the mark DG that the result is the code for the function to be computed, and, accordingly, pushes it into T_f.

To compute $h(r)$ and $fh(r)g(s; t; r)$ the interpreter INT needs, by the inductive hypothesis, time m + $jdhejB$ $_1(jrj$ + 1) + $jfh(r)gjB$ $_{|r|}(n$ + 1) (by (1))m + $jdhejB$ $(n$ + 1) m + cB $(n$ + 1)$:$

Reduction to two tapes. The interpreter we have just de ned uses a number of tapes d that depends on the construction of the simulated function. We use now the general procedure showed in [9] to reduce a k-tape TM bounded by $T(n)$ to a 2-tape TM bounded by $kT(n)\log T(n)$; thus, we obtain a 2-tape interpreter bounded by

$$T(n) = kB \ (n + 1)\log B \ (n + 1) \quad kB \ (n + 2)\log(n + 1) \quad B \ (n + 4)$$

4.2 Simulation of TM's by Functions

Lemma 3. For all 1 $<$ $_0$, dtimef(B (n)) T .

Proof. Let M be a TM in dtimef(B (n)). There exists a function nxt_M 2 T_0 such that, for input the code of an istantaneous description of M, nxt_M returns the code of the next description. We de ne the following function:

$$(s; d \ e) = \begin{cases} sdnxt_M e & \text{if } = 0 \\ (s; d \ e) \ (s; d \ e) d\text{srec}e & \text{if } = \ + 1 \\ (s; d \ _1e) \ (s; d \ _{jrj}e) d\text{srec}e d\text{cdiag}e & \text{if } \text{ is a limit ordinal} \end{cases}$$

We prove by induction on that the function whose code is generated by is in T .

Base. = 0. We have that nxt_M 2 T_0, by hypothesis.

Step. Case 1. = + 1. We have that f $(s; d \ e)g$ =srec($(s; d \ e);$ $(s; d \ e)$). The function is in T , by induction on and de nition 17.

Case 2. = . We have that f $(s; d \ e)g$ = cdiag(srec($(s; d \ _1e);$ $(s; d \ _{jsj}e)$)). This function is in T , since f $(s; d \ _1e)g$ 2 T $_1$ and, for all s, f $(s; d \ _{jsj}e)g$ 2 T $_{|s|}$.

The function writes the code of a function which iterates nxt_M for B (n) times; by input the code of the initial con guration of M, this function returns the codes of the nal con guration.

Note that, given the code of a limit ordinal , we need at most a quadratic amount of time to return the code of $_n$.

4.3 Time-Space Classes

Lemma 4. For all f in S_p, we have $jf(s; t; r)j$ max($jsj; jtj; jrj$).

Proof. By induction on p. Base. f 2 S_1.

Case 1. f is de ned by iteration of a function g in S_0; we have, by induction on r, $jf(s; a)j$ = jsj, and $jf(s; ra)j$ = $jg(f(s; r))j$ $jf(s; r)j$ max($jsj; jrj$)$:$

Case 2. f is de ned by simple scheme or cmp. The result follows by the inductive hypothesis.

Step. Given $f\ 2\ S_{p+1}$, de ned by srec in functions g and h in S_p, we have

$$\begin{array}{lll} jf(s;t;a)j = jg(s;t)j & & \text{by de nition of } f \\ & j\max(jsj;jtj)j & \text{by inductive hypothesis.} \end{array}$$

and

$$\begin{array}{lll} jf(s;t;ra)j = & jh(f(s;t;r);t;ra)j & \text{by de nition of } f \\ & j\max(jf(s;t;r)j;jtj;jraj)j & \text{by inductive hypothesis on } h \\ & j\max(\max(jsj;jtj;jrj);jtj;jraj)j & \text{by induction on } r \\ & j\max(jsj;jtj;jraj)j: & \end{array}$$

Lemma 5. TS_{qp} dtimespacef$(n^p;n^q)$.

Proof. Let f be a function in TS_{qp}. By de nition 20, f is de ned by weak substitution of a function $h\ 2\ T_q$ into a function $g\ 2\ S_p$, that is, $f(s;t;r) = g(h(s;t;r);t;r)$. The theorem 1 states that there exists an interpreter INT computing the values of h within time n^q, and computing the values of g within time n^p. The lemma 4 holds for g, since g belongs to S_p; thus, the space needed by INT to compute g is at most n.
De ne now a TM M that, by input $dfe;s;t$, and r performs the following steps (recall lemma 2 for the de nition of INT):
(1) it pushes $dgedhe$ into the tape T^f of INT, which contains the codes of the functions that the interpreter will compute;
(2) it calls INT on input $dgedhe;s;t;r$.
The time complexity of (1) is linear in the length of dfe; in (2), INT needs time equal to n^q to compute h, and needs only n^p (and not n^{q^p}) to compute g. This happens because $h(s;t;r)$ is computed in the safe position, and this implies that its length does not a ect the number of steps performed by the second call to INT. In fact, INT never moves the content of a safe position into the tapes whose values play the role of recursive counters; they depend only on n, the length of the original input. Thus, the overall time bound is $n^q + n^p$, which can be reduced to n^p, being $q < p$.
INT requires space n^q to compute the value of h on input $s;t;r$; as we noted above, the space needed for the computation of g is linear in the length of the input, and thus the overall space needed by M is still n^q.

Lemma 6. dtimespacef$(n^p;n^q)$ TS_{qp}

Proof. Let M be a TM in dtimespacef$(n^p;n^q)$. This means that the computation of M is time-bounded by n^q and, simultaneously, it is space-bounded by n^p. M can be simulated by the composition of two TM's, M_g and M_h, with $M_h\ 2$dtimef(n^q) and $M_g\ 2$dtimespacef$(n^p;n)$: the former constructs (within polynomial time) the space that the latter will successively use in order to simulate M.
By theorem 1 there exists a function $h\ 2\ T_q$ which simulates the behaviour of M_h, and there exists a function $g\ 2\ S_q$ which simulates the behaviour of M_g; in

particular, there exists the function nxt_g 2 T_0. Note that nxt_g belongs to S_0, since it never adds a digit to the description of M_g without erasing another one. We de ne the function $_0$:=iter(nxt_g), and the sequence y_{n+1} :=srec($_n;$ $_n$), with $_{n+1}$:=idt$_z$(y_{n+1}).
We have that

$$y_1(s;t;a) = \text{iter}(nxt_g)(s;t)$$
$$y_1(s;t;ra) = \text{iter}(nxt_g)(y_1(s;t;r);t)$$

and

$$y_{n+1}(s;t;a) = \quad _n(s;t)$$
$$y_{n+1}(s;t;ra) = \quad _n(y_{n+1}(s;t;r);t;ra)$$

We can easily see that $_0$ 2 S_1, by de nition of this class and, again by de nition 18, that $_n$ 2 S_{n+1}.
Given the code s of the initial id of M, we can de ne $sim_M(s) = \quad _{p-1}(h(s);s)$, which simulates the behaviour of M. This function is de ned in TS_{qp}.

References

1. S.J. Bellantoni, *Predicative recursion and the polytime hierarchy*, in P.Clote and J.Remmel (eds), Feasible Mathematics II (Birkauser, 1994), 320-343.
2. S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of the polytime functions*, Computational Complexity 2(1992)97-110.
3. S. Caporaso, *Safe Turing machines, Grzegorczyc classes and polytime*, Intern. J. Found. Comp. Sc., 7.3(1996)241-252.
4. S. Caporaso, N. Galesi, M. Zito, *A predicative and decidable characterization of the polytime classes of languages*, to appear in Theoretical Comp. Sc.
5. S. Caporaso, M. Zito, N. Galesi, E. Covino *Syntactic characterization in Lisp of the polynomial complexity classes and hierarchies*, in G. Bongiovanni, D.P. Bovet, G. Di Battista (eds), Algorithms and Complexity, LNCS 1203(1997)61-73.
6. S. Caporaso, G. Pani, E. Covino, *Predicative recursion, constructive diagonalization and the elementary functions*, Workshop on Implicit Computational Complexity (ICC99), a liated with LICS99, Trento, 1999.
7. P. Clote, *A time-space hierarchy between polynomial time and polynomial space*, Math. Sys. The. 25(1992)77-92.
8. A. Cobham, *The intrinsic computational di culty of functions*, in Y. Bar-Hillel (ed), Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science, pages 24-30, North-Holland, Amsterdam, 1962.
9. F.C.Hennie and R.E.Stearns, *Two-tape simulation of multi-tapes TM's*, Journal of ACM 13.4(1966)533-546.
10. D. Leivant, *A foundational delineation of computational feasibility*, Proc. of the 6th Annual IEEE symposium on Logic in Computer Science, (IEEE Computer Society Press, 1991), 2-18.
11. D. Leivant, *Strati ed functional programs and computational complexity*, in Conference Records of the 20th Annual ACM Symposium on Principles of Programming Languages, New York, 1993, ACM.
12. D. Leivant, *Rami ed recurrence and computational complexity I: word recurrence and polytime*, in P.Clote and J.Remmel (eds), Feasible Mathematics II (Birkauser, 1994), 320-343.

13. D. Leivant and J.-Y. Marion, *Rami ed recurrence and computational complexity II: substitution and polyspace*, in J. Tiuryn and L. Pocholsky (eds), Computer Science Logic, LNCS 933(1995) 486-500.
14. I. Oitavem, *New recursive characterization of the elementary functions and the functions computable in polynomial space*, Revista Matematica de la Universidad Complutense de Madrid, 10.1(1997)109-125.
15. R.W. Ritchie, *Classes of predictable computable functions*, Transactions of A.M.S., 106, 1993.
16. H.E. Rose, *Subrecursion: Functions and hierarchies*, Oxford University Press, Oxford, 1984.
17. H. Simmons, *The realm of primitive recursion*, Arch.Math. Logic, 27(1988)177-188.

Group Updates for Red-Black Trees

Sabine Hanke and Eljas Soisalon-Soininen

[1] Institut für Informatik, Universität Freiburg,
Am Flughafen 17, D-79110 Freiburg, Germany,
hanke@informatik.uni-freiburg.de

[2] Department of Computer Science and Engineering, Helsinki University of Technology, P.O.Box 5400, FIN02015 HUT, Finland,
ess@cs.hut.fi

Abstract. If several keys are inserted into a search structure (or deleted from it) at the same time, it is advantageous to sort the keys and perform a group update that brings the keys into the structure as a single transaction. A typical application of group updates is full-text indexing for document databases. Then the words of an inserted or deleted document, together with occurrence information, form the group to be inserted into or deleted from the full-text index. In the present paper a new group update algorithm is presented for red-black search trees. The algorithm is designed in such a way that the concurrent use of the structure is possible.

1 Introduction

If a large number of keys are to be inserted into a database index at one time, then it is important for e ciency that the keys are sorted and inserted into the index tree as a group. In this way, it is not necessary to traverse the whole path from the root to the leaf when performing each insertion.

Typical applications where group operations are needed are databases in which large collections of data are stored at one time, such as document databases or WWW search engines [13, 14]. In such systems, full-text indexing is applied for term search. In the inverted-index technique, for each index term an occurrence list is created that includes all documents the term appears in. The terms themselves are organized as a search structure, typically as a tree. When inserting a new document, an update of the inverted index is required for each term in the inserted document [4, 5]. This can be a long transaction, and without concurrency intolerable delays can occur if searches in the database are frequent. To overcome this problem concurrent group update algorithms have been designed [13].

Other applications arise, e.g., when updates occur in bursts and are collected into groups that are merged in certain intervals with the main index [7], and when large indexes are constructed on-line, i.e., during the construction of the index, new records may be inserted into the le to be indexed [11, 16]. To nish the indexing, a group update is created from the updates which occurred during the main construction.

In this paper we present an e cient group update algorithm for red-black search trees [6]. The algorithm has two steps: First, the operations in the underlying group are performed without any balancing, except for subgroups between two consecutive keys in the original tree. In this way the updates are made available as soon as possible without sacri cing the logarithmic search time. In the second step, the tree will be balanced, i.e., transformed into a tree satisfying the (local) balance criteria of red-black trees. Balancing is designed as a background process allowing the concurrent use of the structure. The balancing time is comparable with earlier results in cases when balancing is strictly connected with individual updates.

Recently, a group insertion algorithm for AVL-trees [1] was presented in [10]. The motivation for the new algorithm is two-fold. First, red-black trees are more e cient than AVL-trees as regards the number of rebalancing operations needed for updates, see [8], for example. Thus, it is important to develop group update algorithms for red-black trees, not only for AVL-trees. Moreover, our new implementation of group updates for red-black trees has an important advantage over the implementation given in [10]. The algorithm given in [10] is based on height-valued trees de ned in [9]. This implies that even when a subgroup consists of a single key and no rebalancing is needed, all nodes in the search path must be checked for imbalance and the balance information must be stored. Our new algorithm is based on a generalization of red-black trees, called chromatic trees [12] or relaxed red-black trees [8], and no unnecessary checking for imbalance and restoring balance information is needed.

2 Chromatic Trees

We shall consider *leaf-oriented* binary search trees, which are full binary trees (each node has either two or no children) with the keys stored in the leaves. The internal nodes contain *routers*, which guide the search through the tree. The router stored in a node v must be greater than or equivalent to any key stored in the leaves of v's left subtree and smaller than any key in the leaves of v's right subtree.

We de ne a *chromatic tree* as a relaxed version of a red-black tree. Instead of using the two colors, red and black, we use *weights*. Each node in the tree has a non-negative integer weight. We refer to nodes with weight zero as red and nodes with weight one as black. If a node has a larger weight, we call it *overweighted*, and its amount of *overweight* is its weight subtracted by 1. The only requirements in this chromatic tree are that *all paths from the root to a leaf have the same weight* and that *leaves have a weight of at least one*.

In a chromatic tree two consecutive red nodes are referred to as a *red-red conflict* (which is assigned to the lower of both nodes), and an overweighted node as an *overweight conflict*. A red-black tree can be de ned as a chromatic tree without any conflicts. The purpose of the rebalancing operations is to transform a chromatic tree into a red-black tree by removing all conflicts.

Operations for the updates, insertion, and deletion, as well as for rebalancing, can be found in [2, 12]. Note that all operations preserve the tree as a chromatic tree, i.e., leaves are not red and all paths have the same weight.

3 Group Insertion

The overall structure of an e cient group update for red-black trees is the same as for AVL trees [10]. For insertions, this essentially amounts to inserting an ordered set of keys, starting at the root of the tree, and propagating subsets down the tree to the appropriate insertion locations. For a leaf search tree, we arrive at a set of leaves, with a set of keys to be inserted at each of these leaves.

Thereafter, each set of keys to be inserted is turned into a (usually small) red-black tree. With a certain number of rebalancing operations that arise from applying the rebalancing scheme of chromatic trees, the balance condition is restored.

As result of a group insertion, a node of the tree may also get negative units of overweight. Nodes that have negative weights are called *underweighted* nodes. An underweighted node p corresponds to a red node, the color of which additionally stores information about how many black nodes are too much on each search path in the subtree rooted at p compared with the rest of the tree.

The motivation for using underweighted nodes is to expedite the rebalancing, since the nodes of a new subtree T_i that is inserted by the group insertion can be colored red-black during the creation of T_i without much e ort. On the other hand, if all internal nodes of T_i are colored red analogously, as by a sequence of single insertions, the red-black coloring of T_i must be part of the rebalancing. Because the color of at least each second node on a path in T_i must then be changed, $\Omega(m)$ transformations are needed.

Let T be a chromatic tree of size n that ful lls the balance conditions of red-black trees and K a group of m sorted keys. K is split into b subgroups K_i ($i = 1; \ldots; b$) of m_i keys so that the search for each of the keys of K_i ends at the same leaf l_i of T. Let l_i store key k_i.

Operation group insertion:

Step 1 For all $i = 1; \ldots; b$ construct a balanced red-black tree T_i that stores the m_i keys of the ith subgroup and the key k_i. The root of T_i gets the underweight $1 - w_i$, where w_i is the number of black nodes on each path from the children of the root to the leaves.

Step 2 For all $i = 1; \ldots; b$ replace leaf l_i of T by the tree T_i. Denote the resulting tree by T^0.

Step 3 Rebalance T^0 by small local transformations.

In order to construct T_i in $O(m_i)$ time, a leaf-oriented AVL-tree [1] is generated from the set of $m_i + 1$ keys by applying a simple divide-and-conquer algorithm. During the construction the nodes are colored red and black using the criterion by Guibas and Sedgewick [6].

In order to rebalance T^0 basically the set of rebalancing transformations of the chromatic tree is used [2]. The underweights only serve for storing information about accumulated insertions. During the rebalancing they are transformed into red-red conflicts and overweight conflicts again.

An underweighted node p is handled as follows. If p is the root of the tree, then the underweight conflict is resolved by coloring p black (cf. Figure 1, u-root). Otherwise, assume that p has weight $w(p) = -g$, and the sibling q of p has weight greater than or equal to $-g$. Then p is colored red, the weight of p's parent is decreased by g, and the weight of q is increased by g (cf. Figure 1, reversal).

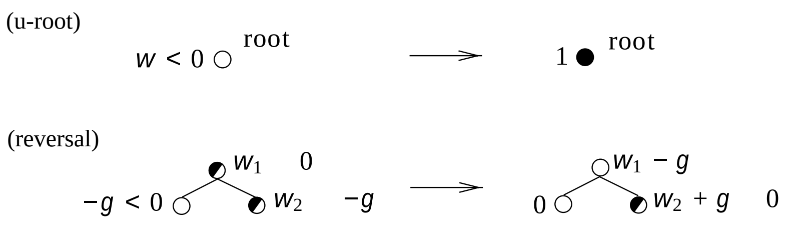

Fig. 1. Handling of an underweight conflict. Symmetric cases are omitted. (A label beside a node denotes the weight of the node. In order to represent the colors of the nodes more clearly, additional black or overweighted nodes are lled and red or underweighted nodes are un lled. Half lled nodes have an arbitrary weight.)

If the parent of p is black or overweighted before applying the reversal transformation, then the reversal transformation resolves at least one unit of underweight from the tree. Otherwise, the underweight is only shifted from p to its parent. By a reversal transformation to handle the underweight of p, the sibling q of p may become overweighted, $w(q) \quad g$. Furthermore, a constant number of red-red conflicts may be generated by a reversal transformation (at the nodes which are colored red and at the red children of these nodes).

In the following we analyze the costs of rebalancing T^0. Let r_i ($i = 1;:::;b$) be the underweighted nodes of T^0 with weights $w(r_i) = -g_i \quad 0$. The r_i are the roots of the new subtrees T_i created by the group insertion.

The ith *branching node* ($i = 1;:::;k-1$) of a group search path from the root to nodes $q_1;:::;q_k$ is de ned as the nearest common ancestor of q_i and q_{i+1}. A *stopping node* is a branching node, both subtrees of which contain red-red conflicts.

Lemma 1. *At most* $2 \quad \sum_{i=1}^{b} g_i$ *reversal transformations are needed to resolve the* $\sum_{i=1}^{b} g_i$ *units of underweight from* T^0.

Proof. The rebalancing of T^0 is done along the group search path to the nodes $r_1, \ldots, r_b$. We start with transforming the underweights into red-red and overweight conflicts by using reversal transformations. This transformation is always done in bottom-up direction. That means a reversal transformation is applied at an underweighted node p only if the subtrees rooted at p and its sibling q do not contain any underweighted nodes except for p and q. The question whether those two subtrees contain underweighted nodes can be answered locally by marking the branching nodes during the search phase of the group insertion and for each T_i storing the information as to whether the underweight still exists.

Since T ful lled the balance conditions of red-black trees before the group insertion, at least every second node on the path from the root to a node r_i is black in T^0. Therefore, at most $2g_i$ reversal transformations are needed to resolve the underweight of r_i.

If the underweights of two nodes r_i and r_{i+1} meet at children of a branching node, both underweights are handled by the same reversal transformation. Thereby one of the underweights is always resolved (cf. Figure 1). □

Let us now consider the situation after performing the reversal transformations. Since each reversal transformation that handles an underweight which was originally created at r_i may generate only a constant number of red-red conflicts and g_i units of overweight, it follows:

Lemma 2. *The number of red-red conflicts generated by the reversal transformations is $O(\sum_{i=1}^{b} g_i)$, and the sum of units of overweight is $O(\sum_{i=1}^{b} g_i^2)$.*

In contrast to underweights and overweights, red-red conflicts must always be handled in a top-down manner. Thus, the rebalancing of the red-red conflicts starts with the top-most red-red conflict on the group search path from the root to the nodes $r_1, \ldots, r_b$. We demand that during the rebalancing of the red-red conflicts the following condition must be guaranteed:

Condition: Whenever the parent or the grandparent of a node has a red-red conflict which is handled as a stopping node, then both children of this stopping node must be red.

The Condition is motivated by the following idea: a rebalancing transformation should be carried out at a branching node only if either red-red conflicts of both subtrees have been bubbled up to this node, so that they can be handled together, or if one of the subtrees contains no red-red conflicts any more.

Lemma 3. *The number of transformations needed to resolve all red-red conflicts is $O(\sum_{i=1}^{b} g_i + L)$, where L is the number of di erent nodes on the group search path from the root to $r_1, \ldots, r_b$.*

Proof (sketch). Let k be the number of red-red conflicts. l denotes the number of pairs of red siblings where at least one of the siblings lies on the group search path from the root to $r_1, \ldots, r_b$. Let v be the number of nodes p with red-red conflicts that have a distance greater than one to the nearest stopping node that is an ancestor of p. We de ne $\ := 5k + 2l + v \quad 0$.

By a simple case analysis it can be veri ed that, by performing a rebalancing transformation to handle a red-red conflict, is always decreased if, during the handling of the red-red conflicts, the Condition is guaranteed. Therefore, at most $5k + 2l + v$ rebalancing transformations are needed to resolve all red-red conflicts. Since k and v are in $O(\sum_{i=1}^{b} g_i)$, cf. Lemma 2, and $l \quad L$, the claim follows. □

Theorem 1. *$O(\sum_{i=1}^{b} \log^2 m_i)$ rotations and $O(\sum_{i=1}^{b} \log^2 m_i + L)$ color changes are needed to rebalance T^0, where L is the number of di erent nodes on the group search path from the root to $r_1; : : : ; r_b$.*

Proof. After inserting the subtrees $T_1; : : : ; T_b$ into T, the tree contains b underweighted nodes $r_1; : : : ; r_b$ with weights $w(r_i) = -g_i$ $(i = 1; : : : ; b)$, where g_i is in $O(\log m_i)$, since the subtree T_i rooted by r_i contains $O(\log m_i)$ black nodes on each search path.

First, the $\sum_{i=1}^{b} g_i$ units of underweight are transformed into $O(\sum_{i=1}^{b} g_i)$ red-red conflicts and $O(\sum_{i=1}^{b} g_i^2)$ units of overweight (Lemma 2) at at most $2\sum_{i=1}^{b} g_i$ overweighted nodes by using $O(\sum_{i=1}^{b} g_i) = O(\sum_{i=1}^{b} \log m_i)$ reversal transformations (Lemma 1). Then the red-red conflicts are resolved from the tree by using $O(\sum_{i=1}^{b} g_i) = O(\sum_{i=1}^{b} \log m_i)$ rotations and $O(\sum_{i=1}^{b} \log m_i + L)$ color changes (Lemma 3). Finally, the overweight conflicts are handled analogously as in Step 2 of a group deletion (cf. the following section). For this, $O(\sum_{i=1}^{b} g_i^2) = O(\sum_{i=1}^{b} \log^2 m_i)$ rotations and $O(\sum_{i=1}^{b} g_i^2 + L \ )$ color changes are necessary, where L is the number of di erent nodes on the group search path from the root to the overweighted nodes.

Before handling the red-red conflicts $L \quad L + 2\sum_{i=1}^{b} g_i$, because all of the at most $2\sum_{i=1}^{b} g_i$ overweighted nodes are siblings of nodes on the group search path from the root to $r_1; : : : ; r_b$. Since each of the $O(\sum_{i=1}^{b} g_i)$ rotations needed to handle the red-red conflicts increases L by at most one, L is in $O(L + \sum_{i=1}^{b} g_i)$. So, the number of color changes needed to resolve all overweights from the tree is bounded by $O(\sum_{i=1}^{b} \log^2 m_i + L)$. Therefore, after performing step 1 and 2 of a group insertion, the tree can be rebalanced by using $O(\sum_{i=1}^{b} \log^2 m_i)$ rotations and $O(\sum_{i=1}^{b} \log^2 m_i + L)$ color changes. □

Assuming that each path in T from the root to a leaf contains the same number of black nodes, it can analogously be shown as for 2-3-trees [3] that the number L of di erent nodes on the group search path is $O(\log n + \sum_{i=1}^{b-1} \log d_i)$, where d_i denotes the number of leaves between l_i and l_{i+1}.

4 Group Deletion

Let T be a chromatic tree of size n that ful lls the balance conditions of red-black trees and K a group of m sorted keys. K is split into b subgroups K_i $(i = 1; : : : ; b)$

of m_i keys so that for each K_i, the tree T contains a subtree T_i that stores all keys of K_i plus an additional key k_i at its leaves.

Operation group deletion:

Step 1 For all $i = 1; : : : ; b$ replace the subtree T_i by a leaf l_i that stores the key k_i. l_i gets weight w_i, where w_i is the sum of weights on each path from the root of T_i to a leaf. Denote the resulting tree by T^0.

Step 2 Rebalance T^0 by small local transformations.

In order to estimate the costs of rebalancing T^0, rst we consider the following situation: Let p be an overweighted node, of which the sibling q is the root of a balanced subtree T^q that contains no overweights. (After performing a group deletion, this situation occurs at $p = l_i$, for example, if the sibling q is not equal l_{i-1} or l_{i+1} respectively.) Denote the parent of p and q by u. Let $w(p) = g+1 \quad 2$ be the weight of p.

Lemma 4. *At most* $2g$ *push transformations plus* g *further rebalancing transformations are needed in order to transform* T^u *into a balanced red-black tree* T^{u^0}. *Afterwards the root* u^0 *of* T^{u^0} *has weight* $w(u^0) \quad w(u) + 1$.

Proof (sketch). In order to decrease $w(p)$ from $g + 1$ to 1, g rebalancing transformations are carried out. Thereby the subtree T^u rooted at u is replaced by a subtree T^u rooted by u (cf. Figure 2).

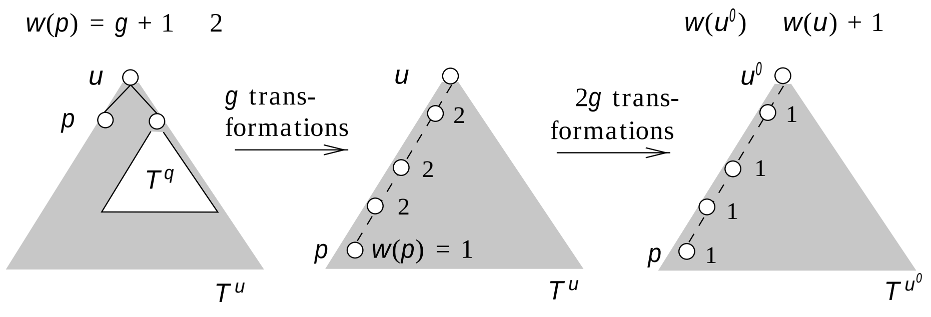

Fig. 2. Rebalancing T^u.

By observing the weight-balancing transformations, it can be shown that, thereby, at most half of the g units of overweight are resolved. The remaining k overweight conflicts are spread out over the path from u to p, i.e., except for u , all nodes on the path from u to p (of length $\quad 2(g - k) + 2$) have a weight less than or equal to two (see Figure 2).

Then the overweights on the path from u to p are handled in bottom-up direction by using at most $2g - k$ push transformations and k further rebalancing transformations. Thereby, T^u is replaced by a subtree T^{u^0} rooted by u^0. Only one of the k units of overweight may remain at u^0.

Since the set of the g rebalancing transformations that transform T^u into T^u contain at most k push transformations, in total at most $2g$ push transformations plus g further rebalancing transformations are needed in order to rebalance T^u. Afterwards $w(u^0) \quad w(u) + 1$. □

Theorem 2. *$O(\sum_{i=1}^{b} \log m_i)$ rotations and $O(\sum_{i=1}^{b} \log m_i + L)$ color changes are needed to rebalance T^0, where L is the number of di erent nodes on the group search path from the root to the overweighted leaves $l_1; : : : ; l_b$.*

Proof. For $i = 1; : : : ; b$ let $w(l_i) = g_i + 1$.

In order to avoid unnecessary work during the rebalancing, we slightly modify the w7 transformation [2], which handles overweight conflicts at sibling nodes. Instead of resolving only one unit of overweight as w7 does, w7 resolves the maximum number of overweight conflicts at a time (cf. Figure 3).

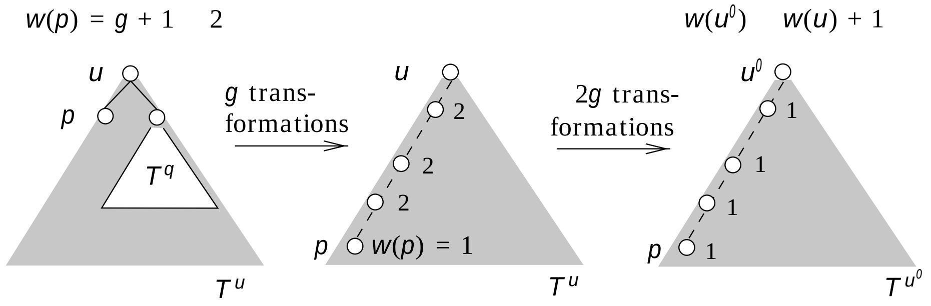

Fig. 3. Modi ed (w7)-transformation.

The rebalancing of T^0 is always done in bottom-up direction. That means an overweight conflict at a node p is handled only if the subtrees rooted at p and its sibling q both are balanced red-black trees that do not contain overweighted nodes except for p and q. Such a node p always exists in T^0, if the group deletion has not removed all leaves of the tree. At the beginning of the rebalancing, p is one of the leaves l_i and q may be l_{i-1} or l_{i+1} respectively.

The question as to whether two subtrees rooted at sibling nodes are balanced can be answered locally by marking the branching nodes during the search phase of the group deletion and by storing for each l_i the information whether overweight-conflicts generated at l_i still exist in the tree.

Let p be an overweighted node so that the subtree rooted at p and the subtree rooted at p's sibling q are both balanced. Without loss of generality $w(p) \quad w(q)$. Let u be the parent of p and q. T^u denotes the subtree rooted at u.

Case 1: [$w(q) \quad 1$ and $w(p) = 2$] One single transformation is su cient either to resolve the overweight conflict at p or to shift it to p's parent u.

Case 2: *[$w(q)$ 1 and $w(p) := g_p + 1 > 2$]* In this case, the subtree T^u is rebalanced analogously as described in the proof of Lemma 4 by using at most $3g_p$ transformations. Thereby, at least $g_p - 1$ units of overweight are removed from the tree. T^u is replaced by the subtree T^{u^0} and the weight of u^0 is less or equal than $w(u) + 1$.

Case 3: *[$w(q) := g_q + 1$ 2]* By performing a w7 transformation, the g_q units of overweight are removed from q, and g_q units of overweight are shifted from p to p's parent u.

The rebalancing of T^0 is now done as follows. We start at a leaf l_i and apply Case 1 and Case 2, as long as all overweight conflicts are resolved locally or, respectively, the child of a branching node is reached. Then the overweight conflicts at another l_j are handled analogously. If both children of a branching node become overweighted, Case 3 applies. If only one subtree of a branching node contains overweights, these overweight conflicts are handled by applying Case 1 and Case 2. Then at most one unit of overweight may remain at the branching node. In both cases, afterwards an overweight conflict at the branching node is handled analogously as at one of the leaves l_i.

Because T^0 contains $\sum_{i=1}^{b} g_i$ units of overweight, the number of rotations needed to rebalance T^0 is $O(\sum_{i=1}^{b} g_i) = O(\sum_{i=1}^{b} \log m_i)$.

Each time Case 2 or Case 3 applies, the number of overweight conflicts is reduced. Thereby, $O(\sum_{i=1}^{b} \log m_i)$ transformations are performed. In Case 1 either the overweight conflict is resolved or it is shifted along the search path towards the root. Thus, Case 1 applies $O(L)$ times. Therefore, the number of color changes needed to rebalance T^0 is $O(\sum_{i=1}^{b} \log m_i + L)$. □

5 Conclusions

There are applications in which a large number of updates for a search structure is created in a very short time, for example by measuring equipment, or when a document is inserted into a document database. It is often important that such a group is brought into the structure as fast as possible, and that during this group update the concurrent use of the structure is allowed.

Our work in the present paper is along the lines of [10], where a group insertion algorithm for AVL-trees was presented and analyzed. The novelty of the present paper is that we consider red-black binary trees, and that we obtain a group update algorithm that is more e cient than the one given in [10] in the following sense: In the algorithm of [10], all nodes in the group search path must be marked as unbalanced nodes and the balance in these nodes must be restored during the rebalancing phase of the algorithm. Our new algorithm is able to restrict the balance restoring to those nodes that have gotten out of balance because of the group update.

One important aspect of group updates not considered in the present paper is recovery, i.e., the question of how a valid search structure can be e ciently restored after a possible failure during a group update. These questions have been considered in [15] for AVL-trees, and we believe that similar methods can be applied to red-black trees.

References

[1] G.M Adel'son-Vels'kii and E.M. Landis. An algorithm for the organisation of information. *Soviet Math. Dokl.*, 3:1259{1262, 1962.

[2] J. Boyar, R. Fagerberg, and K. Larsen. Amortization results for chromatic search trees, with an application to priority queues. *Journal of Computer and System Sciences*, 55(3):504{521, 1997.

[3] M. Brown and R. Tarjan. Design and analysis of a data structure for representing sorted lists. *SIAM Journal of Computing*, 9(3):594{614, 1980.

[4] C. Faloutsos and S. Christodoulakis. Design of a signature le method that accounts for non-uniform occurrence and query frequencies. In *Proc. Intern. Conf. on Very Large Data Bases*, pages 165{180, 1985.

[5] C. Faloutsos and H.V. Jagadish. Hybrid text organizations for text databases. In *Proc. Advances in Database Technology*, volume 580 of *LNCS*, pages 310{327, 1992.

[6] L.J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th IEEE Symposium on Foundations of Computer Science*, pages 8{21, 1978.

[7] S.D. Lang, J.R. Driscoll, and J.H. Jou. Batch insertion for tree-structured le organizations { improving di erential database representation. *Information Systems*, 11(2):167{175, 1992.

[8] K. Larsen. Amortized constant relaxed rebalancing using standard rotations. *Acta Informatica*, 35(10):859{874, 1998.

[9] K. Larsen, E. Soisalon-Soininen, and P. Widmayer. Relaxed balance through standard rotations. In *Proc. 5th Workshop on Algorithms and Data Structures*, volume 1272 of *LNCS*, pages 450{461, August 1997.

[10] L. Malmi and E. Soisalon-Soininen. Group updates for relaxed height-balanced trees. In *Proc. 18th ACM Symposium on the Principles of Database Systems*, pages 358{367, 1999.

[11] C. Mohan and I. Narang. Algorithms for creating indexes for very large tables without quiescing updates. In *Proc. 19th ACM SIGMOD Conf. on the Management of Data*, pages 361{370, 1992.

[12] O. Nurmi and E. Soisalon-Soininen. Chromatic binary search trees: A structure for concurrent rebalancing. *Acta Informatica*, 33:547{557, 1996.

[13] K. Pollari-Malmi, E. Soisalon-Soininen, and T. Ylönen. Concurrency control in B-trees with batch updates. *IEEE Transactions on Knowledge and Data Enineering*, 8(6):975{983, 1996.

[14] M. Rossi. Concurrent full text database. Master's thesis, Department of Computer Science, Helsinki University of Technology, Department of Computer Science and Engineering, Finland, 1997.

[15] E. Soisalon-Soininen and P. Widmayer. Concurrency and recovery in full-text indexing. In *Proc. Symposium on String Processing and Information Retrieval (SPIRE'99)*, pages 192{198. IEEE Computer Society, 1999.

[16] V. Srinivasan and M.J. Carey. Performance of on-line index construction algorithms. In *Proc. Advances in Database Technology*, volume 580 of *LNCS*, pages 293{309, 1992.

Approximating SVP_∞ to within Almost-Polynomial Factors Is NP-Hard

Irit Dinur

Tel-Aviv University
dinur@math.tau.ac.il

Abstract. We show SVP_∞ and CVP_∞ to be NP-hard to approximate to within $n^{c/\log\log n}$ for some constant $c > 0$. We show a direct reduction from SAT to these problems, that combines ideas from [ABSS93] and from [DKRS99], along with some modifications. Our result is obtained without relying on the PCP characterization of NP, although some of our techniques are derived from the proof of the PCP characterization itself [DFK+99].

1 Introduction

Background

A lattice $L = L(v_1, \ldots, v_n)$, for linearly independent vectors $v_1, \ldots, v_n \in R^k$ is the additive group generated by the basis vectors, i.e. the set $L = \{\sum a_i v_i \mid a_i \in \mathbb{Z}\}$. Given L, the Shortest Vector Problem (SVP_p) is to find the shortest non-zero vector in L. The length is measured in Euclidean l_p norm ($1 \le p \le \infty$). The Closest Vector Problem (CVP_p) is the non-homogeneous analog, i.e. given L and a vector y, find a vector in L, closest to y.

These lattice problems have been introduced in the previous century, and have been studied since. Minkowsky and Dirichlet tried, with little success, to come up with approximation algorithms for these problems. It was much later that the lattice reduction algorithm was presented by Lenstra, Lenstra and Lovasz [LLL82] , achieving a polynomial-time algorithm approximating the Shortest Lattice Vector to within the exponential factor $2^{n/2}$, where n is the dimension of the lattice. Babai [Bab86] applied LLL's methods to present an algorithm that approximates CVP to within a similar factor. Schnorr [Sch85] improved on LLL's technique, reducing the factor of approximation to $(1 + \varepsilon)^n$, for any constant $\varepsilon > 0$, for both CVP and SVP. These positive approximation results hold for l_p norm for any $p \ge 1$ yet are quite weak, achieving only extremely large (exponential) approximation factors. The shortest vector problem is particularly important, quoting [ABSS93], because even the above relatively weak approximation algorithms have been used in a host of applications, including integer programming, solving low-density subset-sum problems and breaking knapsack based codes [LO85], simultaneous diophantine approximation and factoring polynomials over the rationals [LLL82], and strongly polynomial-time algorithms in combinatorial optimization [FT85].

G. Bongiovanni, G. Gambosi, R. Petreschi (Eds.): CIAC 2000, LNCS 1767, pp. 263–276, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Interest in lattice problems has been recently renewed due to a result of Ajtai [Ajt96], showing a reduction, from a version of SVP, to the *average-case* of the same problem.

Only recently [Ajt98] showed a randomized reduction from the NP-complete problem Subset-Sum to SVP. This has been improved [CN98], showing approximation hardness for some small factor $(1 + \frac{1}{n^{\varepsilon}})$. Very recently [Mic98] has significantly strengthened Ajtai's result, showing SVP hard to approximate to within some constant factor.

The above results all apply to SVP$_p$, for finite p. SVP with the maximum norm l_∞, appears to be a harder problem. A g-approximation algorithm for SVP$_2$ implies a $\sqrt{n}g$-approximation algorithm for SVP$_\infty$, since for every vector v, $\|v\|_\infty \le \|v\|_2 \le \|v\|_\infty \cdot \sqrt{n}$. Thus hardness for approximating SVP$_\infty$ to within a factor $\sqrt{n}g$ will imply the hardness for approximating SVP$_2$ to within factor g. Lagarias showed SVP$_\infty$ to be NP-hard in its exact decision version. Arora et al. [ABSS93] utilized the PCP characterization of NP to show that both CVP (for l_p norm for any p) and SVP$_\infty$ are quasi-NP-hard to approximate to within $2^{(\log n)^{1-\varepsilon}}$ for any constant $\varepsilon > 0$. Recently, the hardness result for approximating CVP has been strengthened [DKS98, DKRS99] showing that it is NP-hard to approximate to within a factor of $n^{\Omega(1)/\log\log n}$ (where n is the lattice dimension). In this paper we similarly strengthen the hardness result for approximating SVP$_\infty$.

So far there is still a huge gap between the positive results, showing approximations for SVP and CVP with exponential factors, and the above hardness results. Nevertheless, some other results provide a discouraging indication for improving the hardness result beyond a certain factor. [GG98] showed that approximating both SVP$_2$ and CVP$_2$ to within $\sqrt{n}$ and approximating SVP$_\infty$ and CVP$_\infty$ to within $n/O(\log n)$ is in NP $\cap$ co-AM. Hence it is unlikely for any of these problems to be NP-hard.

Our Result

We prove that approximating SVP$_\infty$ and CVP$_\infty$ to within a factor of $n^{c/\log\log n}$ is NP-hard (where n is the lattice dimension and $c > 0$ is some arbitrary constant).

Technique

We obtain our result by modifying (and slightly simplifying) the framework of [DKS98, DKRS99]. Starting out from SAT, we construct a new SAT instance that has the additional property that it is either totally satisfiable, or, not even weakly-satisfiable in some specific sense (to be elaborated upon below). We refer to such a SAT instance as an $SSAT_\infty$ instance (this is a variant of [DKS98]'s $SSAT$). The construction reducing SAT to $SSAT_\infty$ is the main part of the paper. The construction has a tree-like recursive structure that is a simplification of techniques from [DKS98, DKRS99], along with some additional observations tailored to the l_∞ norm.

We finally obtain our result by reducing $SSAT_\infty$ to SVP$_\infty$ and to CVP$_\infty$. These reductions are relatively simple combinatorial reductions, utilizing an additional idea from [ABSS93].

Hardness-of-approximation results are naturally divided into those that are obtained via reduction from PCP, and those that are not. Although the best previous hardness result for SVP$_\infty$ [ABSS93] relies on the PCP characterization of NP, our proof does not. We do, however, utilize some techniques similar to those used in the proof of the PCP characterization of NP itself. In fact, the nature of the SVP$_\infty$ problem eliminates some of the technical complications from [DFK$^+$99, DKS98, DKRS99]. Thus, we believe that SVP$_\infty$ makes a good candidate (out of all of the lattice problems) for pushing the hardness-of-approximation factor to within polynomial range.

Structure of the Paper

Section 2 presents a variant of the $SSAT$ problem from [DKS98] which we call $SSAT_\infty$. It then proceeds with some standard (and not so standard) definitions. Section 3 gives the reduction from SAT to $SSAT_\infty$, whose correctness is proven in Section 4. Finally, in Section 5 we describe the (simple) reduction from $SSAT_\infty$ to SVP$_\infty$ and to CVP$_\infty$, establishing the hardness of approximating SVP$_\infty$ and CVP$_\infty$.

2 Definitions

2.1 $SSAT_\infty$

A SAT instance is a set $\Psi = \{\varphi_1, \ldots, \varphi_n\}$ of *tests* (Boolean functions) over variables $V = \{v_1, \ldots, v_m\}$. We denote by R_{φ_i} the set of satisfying assignments for $\varphi_i \in \Psi$. The Cook-Levin [Coo71, Lev73] theorem states that it is NP-hard to distinguish whether or not the system is satisfiable (i.e. whether there is an assignment to the variables that satisfies all of the tests). We next define $SSAT_\infty$, a version of SAT that has the additional property that when the instance is not satisfiable, it is not even 'weakly-satisfiable' in a sense that will be formally defined below.

We recall the following definitions (Definitions 1,2 and 3) from [DKS98],

Definition 1 (Super-Assignment to Tests). *A super-assignment is a function S mapping to each $\varphi \in \Psi$ a value from $\mathbb{Z}^{R_\varphi}$. $S(\varphi)$ is a vector of integer coefficients, one for each value $r \in R_\varphi$. Denote by $S(\varphi)[r]$ the r^{th} coordinate of $S(\varphi)$.*

If $S(\varphi) = \vec{0}$ we say that $S(\varphi)$ is trivial. If $S(\varphi)[r] \neq 0$, we say that the value r appears in $S(\varphi)$. A *natural assignment* (an assignment in the usual sense) is identified with a super-assignment that assigns each $\varphi \in \Psi$ a unit vector with a 1 in the corresponding coordinate. In this case, exactly one value appears in each $S(\varphi)$.

We next define the projection of a super-assignment to a test onto each of its variables. Consistency between tests will amount to equality of projections on mutual variables.

Definition 2 (Projection). *Let S be a super-assignment to the tests. We define the projection of $S(\varphi)$ on a variable x of φ, $\pi_x(S(\varphi)) \in \mathbb{Z}^{|\mathcal{F}|}$, in the natural way:*

$$\forall a \in \mathcal{F} : \qquad \pi_x(S(\varphi))[a] \overset{def}{=} \sum_{r \in R_\varphi \,;\, r|_x = a} S(\varphi)[r]$$

We shall now proceed to define the notion of consistency between tests. If the projections of two tests on each mutual variable x are equal (in other words, they both give x the same super-assignment), we say that the super-assignments of the tests are consistent (match).

Definition 3 (Consistency). *Let S be a super-assignment to the tests in Ψ. S is consistent if for every pair of tests φ_i and φ_j with a mutual variable x,*

$$\pi_x(S(\varphi_i)) = \pi_x(S(\varphi_j))$$

Given a system $\Psi = \{\varphi_1, \ldots, \varphi_n\}$, a super-assignment $S : \Psi \to \mathbb{Z}^R$ is called *not-all-zero* if there is at least one test $\varphi \in \Psi$ for which $S(\varphi) \neq \bar{0}$. The *norm* of a super-assignment S is defined

$$\|S\| \overset{def}{=} \max_{\varphi \in \Psi} (\|S(\varphi)\|_1)$$

where $\|S(\varphi)\|_1$ is the standard l_1 norm. The norm of a natural super-assignment is 1.

The gap of $SSAT_\infty$ is formulated in terms of the norm of the minimal super-assignment that maintains consistency.

Definition 4 (g-$SSAT_\infty$). *An instance of $SSAT_\infty$ with parameter g*

$$I = \langle \Psi = \{\varphi_1, \ldots, \varphi_n\}, V = \{v_1, \ldots, v_m\}, \{R_{\varphi_1}, \ldots, R_{\varphi_n}\} \rangle$$

consists of a set Ψ of tests *over a common set V of variables that take values in a field $\mathcal{F}$. The parameters m and $|\mathcal{F}|$ are always bounded by some polynomial in n. Each test $\varphi \in \Psi$ has associated with it a list R_φ of assignments to its variables, called the* satisfying assignments *or the* range *of the test φ. The problem is to distinguish between the following two cases,*

Yes: There is a consistent natural assignment for Ψ.
No: No not-all-zero consistent super-assignment is of norm $> g$.

Remark. The definition of $SSAT_\infty$ differs from that of $SSAT$ only in the characterization of when a super-assignment falls into the 'no' category. On one hand, $SSAT_\infty$ imposes a weaker requirement of not-all-zero rather than the non-triviality of $SSAT$. On the other hand, the norm of a super assignment S is measured by a 'stronger' measure, taking the maximum of $\|S(\varphi)\|_1$ over all φ, rather than the average as in $SSAT$.

Theorem 1 ($SSAT_\infty$ Theorem). *$SSAT_\infty$ is NP-hard for $g = n^{c/\log\log n}$ for some $c > 0$.*

We conjecture that a stronger statement is true, which would imply that SVP$_\infty$ NP-hard to approximate to within a *constant power* of the dimension.

Conjecture 2 *$SSAT_\infty$ is NP-hard for $g = n^c$ for some constant $c > 0$.*

2.2 LDFs, Super-LDFs

Throughout the paper, let F denote a finite field $F = \mathbb{Z}_p$ for some prime number $p > 1$. We will need the following definitions.

Definition 5 (low degree function - $[r,d]$-LDF). *A function $f : F^d \to F$ is said to have degree r if its values are the point evaluation of a polynomial on F^d with degree $\leq r$ in each variable. In this case we say that f is an $[r,d]$-LDF, or $f \in LDF_{r,d}$.*

Sometimes we omit the parameters and refer simply to an LDF.

Definition 6 (low degree extension). *Let m, d be natural numbers, and let $H \subset F$ such that $|H|^d = m$. A vector $(a_0, \ldots, a_{m-1}) \in F^m$ can be naturally identified with a function $A : H^d \to F$ by looking at points in H^d as representing numbers in base $|H|$.*

There exists exactly one $[|H| - 1, d]$-LDF $\hat{A} : F^d \to F$ that extends A. $\hat{A}$ is called the $|H| - 1$ degree extension of A in F.

A $(D+2)$-dimensional affine subspace ($(D+2)$-*cube* for short) $C \subset F^d$ is said to be *parallel* to the axises if it can be written as $C = x + \text{span}(e_{i_1}, \ldots, e_{i_{D+2}})$, where $x \in F^d$ and $e_i \in F^d$ is the i-th axis vector, $e_i = (0, \ldots, 1, \ldots, 0)$. We write the parameterization of the cube C as follows,

$$C(z) \stackrel{def}{=} x + \sum_{j=1}^{D+2} z_j e_{i_j} \in F^d \qquad \text{for } z = (z_1, \ldots, z_{D+2}) \in F^{D+2}$$

We will need the following (simple) proposition,

Proposition 1. *Let $f : F^d \to F$. Suppose, for every parallel $(D+2)$-cube $C \subset F^d$ the function $f|_C : F^{D+2} \to F$ defined by*

$$\forall x \in F^{D+2} \quad f|_C(x) = f(C(x))$$

is an $[r, D+2]$-LDF. Then f is an $[r,d]$-LDF.

Similar to the definition of super-assignments, we define a *super-$[r,d]$-LDF* (or a super-LDF for short) $G \in \mathbb{Z}^{\text{LDF}_{r,d}}$ to be a vector of integer coefficient $G[P]$ per LDF $P \in \text{LDF}_{r,d}$. This definition arises naturally from the fact that the tests in our final construction will range over LDFs. We further define the *norm* of a super-LDF to be the l_1 norm of the corresponding coefficient vector.

We say that an LDF $P \in \mathrm{LDF}_{r;d}$ *appears* in G iff $G[P] \neq 0$. A point x is called *ambiguous* for a super-LDF G, if there are two LDFs P_1, P_2 appearing in G such that $P_1(x) = P_2(x)$. The following (simple) property of *low-norm* super-LDFs is heavily used in this paper.

Proposition 2 (Low Ambiguity). *Let G be a super-$[r;d]$-LDF of norm $\leq g$. The fraction of ambiguous points for G is $\leq \mathrm{amb}(r;d;g) \stackrel{def}{=} \binom{g}{2} \frac{rd}{|F|}$.*

Proof. The number of non-zero coordinates in an integer vector whose l_1 norm is g is $\leq g$. There are $\leq \binom{g}{2}$ pairs of LDFs appearing in G, and each pair agrees on at most $\frac{rd}{|F|}$ of the points in F^d.

The following embedding-extension technique taken from [DFK⁺99] is used in our construction,

Definition 7 (embedding extension). *Let $b \geq 2$, $k > 1$ and t be natural numbers. We define the* embedding extension mapping $E_b : F^t \to F^{t \cdot k}$ *as follows. E_b maps any point $x = (\xi_1, \ldots, \xi_t) \in F^t$ to $y \in F^{t \cdot k}$, $y = E_b(x) = (\eta_1, \ldots, \eta_{t \cdot k})$ by*

$$E_b(\xi_1, \ldots, \xi_t) \stackrel{def}{=} \left(\xi_1, (\xi_1)^b, (\xi_1)^{b^2}, \ldots, (\xi_1)^{b^{k-1}}, \ldots \ldots, \xi_t, (\xi_t)^b, (\xi_t)^{b^2}, \ldots, (\xi_t)^{b^{k-1}}\right)$$

The following (simple) proposition, shows that any LDF on F^t can be represented by an LDF on $F^{t \cdot k}$ with significantly lower degree:

Proposition 3. *Let $f : F^t \to F$ be a $[b^k - 1; t]$-LDF, for integers $t > 0, b > 1, k > 1$. There is a $[b - 1; t \cdot k]$-LDF $f_{ext} : F^{t \cdot k} \to F$ such that*

$$\forall x \in F^t : \quad f(x) = f_{ext}(E_b(x))$$

For any $[b - 1; kt]$-LDF f, its 'restriction' to the manifold $f|_{E_b} : F^t \to F$ is defined as

$$\forall x \in F^t \quad f|_{E_b}(x) \stackrel{def}{=} f(E_b(x))$$

and is a $[b^k - 1; t]$-LDF (the degree in a variable ξ_i of $f|_{E_b}$ is $(b - 1)(b^0 + b^1 + \ldots + b^{k-1}) = b^k - 1$).

Let $\mathcal{G}$ be a super-$[b^k - 1; t]$-LDF (i.e. a vector in $\mathbb{Z}^{\mathrm{LDF}_{r;t}}$). Its *embedding-extension* is the super-$[b - 1; tk]$-LDF G defined by,

$$\forall f \in \mathrm{LDF}_{b-1;tk} \quad G[f] \stackrel{def}{=} \mathcal{G}[f|_{E_b}]$$

In a similar manner, the *restriction* $\mathcal{G}$ of a super-$[b - 1; tk]$-LDF G is a super-$[b^k - 1; t]$-LDF defined by

$$\forall f \in \mathrm{LDF}_{b^k-1;t} \quad G[f] \stackrel{def}{=} \mathcal{G}[f_{ext}]$$

The following proposition holds (e.g. by a counting argument),

Proposition 4. *Let G_1, G_2 be two super-$[b-1, tk]$-LDFs, and let $\tilde{G}_1, \tilde{G}_2$ be their respective restrictions (with parameter b). $\tilde{G}_1 = \tilde{G}_2$ if and only if $G_1 = G_2$.*

3 The Construction

We prove that $SSAT_\infty$ is NP-hard via a reduction from SAT, described herein. We adopt the whole framework of the construction from [DKRS99], and refer the reader there for a more detailed exposition.

Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be an instance of SAT, viewed as a set of Boolean *tests* over Boolean variables $V = \{x_1, \ldots, x_m\}$, ($m = n^c$ for some constant $c > 0$) such that each test depends on $D = O(1)$ variables. Cook's theorem [Coo71] states that it is NP-hard to decide whether there is an assignment for V satisfying all of the tests in Φ.

Starting from Φ, we shall construct an $SSAT_\infty$ test-system Ψ over variables $V_\Psi \supseteq V$. Our new variables V_Ψ will be non-Boolean, ranging over a field F, with $|F| = n^{c/\log\log n}$ for some constant $c > 0$. An assignment to V_Ψ will be interpreted as an assignment to V by identifying the value $0 \in F$ with the Boolean value true and any other non-zero value with false.

3.1 Constructing the CR-Forest

In order to construct the $SSAT_\infty$ instance $I = \langle \Psi, V, \{R_{\psi_1}, \ldots, R_{\psi_n}\}\rangle$ we need to describe for each test $\psi \in \Psi$, which variables it depends on, and its satisfying assignments R_ψ. We begin by constructing the CR-forest, which is a combinatorial object holding the underlying structure of Ψ. The forest $\mathcal{F}_n(\Phi)$ will have a tree $\mathcal{T}_\varphi$ for every test $\varphi \in \Phi$. Each node in the forest will have a set of variables associated with it. For every leaf there will be one test depending on the variables associated with that leaf.

Let us (briefly) describe one tree $\mathcal{T}_\varphi$ in the forest $\mathcal{F}_n(\Phi)$.

Every tree will be of depth $K \approx \log\log n$ (however, not all of the leaves will be at the bottom level).

Each node v in the tree will have a domain $\mathbf{dom}_v = F^d$ of points ($\mathbf{dom}_v = F^{d_0}$ in case v is the root node) associated with it.

The offsprings of a non-leaf node v will be labeled each by a distinct $(D+2)$-cube C_v of $\mathbf{dom}_v$ (this part is slightly simpler than in [DKRS99]),

$$\mathrm{labels}(v) \overset{def}{=} \{C \mid C \text{ is a } (D+2)\text{-cube in } \mathbf{dom}_v\}.$$

The points in the domain $\mathbf{dom}_v$ of each node v will be mapped to some of Ψ's variables, by the injection $\mathrm{var}_v : \mathbf{dom}_v \to V_\Psi$. This mapping essentially describes the relation of a node to its parent, and is defined inductively as follows. For each node v, we denote by V_v the set of 'fresh new' variables mapped from

dom_v (i.e. none of the nodes defined inductively so far have points mapped to these variables). Altogether

$$V \overset{def}{=} V_\Psi = \bigcup_{\substack{v \in \mathcal{T}_\varphi \\ \varphi \in \Psi}} V_v .$$

For the root node, $\text{var}_{root_\varphi} : \text{dom}_{root_\varphi} \to V_\Psi$ is defined (exactly as in [DKRS99]) by mapping $H^{d_0} \subseteq \text{dom}_{root_\varphi} = F^{d_0}$ to V and the rest of the points to the rest of $V_{root_\varphi} \overset{def}{=} \hat{V} \subseteq V_\Psi$ (i.e. the low-degree-extension of V). It is important that $\text{var}_{root_\varphi}$ is defined independently of φ.

For a non-root node v with parent u, the points of the cube $C_v \in \text{labels}(u)$ labeling v are mapped into the domain dom_v by the embedding extension mapping, $E_{b_v} : C_v \to \text{dom}_v$, defined above in Section 2.2 (the parameter b_v specified below depends on the specific node v, rather than just on v's level as in [DKRS99]). These points are u's points that are 'passed on' to the offspring v. We think of the point $y = E_{b_v}(x) \in \text{dom}_v$ as 'representing' the point $x \in C_v \subseteq \text{dom}_u$, and define $\text{var}_v : \text{dom}_v \to V_\Psi$ as follows,

Definition 8 (var_v, for a non-root node v). *Let v be a non-root node, let u be v's parent, and let $C_v \subseteq \text{dom}_u$ be the label attached to v. For each point $y \in E_{b_v}(C_v) \subseteq \text{dom}_v$ define $\text{var}_v(y) \overset{def}{=} \text{var}_u(E_{b_v}^{-1}(y))$, i.e. points that 'originated' from C_v are mapped to the previous-level variables, that their pre-images in C_v were mapped to. For each 'new' point $y \in \text{dom}_v \setminus E_{b_v}(C_v)$ we define $\text{var}_v(y)$ to be a distinct variable from V_v.*

The parameters used for the embedding extension mappings E_{b_v} are $t = D + 2$, $k = d/t = a$. We set the degree of the root node $r_{root_\varphi} = |H| = |F|^{1/10}$ and r_v and b_v (for non-root nodes v) are defined by the following recursive formulas:

$$b_v = \begin{cases} \sqrt[a]{r_u + 1} & C_v \text{ is parallel to the axises} \\ \sqrt[a]{r_u(D + 2) + 1} & \text{Otherwise} \end{cases}$$

$$r_v = b_v - 1$$

We stop the recursion and define a node to be a leaf (i.e. define its labels to be empty) whenever $r_v \le 2(D + 2)$. A simple calculation (to appear in the complete version) shows that b_v, r_v decrease with the level of v until for some level $K < \log\log n$, $r_v \le 2(D + 2) = O(1)$. (This may happen to some nodes sooner than others, therefore not all of the leaves are in level K).

We now complete the construction by describing the tests and their satisfying assignments.

Definition 9 (Tests). *Ψ will have one test ψ_v for each leaf v in the forest. ψ_v will depend on the variables in $\text{var}_v(\text{dom}_v)$. The set of satisfying assignments for ψ_v's variables, R_{ψ_v}, will consist of assignments A that satisfy the following two conditions:*

1. *A is an $[r_v; d]$-LDF on* $\mathrm{var}_v(\mathrm{dom}_v)$
2. *If v 2 $\mathrm{T}_{'}$ for ' 2 and '’s variables appear in* $\mathrm{var}_v(\mathrm{dom}_v)$, *then A must satisfy '.*

4 Correctness of the Construction

4.1 Completeness

Lemma 1 (completeness). *If there is an assignment $A : V\ \ !\ f\mathtt{true}; \mathtt{false}g$ satisfying all of the tests in , then there is a natural assignment $A_\Psi : V_\Psi\ \ !\ F$ satisfying all of the tests in Ψ.*

We extend A in the obvious manner, i.e. by taking its low-degree-extension (see De nition 6) to the variables $\hat{V}$, and then repeatedly taking the embedding extension of the previous-level variables, until we’ve assigned all of the variables in the system. More formally,

Proof. We construct an assignment $A_\Psi : V_\Psi\ \ !\ F$ by inductively obtaining $[r_i; d]$-LDFs $P_v : \mathrm{dom}_v\ \ !\ F$ for every level-i node v of every tree in the CR-forest, as follows. We rst set (for every ' 2) $P_{root_'}$ to be the low degree extension (see De nition 6) of A (we think of A as assigning each variable a value in $f0; 1g$ F rather than $f\mathtt{true}; \mathtt{false}g$, see discussion in the beginning of Section 3). Assume we’ve de ned an $[r_u; d]$-LDF P_u consistently for all level-i nodes, and let v be an o spring of u, labeled by C_v. The restriction $f = P_u j_{C_v}$ of P_u to the cube C_u is an $[r; D+2]$-LDF where $r = r_u$ or $r = r_u(D+2)$ depending on whether C_v is parallel to the axises or not. f can be written as a $[\sqrt[a]{r+1} - 1;\ a\ (D+2)]$-LDF f_{ext} over the larger domain F^d, as promised by Proposition 3. We de ne $P_v = f_{\mathbf{ext}}$ to be that $[r_v; d]$-LDF (recall that $d = a\ (D+2)$ and $b_v = \sqrt[a]{r+1}$).

Finally, for a variable x 2 var_v, x = $\mathrm{var}_v(x)$, we set $A_\Psi(\mathrm{x}) \overset{def}{=} P_v(x)$. The construction implies that there are no collisions, i.e. $\mathrm{x}^0 = \mathrm{var}_{v^0}(x^0) = \mathrm{var}_v(x) =$ x implies $P_v(x) = P_{v^0}(x^0)$. ◻

4.2 Soundness

We need to show that a ’no’ instance of SAT is mapped to a ’no’ instance of $SSAT_1$. We assume that the constructed $SSAT_1$ instance has a consistent non-trivial super-assignment of norm g, and show that { the SAT instance we started with { is satis able.

Lemma 2 (Soundness). *Let $g \overset{def}{=} jFj^{\frac{1}{102}}$. If there exists a consistent super-assignment of norm g for Ψ, then is satis able.*

Let A be a consistent non-trivial super-assignment for Ψ, of size kAk_1 g. It induces (by projection) a super-assignment to the variables

$$m : V_\Psi\ -!\ \mathbb{Z}^{jFj}$$

i.e. for every variable x 2 V_ψ, m assigns a vector $_x(A($ $))$ of integer coe cients, one per value in F where is some test depending on x. Since A is consistent, m is well de ned (independent of the choice of test). Alternatively, we view m as a labeling of the points $\bigcup_{v 2 \mathbf{F}_n( )} \mathrm{dom}_v$ by a 'super-value' { a formal linear combination of values from F. The label of the point x 2 dom_v for some v 2 $\mathrm{F}_n($ $)$, is simply $m(\mathrm{var}_v(x))$, and with a slight abuse of notation, is sometimes denoted $m(x)$. m is used as the \underlying point super-assignment" for the rest of the proof, and will serve as an anchor by which we test consistency.

The central task of our proof is to show that if a tree has a non-trivial leaf, then there is a non-trivial super-LDF for the domain in the root node that is consistent with m. We will later want to construct from these super-LDFs an assignment that satis es all of the tests in . For this purpose, we need the super-LDFs along the way to be legal,

De nition 10 (legal). *An LDF P is called* legal for a node v 2 $\mathrm{T}_{'}$ *(for some ' 2), if it satis es ' in the sense that if ''s variables have pre-images $x_1; ::: ; x_D$ 2 dom_v, then $P(x_1); ::: ; P(x_D)$ satisfy '. A super-LDF G is called* legal for v 2 $\mathrm{T}_{'}$ *if for every LDF P appearing in G, P is legal for v 2 $\mathrm{T}_{'}$.*

The following lemma encapsulates the key inductive step in our soundness proof,

Lemma 3. *Let u 2 nodes_i for some 0 $i < K$. There is a legal super-$[r_u; d]$-LDF G_u with kG_uk_1 $kmk_1 \overset{def}{=} \max_x km(x)k_1$ such that for every x 2 dom_u, $_x(G_u) = m(x)$. Furthermore, if there is a node v in u's sub-tree for which $G_v \not\equiv 0$ then $G_u \not\equiv 0$.*

Due to space limitations, the proof of this lemma is omitted, and appears in the full version of this paper.

In order to complete the soundness proof, we need to nd a satisfying assignment for . We obtained, in Lemma 3, a super-$[r_0; d]$-LDF $G_{'}$ for each root node $root_{'}$, such that $8x$ 2 $\mathrm{dom}_{root_{'}} = F^{d_0}$, $m(x) =$ $_x(G_{'})$. Note that indeed, for every pair of tests $' \not\equiv '^0$, the corresponding super-LDFs must be equal $G_{'} = G_{'^0}$ (denote them by G). This follows because they are point-wise equal $_x(G_{'}) = m(x) =$ $_x(G_{'^0})$, and so the di erence super-LDF $G_{'} - glob_{'^0}$ is trivial on every point, and must therefore (again, by Proposition 2 { low-ambiguity) be trivial.

If A is not trivial, then there is at least one test $_v$ 2 Ψ for which $A($ $) \not\equiv 0$. Thus, denoting by ' the test for which v is a leaf in $\mathrm{T}_{'}$, Lemma 3 implies $G = G_{'} \not\equiv 0$. Take an LDF f that appears in G, and de ne for every v 2 V , $A(v) \overset{def}{=} f(x)$ where x 2 H^{d_0} is the point mapped to v. Since G is legal, is totally satis ed by A.

5 From $SSAT_\infty$ to SVP_∞

In this section we show the reduction from $g\text{-}SSAT_1$ to the problem of approximating SVP_1. This reduction follows the same lines of the reduction in

[ABSS93] from Pseudo-Label-Cover to SVP_∞. We begin by formally defining the gap-version of SVP_∞ (presented in Section 1) which is the standard method to turn an approximation problem into a decision problem.

Definition 11 (g-SVP_∞). *Given a lattice L and a number $d > 0$, distinguish between the following two cases:*

Yes. There is a non-zero lattice vector $v \in L$ with $\|v\|_\infty \le d$.
No. Every non-zero lattice vector $v \in L$ has $\|v\|_\infty > g \cdot d$.

We will show a reduction from g-$SSAT_\infty$ to $\sqrt{g}$-SVP_∞, thereby implying SVP_∞ to be NP-hard to approximate to within a factor of $\sqrt{g} = n^{\Omega(1)/\log\log n}$.

Let $I = \langle \Psi, V, \{R_\psi\} \rangle$ be an instance of g-$SSAT_\infty$, where $\Psi = \{\psi_1, \ldots, \psi_n\}$ is a set of tests over variables $V = \{v_1, \ldots, v_m\}$, and R_{ψ_i} is the set of satisfying assignments for $\psi_i \in \Psi$. We construct a $\sqrt{g}$-SVP_∞ instance $(L(B), d)$ where $d \overset{def}{=} 1$ and B is an integer matrix whose columns form the basis for the lattice $L(B)$.

The matrix B will have a column $v_{[\psi,r]}$ for every pair of test $\psi \in \Psi$ and an assignment $r \in R_\psi$ for it. There will be one additional special column t. The matrix B will have two kinds of rows, consistency rows and norm-measuring rows, defined as follows.

Consistency Rows. B will have $|F| + 1$ rows for each threesome (ψ_i, ψ_j, x) where ψ_i and ψ_j are tests that depend on a mutual variable x. Only the columns of ψ_i and ψ_j will have non-zero values in these rows.

The special column t will have $\sqrt{g}$ in each consistency row, and zero in the other rows.

For a pair of tests ψ_i and ψ_j that depend on a mutual variable x, let's concentrate on the sub-matrix consisting of the columns of these tests, and the $|F| + 1$ rows of the pair ψ_i, ψ_j viewed as a pair of matrices G_1 of dimension $(|F| + 1) \times |R_{\psi_i}|$ and G_2 of dimension $(|F| + 1) \times |R_{\psi_j}|$. Let $r \in R_{\psi_i}$ be a satisfying assignment for ψ_i and $r' \in R_{\psi_j}$ be a satisfying assignment for ψ_j. The r-th column in G_1 equals $\sqrt{g}$ times the unit vector e_i where $i = r|_x$ (i.e. a vector with zeros everywhere and a $\sqrt{g}$ in the $r|_x$-th coordinate). The r'-th column in G_2 equals $\sqrt{g} \cdot (1 - e_i)$ where $i = r'|_x$ and 1 is the all-one vector (i.e. $\sqrt{g}$ everywhere except a zero in the $r'|_x$-th coordinate).

Notice that any zero-sum linear combination of the vectors $\{e_i, 1 - e_i, 1\}_i$ must give e_i the same coefficient as $1 - e_i$, because the vectors $\{1, e_i\}_i$ are linearly independent.

Norm-measuring Rows. There will be a set of $|R_\psi|$ rows designated to each test $\psi \in \Psi$ in which only ψ's columns have non-zero values. The matrix B, when restricted to these rows and to the columns of ψ, will be the $(|R_\psi| \times |R_\psi|)$ Hadamard matrix H (we assume for simplicity that $|R_\psi|$ is a power of 2, thus such a matrix exists, see [Bol86] p. 74). Recall that the Hadamard matrix H_n of order $2^n \times 2^n$ is defined by $\mathrm{H}_0 = (1)$ and $\mathrm{H}_n = \begin{pmatrix} \mathrm{H}_{n-1} & \mathrm{H}_{n-1} \\ \mathrm{H}_{n-1} & -\mathrm{H}_{n-1} \end{pmatrix}$.

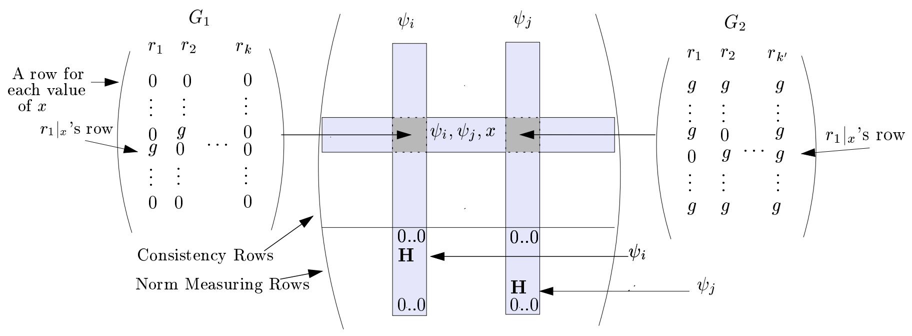

Fig. 1. The matrix B

The vector t, as mentioned earlier, will be zero on these rows.

Proposition 5 (Completeness). *If there is a natural assignment to Ψ, then there is a non-zero lattice vector $v \in L(B)$ with $\|v\|_\infty = 1$.*

Proof. Let A be a consistent natural assignment for Ψ. We claim that

$$v = t - \sum_{\psi \in \Psi} v_{[\psi, A(\psi)]}$$

is a lattice vector with $\|v\|_\infty = 1$. Restricting $\sum_{\psi \in \Psi} v_{[\psi, A(\psi)]}$ to an arbitrary row in the consistency rows (corresponding to a pair of tests ψ_i, ψ_j with mutual variable x), gives $\sqrt{g}$, because $A(\psi_i)|_x = A(\psi_j)|_x$. Subtracting this from t gives zero in each consistency-row.

In the norm-measuring rows, since every test $\psi \in \Psi$ is assigned one value by A, v restricted to ψ's rows equals some column of the Hadamard matrix which is a ± 1 matrix. Altogether, $\|v\|_\infty = 1$ as claimed. ⊓⊔

Proposition 6 (Soundness). *If there is a non-zero lattice vector $v \in L(B)$ with $\|v\|_\infty < \sqrt{g}$, then there is a consistent non-trivial super-assignment A for Ψ, for which $\|A\|_\infty < g$.*

Proof. Let

$$v = c_t \cdot t + \sum_{\psi, r} c_{[\psi, r]} \cdot v_{[\psi, r]}$$

be a lattice vector with $\|v\|_\infty < \sqrt{g}$. The entries in the consistency rows of every lattice vector, are integer multiples of $\sqrt{g}$. The assumption $\|v\|_\infty < \sqrt{g}$ implies that v is zero on these rows.

Define a super-assignment A to Ψ by setting for each $\psi \in \Psi$ and $r \in R_\psi$, $A(\psi)[r] \overset{def}{=} c_{[\psi, r]}$.

To see that A is consistent, let $_i, _j \in \Psi$ both depend on the variable x. Notice that (as mentioned above) any zero-sum linear combination of the vectors $\{1, e_k, 1 - e_k\}_k$ must give e_k and $1 - e_k$ the same coe cient because the vectors $\{1, e_k\}_k$ are linearly independent. This implies that for any value $k \in F$ for x,

$$\sum_{r|_x = k} c_{[\ _i, r]} = \sum_{r'|_x = k} c_{[\ _j, r']}$$

This, in turn, means that $_x(A(\ _i)) = \ _x(A(\ _j))$ thus A is consistent.

A is also not-all-zero because $v \neq 0$ (if only c_t was non-zero, then $\|v\|_\infty = \sqrt{g}$). The norm of A is de ned as

$$\|A\|_\infty = \max_{\ \in \Psi} (\|A(\ )\|_1)$$

The vector v restricted to the norm-measuring rows of is exactly H $A(\ )$. Now since $\frac{1}{\sqrt{|R\ |}}$H is a $(|R\ | \ |R\ |)$ orthonormal matrix, we have

$$\|\frac{1}{\sqrt{|R\ |}} \mathrm{H} A(\ )\|_2 = \|A(\ )\|_2$$

Since for every $z \in \mathbb{R}^n$, $\|z\|_\infty \ \|z\|_2/\sqrt{n}$, we obtain $\|\mathrm{H}A(\ )\|_\infty \ \|A(\ )\|_2$. Now for every integer vector z, $\sqrt{\|z\|_1} \ \|z\|_2$, and altogether,

$$\sqrt{\|A(\ )\|_1} \ \|A(\ )\|_2 \ \|\mathrm{H}A(\ )\|_\infty \ \|v\|_\infty < \sqrt{g}$$

showing $\|A\|_\infty \overset{def}{=} \max_{\ \in\Psi} (\|A(\ )\|_1) < g$ as claimed. ∎

Finally, if Ψ is a $SSAT_\infty$ no instance, then the norm of any consistent super-assignment A must be at least g, and so the norm of the shortest lattice vector in $L(B)$, must be at least g. This completes the proof of the reduction.

The reduction to CVP$_\infty$ is quite similar, taking t to be the target vector, and is omitted.

References

[ABSS93] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 724–733, 1993.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. In *Proc. 28th ACM Symp. on Theory of Computing*, pages 99–108, 1996.

[Ajt98] Miklos Ajtai. The shortest vector problem in *L2* is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 10–19, New York, May 23–26 1998. ACM Press.

[Bab86] L. Babai. On Lovasz's lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–14, 1986.

[Bol86] B. Bollobas. *Combinatorics.* Cambridge University Press, 1986.

[CN98] J.Y. Cai and A. Nerurkar. Approximating the SVP to within a factor (1 + 1=dim$^{"}$) is NP-hard under randomized reductions. In *Proc. of the 13th Annual IEEE Conference on Computational Complexity*, pages 46{ 55. 1998.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151{158, 1971.

[DFK^{+}99] Dinur, Fischer, Kindler, Raz, and Safra. PCP characterizations of NP: Towards a polynomially-small error-probability. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1999.

[DKRS99] I. Dinur, G. Kindler, R. Raz, and S. Safra. Approximating-CVP to within almost-polynomial factors is NP-hard. Manuscript, 1999.

[DKS98] Dinur, Kindler, and Safra. Approximating-CVP to within almost-polynomial factors is NP-hard. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1998.

[FT85] Andras Frank and Eva Tardos. An application of simultaneous approximation in combinatorial optimization. In *26th Annual Symposium on Foundations of Computer Science*, pages 459{463, Portland, Oregon, 21{ 23 October 1985. IEEE.

[GG98] O. Goldreich and S. Goldwasser. On the limits of non-approximability of lattice problems. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 1{9, 1998.

[Lev73] L. Levin. Universal'ny e pereborny e zadachi (universal search problems : in Russian). *Problemy Peredachi Informatsii*, 9(3):265{266, 1973.

[LLL82] A.K. Lenstra, H.W. Lenstra, and L. Lovasz. Factoring polynomials with rational coe cients. *Math. Ann.*, 261:513{534, 1982.

[LO85] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM*, 32(1):229{246, January 1985.

[Mic98] D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proc. 39th IEEE Symp. on Foundations of Computer Science*, 1998.

[Sch85] C.P. Schnorr. A hierarchy of polynomial-time basis reduction algorithms. In *Proceedings of Conference on Algorithms, Pecs (Hungary)*, pages 375{ 386. North-Holland, 1985.

Convergence Analysis of Simulated Annealing-Based Algorithms Solving Flow Shop Scheduling Problems[*]

Kathleen Steinhöfel[1], Andreas Albrecht[2], and Chak-Kuen Wong[2]

[1] GMD – National Research Center for Information Technology,
Kekulestr. 7, D-12489 Berlin, Germany
[2] Dept. of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Abstract. In the paper, we apply logarithmic cooling schedules of inhomogeneous Markov chains to the flow shop scheduling problem with the objective to minimize the makespan. In our detailed convergence analysis, we prove a lower bound of the number of steps which are sufficient to approach an optimum solution with a certain probability. The result is related to the maximum escape depth Γ from local minima of the underlying energy landscape. The number of steps k which are required to approach with probability $1 - \varepsilon$ the minimum value of the makespan is lower bounded by $n^{O(\Gamma)} \cdot \log^{O(1)}(1/\varepsilon)$. The auxiliary computations are of polynomial complexity. Since the model cannot be approximated arbitrarily closely in the general case (unless $P = NP$), the approach might be used to obtain approximation algorithms that work well for the average case.

1 Introduction

In the flow shop scheduling problem n jobs have to be processed on m different machines. Each job consists of a sequence of tasks that have to be processed during an uninterrupted time period of a fixed length. The order in which each job is processed by the machines is the same for all jobs. A schedule is an allocation of the tasks to time intervals on the machines and the aim is to find a schedule that minimizes the overall completion time which is called the makespan.

Flow shop scheduling has long been identified as having a number of important practical applications. Baumgärtel addresses in [4] the flow shop problem in order to deal with the planning of material flow in car plants. His approach was applied to the logistics for the Mercedes Benz automobile. The NP-hardness of the general problem setting with $m \geq 3$ was shown by Garey, Johnson, and Sethi

[*] Research partially supported by the Strategic Research Program at the Chinese University of Hong Kong under Grant No. SRP 9505, by a Hong Kong Government RGC Earmarked Grant, Ref. No. CUHK 4367/99E, and by the HK-Germany Joint Research Scheme under Grant No. D/9800710.

[10] in 1976. The existence of a polynomial approximation scheme for the flow shop scheduling problem with an arbitrary fixed number of machines is demonstrated by Hall in [12]. A recent work of Williamson et al. constitutes theoretical evidence that the general problem, which is considered in the present paper, is hard to solve even approximately. They proved that finding a schedule that is shorter than 5/4 times the optimum is NP-hard [23].

We are concentrating on the convergence analysis of simulated annealing-based algorithms which employ a logarithmic cooling schedule. The algorithm employs a simple neighborhood which is reversible and ensures a priori that transitions always result in a feasible solution. The neighborhood relation determines a landscape of the objective function over the configuration space $\mathcal{F}$ of feasible solutions of a given flow shop scheduling problem. Let $a_S(k)$ denote the probability to obtain the schedule $S \in \mathcal{F}$ after k steps of a logarithmic cooling schedule. The problem is to find an upper bound for k such that $\sum_{S \in \mathcal{F}_{\min}} a_S(k) > 1 - \varepsilon$ for schedules S minimizing the makespan. The general framework of logarithmic cooling schedules has been studied intensely, e.g., by B. Hajek [11] and O. Catoni [5,6].

Our convergence result, i.e., the lower bound of the number of steps k, is based on a very detailed analysis of transition probabilities between neighboring elements of the configuration space $\mathcal{F}$. We obtain a run-time of $n^{O(\Gamma)} \cdot \log^{O(1)}(1/\delta)$ to have with probability $1 - \delta$ a schedule with the minimum value of the makespan, where Γ is a parameter of the energy landscape characterizing the escape depth from local minima.

2 The Flow Shop Problem

The flow shop scheduling problem can be formalized as follows. There are a set J of l jobs and a set M of m machines. Each job has exactly one task to be processed on each machine. Therefore, we have $n := l \cdot m$ tasks each with a given processing time $p(t) \in \mathbb{N}$. There is a binary relation R on the set of tasks T that decomposes T into chains corresponding to the jobs. The binary relation, which represents *precedences* between the tasks is defined as follows: For every $t \in T$ there exists at most one t' such that $(t, t') \in R$. If $(t, t') \in R$, then $J(t) = J(t')$ and there is no $x \notin \{t, t'\}$ such that $(t, x) \in R$ or $(x, t') \in R$. For any $(v, w) \in R$, v has to be performed before w. R induces a total ordering of the tasks belonging to the same job. There exist no precedences between tasks of different jobs. Clearly, if $(v, w) \in R$ then $M(v) \neq M(w)$. The order in which a job passes all machines is the same for all jobs. As the *task number* of a task t we will denote the number of tasks preceding t within its job. We can therefore assume that all tasks with task number i are processed on machine M_i. A schedule is a function $S : T \to \mathbb{N} \cup \{0\}$ that for each task t defines a starting time $S(t)$.

The *length*, respectively the *makespan* of a schedule S is defined by

$$\lambda(S) := \max_{v \in T} \bigl(S(v) + p(v) \bigr), \tag{1}$$

i.e., the earliest time at which all tasks are completed. The problem is to find an *optimum* schedule, that is feasible and of minimum length. A flow shop scheduling problem can be represented by a *disjunctive graph*, a model introduced by Roy and Sussmann in [17]. The disjunctive graph is a graph $G = (V, A, E, \mu)$, which is defined as follows:

$$\begin{aligned} V &= T \cup \{I, O\}; \\ A &= \big\{[v,w] \mid v,w \in T, (v,w) \in R\big\} \cup \\ &\quad \big\{[I,w] \mid w \in T, \nexists v \in T : (v,w) \in R\big\} \cup \\ &\quad \big\{[v,O] \mid v \in T, \nexists w \in T : (v,w) \in R\big\}; \\ E &= \big\{\{v,w\} \mid v,w \in T, v \neq w, M(v) = M(w)\big\}; \\ \mu &: V \to \mathbb{N}. \end{aligned}$$

The vertices in V represent the tasks. In addition, there are a source (I) and a sink (O) which are two dummy vertices. All vertices in V are weighted. The weight of a vertex $\mu(v)$ is given by the processing time $p(v)$, $\mu(v) := p(v)$, $(\mu(I) = \mu(O) = 0)$. The arcs in A represent the given precedences between the tasks. The edges in E represent the machine capacity constraints, i.e., $\{v,w\} \in E$ with $v,w \in T$ and $M(v) = M(w)$ denotes the disjunctive constraint and the two ways to settle the disjunction correspond to the two possible orientations of $\{v,w\}$. The source I has arcs emanating to all the first tasks of the jobs and the sink O has arcs coming from all final tasks of jobs.

An *orientation* on E is a function $\Omega : E \to T \times T$ such that $\Omega(\{v,w\}) \in \{\langle v,w\rangle, \langle w,v\rangle\}$ for each $\{v,w\} \in E$. A schedule is *feasible* if the corresponding orientation Ω on E ($\Omega(E) = \{\Omega(e) \mid e \in E\}$) results in a directed graph (called digraph) $D := G' = (V, A, E, \mu, \Omega(E))$ which is acyclic.

A *path* P from x_i to x_j, $i, j \in \mathbb{N}, i < j : x_i, x_j \in V$ of the digraph D is a sequence of vertices $(x_i, x_{i+1}, \ldots, x_j) \in V$ such that for all $i \le k < j$, $[x_k, x_{k+1}] \in A$ or $\langle x_k, x_{k+1}\rangle \in \Omega(E)$.

The length of a path $P(x_i, x_j)$ is defined by the sum of the weights of all vertices in P: $\mu(P(x_i, x_j)) = \sum_{k=i}^{j} \mu(x_k)$. The makespan of a feasible schedule is determined by the length of a longest path (i.e., a critical path) in the digraph D. The problem of minimizing the makespan therefore can be reduced to finding an orientation Ω on E that minimizes the length of $\mu(P_{\max})$.

3 Basic Definitions

Simulated annealing algorithms are acting within a configuration space in accordance with a certain neighborhood structure or a set of transition rules, where the particular steps are controlled by the value of an objective function. The configuration space, i.e., the set of feasible solutions of a given problem instance is denoted by $\mathcal{F}$. For all instances, the number of tasks of each job equals the number of machines and each job has precisely one operation on each machine. Therefore, the size of $\mathcal{F}$ can be upper bounded in the following way: In the disjunctive graph G there are at most $l!$ possible orientations to process l tasks on a single machine. Hence, we have $|\mathcal{F}| \le \left(l!\right)^{m}$.

To describe the neighborhood of a solution $S \in F$, we define a *neighborhood function* $: F \to \wp(F)$. The neighborhood of S is given by $(S) \subseteq F$, and each solution in (S) is called a neighbor of S. Van Laarhoven et al. [22] propose a neighborhood function $_L$ for solving job shop scheduling problems which is based on interchanging two adjacent tasks of a block. A *block* is a maximal sequence of adjacent tasks that are processed on the same machine and do belong to a longest path. We will use their neighborhood function with the extension that we allow changing the orientation of an arbitrary arc which connects two tasks on the same machine:

(i) Choosing two vertices v and w such that $M(v) = M(w) = k$ with $e = \langle v, w \rangle \in \Omega(E)$;
(ii) Reversing the order of e such that the resulting arc $e' \in \Omega'(E)$ is $\langle w, v \rangle$;
(iii) If there exists an arc $\langle u, v \rangle$ such that $v \neq u, M(u) = k$, then replace the arc $\langle u, v \rangle$ by $\langle u, w \rangle$;
(iv) If there exists an arc $\langle w, x \rangle$ such that $w \neq x, M(x) = k$, then replace the arc $\langle w, x \rangle$ by $\langle v, x \rangle$.

Thus, the neighborhood structure is characterized by

Definition 1 *The schedule S' is a neighbor of S, $S' \in (S)$, if S' can be obtained by the transition rules* $1 - 4$ *or $S' = S$.*

Our choice is motivated by two facts:

- In contrast to the job shop scheduling the transition rules do guarantee for the flow shop a priori that the resulting schedule is feasible, i.e., that the corresponding digraph is acyclic.
- The extension of allowing to reverse the orientation of an arbitrary arc leads to an important property of the neighborhood function, namely reversibility.

Thus, the neighborhood structure is such that the algorithm visits only digraphs corresponding to feasible solutions and is equipped with a symmetry property which is required by our convergence analysis.

Lemma 1 *Suppose that $e = \langle v, w \rangle \in \Omega(E)$ is an arbitrary arc of an acyclic digraph D. Let D' be the digraph obtained from D by reversing the arc e. Then D' is also acyclic.*

Proof: Suppose D' is cyclic. Because D is acyclic, the arc $\langle w, v \rangle$ is part of the cycle in D'. Consequently, there is a path $P = (v, x_1, x_2, \ldots, x_i, w)$ in D'. Since w is processed before v on machine M_k at least two arcs of the path P are connecting vertices of the same job. From the definition of the flow shop problem that implies at least two vertices have a task number greater than k. Neither within a job nor within a machine there is an arc $\langle y, z \rangle$ such that the task number of y is greater than the task number of z. This contradicts that the path P exists in D'. Hence, D' is acyclic. q.e.d.

As already mentioned in Section 2, the objective is to minimize the makespan of feasible schedules. Hence, we define $Z(S) := \lambda(P_{\max})$, where $P_{\max}$ is a longest path in $D(S)$. Furthermore, we set

$$F_{\min} := \Big\{ S \mid S \in F \text{ and } \forall S' \big(S' \in F \rightarrow Z(S') \geq Z(S) \big) \Big\}. \tag{2}$$

For the special case of η_L, Van Laarhoven et al. have proved the following

Theorem 1 [22] *For each schedule $S \notin F_{\min}$, there exists a finite sequence of transitions leading from S to an element of $F_{\min}$.*

The probability of generating a solution S' from S can be expressed by

$$G[S; S'] := \begin{cases} \frac{1}{|\eta(S)|}, & \text{if } S' \in \eta(S) \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

with $|\eta(S)| \leq n - m + 1$ which follows from Definition 1.

Since the neighborhood function η_L from [22] is a special case of our transition rules 1 − 4, we have:

Lemma 2 *Given $S \in F \setminus F_{\min}$, there exists $S' \in \eta(S)$ such that $G[S; S'] > 0$.*

The acceptance probability $A[S; S']$, $S' \in \eta(S) \subseteq F$, is given by:

$$A[S; S'] := \begin{cases} 1, & \text{if } Z(S') - Z(S) \leq 0, \\ e^{-\frac{Z(S') - Z(S)}{c}}, & \text{otherwise,} \end{cases} \tag{4}$$

where c is a control parameter having the interpretation of a *temperature* in annealing procedures. Finally, the probability of performing the transition between S and S', $S, S' \in F$, is defined by

$$\Pr\{S \rightarrow S'\} = \begin{cases} G[S; S'] \cdot A[S; S'], & \text{if } S' \neq S, \\ 1 - \sum_{Q \neq S} G[S; Q] \cdot A[S; Q], & \text{otherwise.} \end{cases} \tag{5}$$

Let $a_S(k)$ denote the probability of being in the configuration S after k steps performed for the same value of c. The probability $a_S(k)$ can be calculated in accordance with

$$a_S(k) := \sum_{Q} a_Q(k-1) \cdot \Pr\{Q \rightarrow S\}. \tag{6}$$

The recursive application of (6) defines a Markov chain of probabilities $a_S(k)$. If the parameter $c = c(k)$ is a constant c, the chain is said to be a *homogeneous* Markov chain; otherwise, if $c(k)$ is lowered at any step, the sequence of probability vectors $a(k)$ is an *inhomogeneous* Markov chain.

We consider a cooling schedule which defines a special type of inhomogeneous Markov chains. For this cooling schedule, the value $c(k)$ changes in accordance with

$$c(k) = \frac{\Gamma}{\ln(k+2)}, \; k = 0, 1, \ldots . \tag{7}$$

The choice of $c(k)$ is motivated by Hajek's Theorem [11] on logarithmic cooling schedules for inhomogeneous Markov chains. If there exists $S_0, S_1, \ldots, S_r \in \mathcal{F} \; S_0 = S \wedge S_r = S'$ such that $G[S_u, S_{u+1}] > 0, u = 0, 1, \ldots, (r-1)$ and $Z(S_u) \leq h$, for all $u = 0, 1, \ldots, r$, we denote $height(S \Rightarrow S') \leq h$. The schedule S is a *local minimum*, if $S \in \mathcal{F} \setminus \mathcal{F}_{\min}$ and $Z(S') > Z(S)$ for all $S' \in \mathcal{N}_L(S) \setminus S$. By $depth(S_{\min})$ we denote the smallest h such that there exists a $S' \in \mathcal{F}$, where $Z(S') < Z(S_{\min})$, which is reachable at height $Z(S_{\min}) + h$.

The following convergence property has been proved by B. Hajek:

Theorem 2 [11] *Given a configuration space $\mathcal{C}$ and a cooling schedule defined by*

$$c(k) = \frac{\Gamma}{\ln(k+2)}, \; k = 0, 1, \ldots,$$

the asymptotic convergence $\sum_{H \in \mathcal{C}} a_H(k) \underset{k \to \infty}{\longrightarrow} 1$ of the stochastic algorithm, which is based on (2), (4), and (5), is guaranteed if and only if

(i) $\forall H, H' \in \mathcal{C} \, \exists H_0, H_1, \ldots, H_r \in \mathcal{C} \; H_0 = H \wedge H_r = H' : G[H_u, H_{u+1}] > 0,$ $l = 0, 1, \ldots, (r-1)$;

(ii) $\forall h : height(H \Rightarrow H') \leq h \iff height(H' \Rightarrow H) \leq h$;

(iii) $\Gamma \geq \max\limits_{H_{\min}} depth(H_{\min})$.

The condition (i) expresses the connectivity of the configuration space. As already mentioned above, with the choice of our neighborhood relation we can guarantee the mutual reachability of schedules. Therefore, Hajek's Theorem can be applied to our configuration space $\mathcal{F}$ with the neighborhood relation $\mathcal{N}$.

Before we perform the convergence analysis of the logarithmic cooling schedule defined in (7), we point out some properties of the configuration space and the neighborhood function. Let S and S' be feasible schedules and $S' \in \mathcal{N}(S)$. To obtain S' from S, we chose the arc $e = \langle v, w \rangle$ with $M(v) = M(w)$.

If $Z(S) < Z(S')$, then only a path containing one of the selected vertices v, w can determine the new makespan after the transition move. It can be shown that all paths whose length increase contain the edge $e' = \langle w, v \rangle$. Therefore, we have the following upper bound.

Lemma 3 *The increase of the objective function ΔZ in a single step according to $\mathcal{N}(S \to S')$ can be upper by $p(v) + p(w)$.*

The reversibility of the neighborhood function implies for the maximum distance of neighbors $S' \in \mathcal{N}(S)$ to $\mathcal{F}_{\min}$ in relation to S itself: If the minimum number of transitions to reach from S an optimum element is N, then for any $S' \in \mathcal{N}(S)$ the minimum number of transitions is at most $N + 1$.

4 Convergence Analysis

Our convergence results will be derived from a careful analysis of the "exchange of probabilities" among feasible solutions which belong to adjacent distance levels to optimum schedules, i.e., in addition to the value of the objective function, the elements of the configuration space are further distinguished by the minimal number of transitions required to reach an optimum schedule. We first introduce a recurrent formula for the expansion of probabilities and then we prove the main result on the convergence rate which relates properties of the configuration space to the speed of convergence. Throughout the section we employ the fact that for a proper choice of Γ the logarithmic cooling schedule leads to an optimum solution.

To express the relation between S and S' according to their value of the objective function we will use $<_Z$, $>_Z$, and $=_Z$ to simplify the expressions:

$$\begin{array}{lll} S <_Z S' \text{ instead of} & & S' \in \mathcal{N}(S) \,\&\, (Z(S) < Z(S')), \\ S >_Z S' \text{ instead of} & & S' \in \mathcal{N}(S) \,\&\, (Z(S) > Z(S')), \\ S =_Z S' \text{ instead of} & S \neq S' \,\&\, & S' \in \mathcal{N}(S) \,\&\, (Z(S) = Z(S')). \end{array}$$

Furthermore, we denote:

$$p(S) := |\{S <_Z S'\}|; \quad q(S) := |\{S =_Z S'\}|; \quad r(S) := |\{S >_Z S'\}|.$$

These notations imply

$$p(S) + q(S) + r(S) = |\mathcal{N}(S)| - 1 = m \cdot l - m - 1. \tag{8}$$

The equation is valid because there are $m \cdot (l - 1)$ arcs which are allowed to be switched and S belongs to its own neighborhood. Therefore, the size of the neighborhood is independent of the particular schedule S, and we set $n' := m \cdot l - m$.

Now, we analyze the probability $\mathbf{a}_S(k)$ to be in the schedule $S \in \mathcal{F}$ after k steps of the logarithmic cooling schedule defined in (7), and we use the notation

$$\frac{1}{\left(k+2\right)^{\frac{Z(S)-Z(S')}{\Gamma}}} = e^{-\frac{Z(S)-Z(S')}{c(k)}}, \; k \geq 0. \tag{9}$$

By using (3) till (5), one obtains from (6) by straightforward calculations

$$\mathbf{a}_S(k) = \mathbf{a}_S(k-1) \cdot \left(\frac{p(S)+1}{n'} - \sum_{\substack{i=1 \\ S <_Z S_i}}^{p(S)} \frac{1}{n'} \cdot \frac{1}{\left(k+1\right)^{\frac{Z(S_i)-Z(S)}{\Gamma}}} \right) + \tag{10}$$

$$+ \sum_{\substack{i=1 \\ S_i \leq_Z S}}^{p(S)+q(S)} \frac{\mathbf{a}_{S_i}(k-1)}{n'} + \sum_{\substack{j=1 \\ S_j <_Z S}}^{r(S)} \frac{\mathbf{a}_{S_j}(k-1)}{n'} \cdot \frac{1}{\left(k+1\right)^{\frac{Z(S)-Z(S_j)}{\Gamma}}}.$$

The representation (expansion) will be used in the following as the main relation reducing $\mathbf{a}_S(k)$ to probabilities from previous steps. We introduce the following partition of the set of schedules with respect to the value of the objective function:

$$L_0 := \mathcal{F}_{\min} \text{ and } L_{h+1} := \{S : S \in \mathcal{F} \wedge \forall S' \Big(S' \in \mathcal{F} \setminus \bigcup_{i=0}^{h} L_i \rightarrow Z(S') \leq Z(S) \Big) \}.$$

The highest level within F is denoted by $L_{h_{max}}$. Given $S \in F$, we further denote by $W_{min}(S) := [S; S_{k-1}; \ldots; S']$ a shortest sequence of transitions from S to $F_{\min}$, i.e., $S' \in F_{\min}$. Thus, we have for the distance $d(S) := length\big(W_{min}(S)\big)$. We introduce another partition of F with respect to $d(S)$:

$$S \in D_i \iff d(S) = i \geq 0; \quad \text{and} \quad D_s = \bigcup_{i=1}^{s} D_i; \quad \text{i.e.,} \quad F = D_s.$$

Thus, we distinguish between distance levels D_i related to the minimal number of transitions required to reach an optimal schedule from $F_{\min}$ and the levels L_h which are defined by the objective function. By definition, we have $D_0 := L_0 = F_{\min}$. We will use the following abbreviations:

$$(11) \qquad f(S';S;t) := \frac{1}{\left(k+2-t\right)^{\frac{Z(S')-Z(S)}{\Gamma}}} \quad \text{and}$$

$$(12) \qquad K_S(k-t) := \frac{p(S)+1}{n'} - \sum_{\substack{i=1\\ S <_Z S_i}}^{p(S)} \frac{1}{n'}\left(k+2-t\right)^{-\frac{Z(S_i)-Z(S)}{\Gamma}}.$$

We are going backwards from the k^{th} step and expanding $a_S(k)$ in accordance with (10). Our aim is to find a close upper bound for the value $\sum_{S \notin D_0} a_S(k)$ in terms of probabilities from previous steps.

During the expansion (10) of $a_S(k)$, $S \notin D_0$, terms according to S are generated as well as according to all neighbors S' of S. Some terms generated by the expansion of S contain the factor $a_{S'}(k-1)$ and can therefore be summarized with terms generated by the expansion of S'. However, it is important to distinguish between elements from D_1 and elements from D_i, $i \geq 2$. For all $S \notin D_1$, we obtain the following:

$$(13) \quad a_S(k-1)\cdot\Bigg(\frac{p(S)+1+q(S)+r(S)}{n'} - \sum_{\substack{i=1\\ S <_Z S_i}}^{p(S)} \frac{1}{n'}\cdot\frac{1}{\left(k+1\right)^{\frac{Z(S_i)-Z(S)}{\Gamma}}} + \sum_{\substack{i=1\\ S <_Z S_i}}^{p(S)} \frac{1}{n'}\cdot\frac{1}{\left(k+1\right)^{\frac{Z(S_i)-Z(S)}{\Gamma}}}\Bigg) = a_S(k-1).$$

In case of $S \in D_1$, some neighbors S' of S are elements of D_0 and do not generate the terms related to $S >_Z S'$ because the $a_{S'}(k)$ are not expanded since they are not present in the sum $\sum_{S \notin D_0} a_S(k)$. Therefore, $r'(S) \leq r(S)$ many terms are missing for $S \in D_1$ and the following arithmetic term is generated:

$$(14) \qquad a_S(k-1)\cdot\left(1-\frac{r'(S)}{n'}\right);$$

where $r'(S) := |\{S' : S' \in \mathcal{N}(S) \wedge S' \in D_0\}|$. On the other hand, the expansion of $a_S(k)$ generates terms related to $S' \in D_0$ with $S' <_Z S$ and containing $a_{S'}(k-1)$

as a factor. Those terms are not canceled by expansions of $a_{S'}(k)$. All $S \in D_1$ therefore generate the following term:

$$(15)\qquad \sum_{\substack{j=1\\ S_j \in D_0 \cap (S)}}^{r'(S)} \frac{a_{S_j}(k-1)}{n'} \cdot \left(\frac{1}{k+1}\right)^{\frac{Z(S)-Z(S_j)}{r}}.$$

Now, we consider the entire sum and take the negative product $a_S(k) \cdot r'(S)/n'$ separately. By using the abbreviations introduced in (12) we derive the following lemma.

Lemma 4 *After one step of the expansion of* $\sum_{S \notin D_0} a_S(k)$*, the sum can be represented by*

$$\sum_{S \notin D_0} a_S(k) = \sum_{S \notin D_0} a_S(k-1) - \sum_{S \in D_1} \frac{r'(S)}{n'} \cdot a_S(k-1) +$$
$$+ \sum_{S \in D_1} \sum_{\substack{j=1\\ S_j \in D_0 \cap (S)}}^{r'(S)} \frac{f(S,S_j,1)}{n'} \cdot a_{S_j}(k-1).$$

The diminishing factor $\left(1 - r'(S)/n'\right)$ appears by de nition for all elements of D_1. At subsequent reduction steps, the factor is \transmitted" successively to all probabilities from higher distance levels D_i because any element of D_i has at least one neighbor from D_{i-1}. The main task is now to analyze how this diminishing factor changes, if it is propagated to the next higher distance level. We denote

$$(16)\qquad \sum_{S \notin D_0} a_S(k) = \sum_{S \notin D_0} (S,t) \cdot a_S(k-t) + \sum_{S' \in D_0} (S',t) \cdot a_{S'}(k-t),$$

i.e., the coe cients (S,t) and (S',t) are the factors at probabilities after t steps of an expansion of $\sum_{S \notin D_0} a_S(k)$. Hence, for $S \in D_1$ we have $(S,1) = 1 - r'(S)/n'$, and $(S,1) = 1$ for the remaining $S \in D_s \setminus (D_0 \cup D_1)$. For $S' \in D_0$ we have from Lemma 4:

$$(17)\qquad (S',1) = \sum_{\substack{i=1\\ S_i \in D_1 \wedge S' \in (S_i)}}^{p(S')} \frac{f(S_i,S',1)}{n'}.$$

Starting from step $(k-1)$, the generated probabilities $a_{S'}(k-u)$ are expanded in the same way as all other probabilities. We set $(S,j) := 1 - (S,j)$ because we are mainly interested in the convergence $(S,j) \to 0$. We perform an inductive step from $(k-t+1)$ to $(k-t)$ and obtain for $t \geq 2$:

Lemma 5 *The following recurrent relation is valid for the coe cients* $(S;t)$, $t\ \ 2$:

$$(S;t) = (S;t-1)\ \ K_S(k-t) + \sum_{S\ \ z S'} \frac{(S';t-1)}{n'} + \sum_{S <_z S''} \frac{(S'';t-1)}{n'}\ \ f(S'';S;t).$$

Furthermore, for the three special cases $S \in D_j;\ j > t$, $S \in D_1;\ t = 1$, *and* $S \in D_0;\ t = 1$ *we have,* $(S;t) = 0$, $(S;t) = r'(S)/n'$, *and* $(S;t) = 1 - \sum_{j=1}^{p(S)} f(S_j;S;1)/n'$, *with* $S_j \in D_1 \wedge S \in (S_j)$ *respectively.*

Exactly the same structure of the equation is valid for $(S;t)$ which will be used for elements of D_0 only because these elements are not present in the original sum $\sum_{S \in D_0} a_S(k)$. Now, any $(S;t)$ and $(S;t)$ is expressed by a sum $\sum_u T_u$ of arithmetic terms. We consider in more details the terms associated with elements S° of D_0 and $S^{\bullet}$ of D_1. We assume a representation $(S^{\circ};t-1) = \sum T(S^{\circ})$, and $(S;t-1) = \sum T(S)$, $S \in D_0$.

If we consider $r'(S^{\bullet})/n'$ and $\sum_{S^{\circ} <_z S^{\bullet}} f(S^{\bullet};S^{\circ};t)/n'$ separately, the di culties arising from the de nition $(S;t) := 1 - (S;t)$ can be avoided, i.e., we have to take into account only changing signs of terms during the transmission from D_1 to D_0 and vice versa.

De nition 2 *The two expressions* $r'(S^{\bullet})/n'$, *and* $\sum_{S^{\circ} <_z S^{\bullet}} f(S^{\bullet};S^{\circ};t)/n'$, *are called* source terms *of* $(S^{\bullet};t)$ *and* $(S^{\circ};t)$, *respectively.*

During an expansion of $\sum_{S \in D_0} a_S(k)$ backwards according to (13), the source terms are distributed permanently to higher distance levels D_j. Therefore, at higher distance levels the notion of a source term can be de ned by an inductive step:

De nition 3 *For all* $S \in D_i$, $i > 1$, *any term which is generated according to the equation of Lemma 5 from a source term of* $(S';t-1)$, *where* $S' \in D_{i-1} \cap (S)$, *is said to be a source term of* $(S;t)$.

We introduce a counter $e(T)$ to terms T which indicates the step at which the term has been generated from source terms. The value $e(T)$ is called the rate of a term and we set $e(T) = 1$ for source terms T.

The value $e(T) > 1$ is assigned to terms related to D_0 and D_1 in a slightly di erent way compared to higher distance levels because at the rst step the S° do not participate in the expansion of $\sum_{S \in D_0} a_S(k)$. Furthermore, in the case of D_0 and D_1 we have to take into account the changing signs of terms which result from the simultaneous consideration of $(S^{\bullet};t)$ (for D_1) and $(S^{\circ};t)$ (for D_0).

De nition 4 *A term* T° *is called a* jth rate term *of* $(S^{\circ};t)$, $S^{\circ} \in D_0$ *and* $j\ \ 2$, *if either* $S^{\circ} = -T$ *and* $e(T) = j-1$ *for some* $(S;t-1)$, $S \in D_1 \cap (S^{\circ})$, *or* $e(T^{\circ}) = j-1$ *for some* $(S';t-1)$, $S' \in D_0 \cap (S^{\circ})$.

A term T *is called a* jth rate term *of* $(S;t)$, $S^{\bullet} \in D_1$ *and* $j\ \ 2$, *if* $e(T) = j-2$ *for some* $(S';t-1)$, $S' \in D_2 \cap (S^{\bullet})$, $e(T) = j-1$ *for some* $(S';t-1)$,

$S' \in D_1 \cap (S^{1})$, *or* $T = -T'$ *and* $e(T') = j - 1$ *for some* $S^{\bullet} \in D_0 \cap (S^{1})$ *with respect to* $(S^{\bullet}; t-1)$.

A term T *is called a* j^{th} rate term *of* $(S;t)$, $S \in D_i$ *and* $i; j \;\; 2$, *if* $e(T) = j-1$ *for some* $(S'; t-1)$, $S' \in D_{i+1} \cap (S)$, $e(T) = j-1$ *for some* $(S'; t-1)$, $S' \in D_i \cap (S)$, *or* T *is a* j^{th} *rate term of* $(S; t-1)$ *for some* $S \in D_{i-1}$.

The classi cation of terms will be used for a partition of the summation over all terms which constitute particular values $(S^{1}; t)$ and $(S^{\bullet}; t)$. Let $T_j(S;t)$ be the set of j^{th} rate arithmetic terms of $(S^{1}; t)$ ($(S^{\bullet}; t)$) related to $S \in D_s$. We set

$$(18) \qquad \mathsf{A}_j(S;t) := \sum_{T \in T_j(S;t)} T.$$

The same notation is used in case of $S = S^{\bullet} \in D_0$ with respect to $(S^{1}; t)$, and we obtain by induction

$$(19) \qquad (S;t) = \sum_{j=1}^{t-k+1} \mathsf{A}_j(S;t) \quad \text{and} \quad (S^{\bullet};t) = \sum_{j=1}^{k} \mathsf{A}_j(S^{\bullet};t).$$

For $S \in D_i \neq D_1; D_0$ and $j \;\; 2$ we obtain immediately from Lemma 5 and De nition 4:

$$(20) \quad \mathsf{A}_j(S;t) = \mathsf{A}_{j-1}(S;t-1) \;\; K_S(k-t) + \\
+ \sum_{\substack{S' \;\; z\, S \\ S' \in D_{i+1}}} \frac{\mathsf{A}_{j-2}(S';t-1)}{n'} + \sum_{\substack{S'' >_z S \\ S'' \in D_{i+1}}} \frac{\mathsf{A}_{j-2}(S'';t-1)}{n'} \; f(S'';S;t) + \\
+ \sum_{\substack{S' \;\; z\, S \\ S' \in D_i}} \frac{\mathsf{A}_{j-1}(S';t-1)}{n'} + \sum_{\substack{S'' >_z S \\ S'' \in D_i}} \frac{\mathsf{A}_{j-1}(S'';t-1)}{n'} \; f(S'';S;t) + \\
+ \sum_{\substack{S' \;\; z\, S \\ S' \in D_{i-1}}} \frac{\mathsf{A}_j(S';t-1)}{n'} + \sum_{\substack{S'' >_z S \\ S'' \in D_{i-1}}} \frac{\mathsf{A}_j(S'';t-1)}{n'} \; f(S'';S;t).$$

In case of $S \in D_1$ and $j \;\; 2$, we have in accordance with De nition 4:

$$(21) \quad \mathsf{A}_j(S;t) = \mathsf{A}_{j-1}(S;t-1) \;\; K_S(k-t) + \\
+ \sum_{\substack{S' \;\; z\, S \\ S' \in D_1}} \frac{\mathsf{A}_{j-1}(S';t-1)}{n'} + \sum_{\substack{S'' >_z S \\ S'' \in D_1}} \frac{\mathsf{A}_{j-1}(S'';t-1)}{n'} \; f(S'';S;t) + \\
+ \sum_{\substack{S'' >_z S \\ S'' \in D_2}} \frac{\mathsf{A}_{j-2}(S'';t-1)}{n'} \; f(S'';S;t) - \sum_{S' \in D_0 \cap (S)} \frac{\mathsf{A}_{j-1}(S';t-1)}{n'}.$$

Finally, the corresponding relation for S' is given by

$$(22)\quad \mathrm{A}_j(S';t) = \mathrm{A}_{j-1}(S';t-1) \cdot K_{S'}(k-t) - \sum_{S >_Z S'} \frac{\mathrm{A}_{j-1}(S;t-1)}{n'} \cdot f(S;S';t).$$

We incorporate (20) till (22) in the following upper bound:

Lemma 6 *Given $S \in F$, $k \geq n^{O(\Gamma)}$, there exist constants $a, b, c > 1$ such that*

$$|\mathrm{A}_j(S;t)| < j^a \cdot 2^{-k^{1/b}};$$

where $j \leq k/c$ is required.

The proof is performed by induction, using the representations (20) till (22) for increasing i and j, and we employ similar relations as in [19], Lemma 10 and Lemma 11. In the present case, we utilize the reversibility of the neighborhood relation and therefore the lower bound on k depends directly on Γ.

We compare the computation of $(S;t)$ (and $(S';t)$) for two di erent values $t = k_1$ and $t = k_2$, i.e., $(S;t)$ is calculated backwards from k_1 and k_2, respectively. To distinguish between $(S;t)$ and related values, which are de ned for di erent k_1 and k_2, we will use an additional upper index. At this point, we use again the representation (19) of $(S;t)$ (and the corresponding equation for $(S';t)$).

Lemma 7 *Given $k_2 \geq k_1$ and $S \in D_i$, then*

$$\mathrm{A}_2^1(S;t) = \mathrm{A}_2^2(S;k_2 - k_1 + t); \quad if \quad t \geq i + 2.$$

The proposition can be proved straightforward by induction over i, i.e., the sets D_i. Lemma 7 implies that at step $s + 2$ (with respect to k_1)

$$\mathrm{A}_2^1(S;s+2) = \mathrm{A}_2^2(S;k_2 - k_1 + s + 2) \quad \text{for all} \quad S \in D_s.$$

For $\mathrm{A}_1^1(S;t)$, the corresponding equality is already satis ed in case of $t \geq s$. The relation can be extended to all values $j \geq 2$:

Lemma 8 *Given $k_2 \geq k_1$, $j \geq 1$, and $S \in D_i$, then*

$$\mathrm{A}_j^1(S;t) = \mathrm{A}_j^2(S;k_2 - k_1 + t); \quad if \quad t \geq 2 \cdot (j-1) + i.$$

We recall that our main goal is to upper bound the sum $\sum_{S \notin D_0} a_S(k)$. When $a(0)$ denotes the initial probability distribution, we have from (16):

$$(23)\quad \Big|\sum_{S \notin D_0} a_S(k_1) - \sum_{S \notin D_0} a_S(k_2)\Big| \leq \sum_{S \notin D_0} |(S;k_1) \cdot a_S(0)| + \Big| \sum_{S' \in D_0} (S';k_1) - \sum_{S' \in D_0} (S';k_1) \cdot a_{S'}(0)\Big| :$$

Lemma 9 *Given k_2 $k_1 > n^{O(\Gamma)}$, then*

$$\Bigl|\sum_{S \not\in D_0} \bigl((S;k_2) - (S;k_1) \bigr) a_S(0) \Bigr| < 2^{-k_1^{1/}}$$

for a suitable constant > 1.

The proof follows straightforward from Lemma 6 and Lemma 8. The same relation can be derived for the $(S';k_{1/2})$. Now, we can immediately prove

Theorem 3 *The condition*

$$k > n^{O(\Gamma)} \log^{O(1)} \frac{1}{}$$

implies for arbitrary initial probability distributions $a(0)$ *and* > 0:

$$\sum_{S \not\in D_0} a_S(k) < \quad \text{and therefore,} \quad \sum_{S' \in D_0} a_{S'}(k) \geq 1 - .$$

Proof: We choose k in accordance with Lemma 9 and we consider

$$\begin{aligned}
\sum_{S \not\in D_0} a_S(k) &= \sum_{S \not\in D_0} \bigl(a_S(k) - a_S(k_2) \bigr) + \sum_{S \not\in D_0} a_S(k_2) \\
&= \sum_{S \not\in D_0} \bigl((S;k_2) - (S;k) \bigr) a_S(0) + \\
&\quad + \sum_{S' \in D_0} \bigl((S';k) - (S';k_2) \bigr) a_{S'}(0) + \sum_{S \not\in D_0} a_S(k_2).
\end{aligned}$$

The value k_2 from Lemma 9 is larger but independent of $k_1 = k$, i.e., we can take a $k_2 > k$ such that $\sum_{S \not\in D_0} a_S(k_2) < /3$. Here, we employ Theorem 1 and 2, i.e., if the constant Γ from (7) is sufficiently large, the inhomogeneous simulated annealing procedure defined by (3) till (5) tends to the global minimum of Z on F. We obtain the stated inequality, if additionally both differences $\sum_{S \not\in D_0} \bigl((S;k_2) - (S;k) \bigr)$ and $\sum_{S' \in D_0} \bigl((S';k) - (S';k_2) \bigr)$ are smaller than $/3$. Lemma 9 implies that the condition on the differences is satisfied in case of $k_1^{1/} > \log(3/)$. q.e.d.

References

1. E.H.L. Aarts. *Local Search in Combinatorial Optimization*. Wiley, New York, 1997.
2. E.H.L. Aarts, P.J.M. Van Laarhoven, J.K. Lenstra, and N.L.J. Ulder. A Computational Study of Local Search Algorithms for Shop Scheduling. *ORSA J. on Computing*, 6:118–125, 1994.
3. E.H.L. Aarts and J.H.M. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach*. Wiley, New York, 1989.
4. H. Baumgärtel. Distributed Constraint Processing for Production Logistics. In *PACT'97 – Practical Application of Constraint Technology*, Blackpool, UK, 1997.

5. O. Catoni. Rough Large Deviation Estimates for Simulated Annealing: Applications to Exponential Schedules. *Annals of Probability*, 20(3):1109 { 1146, 1992.
6. O. Catoni. Metropolis, Simulated Annealing, and Iterated Energy Transformation Algorithms: Theory and Experiments. *J. of Complexity*, 12(4):595 { 623, 1996.
7. C. Chen, V.S. Vempati, and N. Aljaber. An Application of Genetic Algorithms for Flow Shop Problems. *European J. of Operational Research*, 80:389{396, 1995.
8. P. Chretienne, E.G. Co man, Jr., J.K. Lenstra, and Z. Liu. *Scheduling Theory and Its Applications*. Wiley, New York, 1995.
9. M.K. El-Najdawi. Multi-Cyclic Flow Shop Scheduling: An Application in Multi-Stage, Multi-Product Production Processes. *International J. of Production Research*, 35:3323{3332, 1997.
10. M.R. Garey, D.S. Johnson, and R. Sethi. The Complexity of Flow Shop and Job Shop Scheduling. *Mathematics of Operations Research*, 1:117{129, 1976.
11. B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13:311 { 329, 1988.
12. L.A. Hall. Approximability of Flow Shop Scheduling. In *36th Annual Symposium on Foundations of Computer Science*, pp. 82{91, Milwaukee, Wisconsin, 1995.
13. C.Y. Lee and L. Lei, editors. *Scheduling: Theory and Applications*. Annals of Operations Research, Journal Edition. Baltzer Science Publ. BV, Amsterdam, 1997.
14. G. Liu, P.B. Luh, and R. Resch. Scheduling Permutation Flow Shops Using The Lagrangian Relaxation Technique. *Annals of Operations Research*, 70:171{189, 1997.
15. E. Nowicki and C. Smutnicki. The Flow Shop with Parallel Machines: A Tabu Search Approach. *European J. of Operational Research*, 106:226 { 253, 1998.
16. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall International Series in Industrial and Systems Engineering. Prentice Hall, Englewood Cli s, N.J., 1995.
17. B. Roy and B. Sussmann. *Les problemes d'Ordonnancement avec Constraints Disjonctives*. Note DS No.9 bis. SEMA, 1964.
18. D.L. Santos, J.L. Hunsucker, and D.E. Deal. Global Lower Bounds for Flow Shops with Multiple Processors. *European J. of Operational Research*, 80:112 { 120, 1995.
19. K. Steinhöfel, A. Albrecht, and C.K. Wong. On Various Cooling Schedules for Simulated Annealing Applied to the Job Shop Problem. In M. Luby, J. Rolim, and M. Serna, editors, *Proc. RANDOM'98*, pages 260 { 279, Lecture Notes in Computer Science, vol. 1518, 1998.
20. K. Steinhöfel, A. Albrecht, and C.K. Wong. Two Simulated Annealing-Based Heuristics for the Job Shop Scheduling Problem. *European J. of Operational Research*, 118(3):524{548,1999.
21. J.D. Ullman. *NP*-Complete Scheduling Problems. *J. of Computer and System Science*, 10(3):384{393, 1975.
22. P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40(1):113{125, 1992.
23. D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevast'janov, and D.B. Shmoys. Short Shop Schedules. *Operations Research*, 45:288{294, 1997.

On the Lovasz Number of Certain Circulant Graphs

Valentin E. Brimkov[1], Bruno Codenotti[2], Valentino Crespi[1], and Mauro Leoncini[2;3]

[1] Department of Mathematics, Eastern Mediterranean University, Famagusta, TRNC
fbrimkov, crespig.as@mozart.emu.edu.tr

[2] Istituto di Matematica Computazionale del CNR, Via S. Maria 46, 56126-Pisa, Italy
fcodenotti, leoncinig@imc.pi.cnr.it

[3] Facolta di Economia, Universita di Foggia, Via IV Novembre 1, 71100 Foggia, Italy

Abstract. The theta function of a graph, also known as the Lovasz number, has the remarkable property of being computable in polynomial time, despite being \sandwiched" between two hard to compute integers, i.e., clique and chromatic number. Very little is known about the explicit value of the theta function for special classes of graphs. In this paper we provide the explicit formula for the Lovasz number of the union of two cycles, in two special cases, and a practically e cient algorithm, for the general case.

1 Introduction

The notion of capacity of a graph was introduced by Shannon in [14], and after that labeled as *Shannon capacity*. This concept arises in connection with a graph representation for the problem of communicating messages in a zero-error channel. One considers a graph G, whose vertices are letters from a given alphabet, and where adjacency indicates that two letters can be confused. In this setting, the maximum number of one-letter messages that can be sent without danger of confusion is given by the independence number of G, here denoted by (G). If (G^k) denotes the maximum number of k-letter messages that can be safely communicated, we see that $(G^k) \quad (G)^k$. Moreover one can readily show that equality does not hold in general (see, e.g., [11]). The Shannon capacity of G is the number $(G) = \lim_{k ! 1} \sqrt[k]{(G^k)};$ which, by the previous observations, satis es $(G) \quad (G)$, where equality does not need to occur.

It was very early recognized that the determination of the Shannon capacity is a very di cult problem, even for small and simple graphs (see [8, 13]). In a famous paper of 1979, Lovasz introduced the theta function $\#(G)$, with the explicit goal of estimating (G) [11].

Shannon capacity and Lovasz theta function attracted a lot of interest in the scienti c community, because of the applications to communication issues, but also due to the connections with some central combinatorial and computational questions in graph theory, like computing the largest clique and nding the chromatic number of a graph (see [2, 3, 4, 6] for a sample of the wealth of di erent results and applications of $\#(G)$ and (G)). Despite a lot of work in the eld, nding the explicit value of the theta function for interesting special classes of graphs is still an open problem.

In this paper we present some results on the theta function of circulant graphs, i.e., graphs which admit a circulant adjacency matrix. We recall that a circulant matrix is fully determined by its rst row, each other row being a cyclic shift of the previous one. Such graphs span a wide spectrum, whose extremes are the single cycle and the complete graph. We either give a formula or an algorithm for computing the Lovasz number of circulant graphs given by the union of two cycles. The algorithm is based on the computation of the intersection of halfplanes and (although its running time is $O(n \log n)$ in the worst case, as compared with the linear time achievable through linear programming) is very e cient in practice, since it exploits the particular geometric structure of the intersection.

2 Preliminaries

There are several equivalent de nitions for the Lovasz theta function (see, e.g., the survey by Knuth [10]). We give here the one that comes out of Theorem 6 in [11], because it requires only little technical machinery.

De nition 1. *Let G be a graph and let $\mathbf{A}$ be the family of matrices $A = (a_{ij})$ such that $a_{ij} = 0$ if i and j are adjacent in G. Also, let ${}_1(A)$ ${}_2(A)$ $:::$ ${}_n(A)$ denote the eigenvalues of A. Then*

$$\#(G) = \max_{A 2 \mathbf{A}} \quad 1 - \frac{{}_1(A)}{{}_n(A)} \quad :$$

Combining the fact that (G) $\#(G)$ with the easy lower bound (C_5) $p\overline{5}$, Lovasz has been able to determine exactly the capacity of C_5, the pentagon, which turns out to be $p\overline{5}$.

For several families of simple graphs, the value of $\#(G)$ is given by explicit formulas. For instance, in the case of odd cycles of length n we have

$$\#(C_n) = \frac{n \cos(\; =n)}{1 + \cos(\; =n)} :$$

We now sketch the proof of correctness of the above formula (see [10] for more details), because it will be instrumental to the more general results obtained in this paper.

With reference to the de nition of the Lovasz number which resorts to the minimum of the largest eigenvalue over all feasible matrices (Section 6 in [10]), in the case of n-cycles, we have that a feasible matrix has ones everywhere, except on the superdiagonal, subdiagonal and the upper-right and lower-left corners, i.e. it can be written as $C = J + xP + xP^{-1}$, where J is a matrix whose entries are all equal to one, and P is the permutation matrix taking j into $(j+1) \mod n$. It is well known and easy to see that the eigenvalues of C are $n + 2x$, and $x(\omega^j + \omega^{-j})$, for $j = 1, \ldots, n-1$, where $\omega = e^{2\pi i/n}$. The minimum over x of the maximum of these values is obtained when $n + 2x = -2x \cos \pi/n$, which immediately leads to the above formula.

3 The Function ϑ of Circulant Graphs of Degree 4

Let n be an odd integer and let j be such that $1 < j \leq \frac{n-1}{2}$. Let $C(n, j)$ denote the circulant graph with vertex set $\{0, \ldots, n-1\}$ and edge set $\{\{i, i+1 \mod n\}, \{i, i+j \mod n\}, i = 0, \ldots, n-1\}$. By using the approach sketched in [10], we can easily obtain the following result.

Lemma 1. *Let $f_0(x, y) = n + 2x + 2y$ and, for some fixed value of j, $f_i(x, y) = 2x \cos \frac{2\pi i}{n} + 2y \cos \frac{2\pi ij}{n}$, $i = 1, \ldots, n-1$. Then*

$$\vartheta(C(n, j)) = \min_{x,y} \max_i \left\{ f_i(x, y) ;\ i = 0, 1, \ldots, \frac{n-1}{2} \right\} . \tag{1}$$

Proof. Follows from the same arguments which lead to the known formula for the Lovasz number of odd cycles [10] (i.e., taking advantage of the fact that we can restrict the set of feasible matrices within the family of circulant matrices) and observing that, for $i \geq 1$, $f_i(x, y) = f_{n-i}(x, y)$. ⊓⊔

3.1 A Linear Programming Formulation

Throughout the rest of this paper we will consider the following linear programming formulation of (1).

$$\begin{aligned} &\text{minimize } z \\ &\quad \text{s.t. } f_i(x, y) - z \leq 0;\ i = 0, \ldots, \tfrac{n-1}{2}; \\ &\qquad\qquad\qquad z \geq 0; \end{aligned} \tag{2}$$

where the $f_i(x, y)$'s are defined in Lemma 1.

Consider the intersection C of the closed halfspaces defined by $z \geq 0$ and $f_i(x, y) - z \leq 0$, $i = 1, \ldots, \frac{n-1}{2}$ (which is not empty, since any point $(0, 0, k)$, $k \geq 0$, satisfies all the inequalities). C is a polyhedral cone with the apex at the origin. This follows from the two following facts, which can be easily verified: (1) the equations $f_i(x, y) - z = 0$, $i \geq 1$, define hyperplanes through the origin; (2) for any $z_0 > 0$, the projection Q_{z_0} of $C \cap \{z = z_0\}$ onto the xy plane is a polygon, i.e., Q_{z_0} is bounded[1] (see Fig. 1, which corresponds to the graph $C(13, 2)$).

Consider now the first constraint of formulation (2). The region represented by such constraint is the halfspace above the plane A with equation $n + 2x +$

[1] In the appendix we shall give a rigorous proof of this fact for the case $j = 2$.

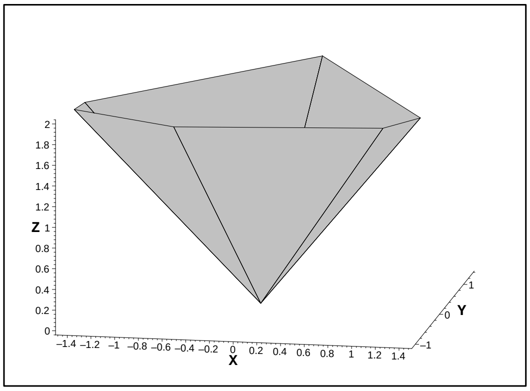

Fig. 1. The polyedral cone for $n = 13$ and $j = 2$ cut at $z = 2$

$2y - z = 0$. It is then easy to see that the minimum z of (2) will correspond to the point $P = (x; y; z)$ of C that is the last met by a sweeping line, parallel to the line $y = -x$, which moves on the surface of A towards the negative ortant (we will simply refer to these as to the *extremal* vertices). In particular, x and y are the coordinates of the extremal vertex v of the convex polygon Q_z in the third quadrant. The lines which de ne v have equations $2x \cos \quad + 2y \cos(\; j) = z$ and $2x \cos \quad + 2y \cos(\; j) = z$, where $\quad = \frac{2 \; i_1}{n}$ and $\quad = \frac{2 \; i_2}{n}$, for some indices i_1 and i_2. The key property, which we will exploit both to determine a closed formula for the # (in the cases $j = 2$ and $j = 3$) and to implement an e cient algorithm for the general case of circulant graphs of degree 4, is that i_1 and i_2 can be computed using \any" projection polygon Q_{z_0}, $z_0 > 0$, and determining its extremal point in the third quadrant. Once i_1 and i_2 are known, z can be computed by solving the following linear system

$$
\begin{array}{l}
8 \\
< 2x \cos \quad + 2y \cos(j \quad) - z = 0; \\
\;\; 2x \cos \quad + 2y \cos(j \quad) - z = 0; \\
: \qquad\qquad\qquad 2x + 2y - z = -n:
\end{array}
\tag{3}
$$

3.2 The Special Case $j = 2$

The detailed proof of the following theorem is deferred to the appendix.

Theorem 1.

$$\#(C(n; 2)) = n \quad 1 - \frac{\frac{1}{2} - \cos(\frac{2}{n} bn{=}3c) - \cos(\frac{2}{n}(bn{=}3c + 1))}{(\cos(\frac{2}{n} bn{=}3c) - 1)(\cos(\frac{2}{n}(bn{=}3c + 1)) - 1)} \quad : \tag{4}$$

3.3 The Special Case $j = 3$

Consider again the projection polygon Q_{z_0}, for some $z_0 > 0$. We know from Section 3.1 that the value of # is the optimal value z of the objective function in

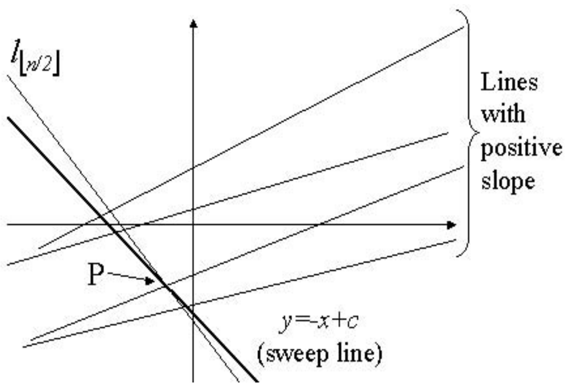

Fig. 2. Lines forming the extremal vertex P

the linear program (2), and that this value is achieved at the extremal vertex P of Q_z in the third quadrant. Also, we know that any projection polygon Q_{z_0} can be used to determine the two lines $f_i(x; y) - z = 0$, i 1, which form P. It turns out that nding such lines is easy when $j = 3$. In the following we will say that the line l_i has positive x-intercept (resp., y-intercept) if the intersection between l_i and the x-axis (resp., y-axis) has positive x-coordinate (resp., y-coordinate), otherwise we will say that the intercept is negative. The crucial observation is the following. Among the lines with negative x- and y-intercepts, $l_{\lfloor n/2\rfloor}$ is the one for which these intersections are closest to the origin. It then follows that P must lay on this line and (after a moment of thought) that the second line forming P must be searched among those with positive slope. Let x_i and y_i denote the coordinates of the intersection between the line l_i and the line $l_{\lfloor n/2\rfloor}$. Now, since $l_{\lfloor n/2\rfloor}$ is slightly steeper than the line with equation $y = -x$, the line sought will be the one with positive x-intercept, negative y-intercept, and such that y_i is maximum (see Fig. 2). We shall prove that such line is the one with index $n = \lceil\frac{n-3}{6}\rceil$.

To this end, observe rst that the requirement of positive x-intercept and negative y-intercept implies $\frac{n}{12} < i < \frac{n}{4}$ (recall that l_i has equation $y = -\frac{\cos\frac{2\ i}{n}}{\cos\frac{6\ i}{n}}x + \frac{z_0}{\cos\frac{6\ i}{n}}$). To prove that l_n maximizes y_i we show that, for any integer $i \neq n$ in the interval $\frac{n}{12} < i < \frac{n}{4}$, the three points $v_i = (x_i; y_i)$, $v_n = (x_n; y_n)$ and $O = (0; 0)$ form a clockwise circuit. We already know (see the Appendix) that this amounts to proving that $d(v_i; v_n; O) = x_i y_n - y_i x_n < 0$. This is easy; the only formal di culty is working with the integer part of $\frac{n-3}{6}$. Clearly this might be circumvent by dealing with the three di erent cases, namely $n = 6k + 1$, $n = 6k + 3$, and $n = 6k + 5$, for some positive integer k. For simplicity we shall prove the rst case only. Now, for $n = 6k + 1$ we have $\lceil\frac{n-3}{6}\rceil = \frac{n-1}{6}$ and, using $z_0 = 2$,

$$x_n = \frac{\cos\frac{3}{n} - \cos\frac{}{n}}{\cos\frac{3}{n}\cos\left(\frac{}{3} - \frac{}{3n}\right) + \cos^2\frac{}{n}}; \quad y_n = \frac{-\cos\frac{}{n} - \cos\left(\frac{}{3} - \frac{}{3n}\right)}{\cos\frac{3}{n}\cos\left(\frac{}{3} - \frac{}{3n}\right) + \cos^2\frac{}{n}};$$

$$x_i = \frac{\cos\frac{3}{n} + \cos\frac{6\ i}{n}}{\cos\frac{3}{n}\cos\frac{2\ i}{n} - \cos\frac{6\ i}{n}\cos\frac{}{n}}; \quad y_i = \frac{\cos\frac{}{n} + \cos\frac{2\ i}{n}}{\cos\frac{}{n}\cos\frac{6\ i}{n} - \cos\frac{3}{n}\cos\frac{2\ i}{n}}:$$

After some relatively simple algebra we obtain

$$d(v_i; v_n; O) = \frac{\ \ +\ \ + y}{\left(\cos\frac{3}{n}\cos\left(\frac{}{3} - \frac{}{3n}\right) + \cos^2\frac{}{n}\right)\left(\cos\frac{3}{n}\cos\frac{2\ i}{n} - \cos\frac{}{n}\cos\frac{6\ i}{n}\right)};$$

where

$$\begin{aligned} &= \cos\frac{}{n}\left(\cos\frac{6\ i}{n} + \cos\frac{}{n}\right); \\ &= \cos\left(\frac{2\ i}{n}\right)\left(\cos\frac{}{n} - \cos\frac{3}{n}\right); \\ y &= \cos\left(\frac{}{3} - \frac{}{3n}\right)\left(\cos\frac{3}{n} + \cos\frac{6\ i}{n}\right): \end{aligned}$$

It is easy to check that $\ ;\ ; y > 0$ while the denominator of $d(v_i; v_n; O)$ is negative for the admissible values of i. We are now able to determine the value of the # function of $C(n; 3)$.

Theorem 2.

$$\#(C(n;3)) = n\left(1 - \frac{\cos^2\frac{}{n} - \cos\frac{}{n}\cos\frac{2\ d\frac{n-3}{6}e}{n} + \cos^2\frac{2\ d\frac{n-3}{6}e}{n} - 1}{\left(\cos\frac{}{n} + 1\right)\left(\cos\frac{2\ d\frac{n-3}{6}e}{n} - 1\right)\left(1 - \cos\frac{}{n} + \cos\frac{2\ d\frac{n-3}{6}e}{n}\right)}\right): \quad (5)$$

Proof. $\#(C(n;3))$ is the value of z in the solution to the linear system (3) where $j = 3$, i.e., $z = n\left(1 - \frac{\cos^2\ \ + \cos\ \ \cos\ \ + \cos^2\ \ - 1}{(1 - \cos\ \ )(\cos\ \ - 1)(\cos\ \ + \cos\ \ + 1)}\right)$; where $\ = \frac{2\ i_1}{n}$ and $\ = \frac{2\ i_2}{n}$. By the previous results we know that $i_1 = b\frac{n}{2}c$ and $i_2 = d\frac{n-3}{6}e$. Plugging these values into the expression for z we get the desired result. ⊓⊔

4 An E cient Algorithm and Computational Results

Although the Lovasz number can be computed in polynomial time, the available algorithms are far from simple and e cient (see, e.g., [1]). It is thus desirable to devise e cient algorithms tailored to the computation of # for special classes of graphs. By reduction to linear programming, the theta function of circulant graphs can be computed in linear time, provided that the number of cycles is independent of n. The corresponding algorithms are not necessarily e cient in practice, though. We briefly describe a practically e cient algorithm for computing $\#(C(n; j))$, i.e., in case of two cycles.

The algorithm rst determines the 2 lines forming the extremal vertex of Q_1 in the third quadrant, then solves the resulting 3 3 linear system (i.e., the system (3)). More precisely, the algorithm incrementally builds the intersection of the halfplanes which de ne Q_1 (considering only the third quadrant) and keeps track of the extremal point. The running time is $O(n \log n)$ in the worst case (i.e., it does not improve upon the optimal algorithms for computing the intersection of n arbitrary halfplanes or, equivalently, the convex hull of n points in the plane). However, it does make use of the properties of the lines bounding the halfplanes to keep the number of vertices of the incremental intersection close to the minimum possible. In some cases (such as $C(n;2)$) this is still $\Omega(n)$, but for most values of n and j it turns out to be substantially smaller.

Using the above algorithm we have performed some preliminary experiments to get insights about the behavior of the theta function for the special class of circulant graphs considered in this abstract. Actually, since the value sandwiched by the clique and the chromatic number of $C(n;j)$ is the theta function of $\overline{C}(n;j)$ (i.e., the complementary graph of $C(n;j)$), the results refer to $\#(\overline{C}(n;j)) = \frac{n}{\#(C(n;j))}$.

Table 4 shows $\#(\overline{C}(n;j))$ approximated to the four decimal place, for a number of values of n and j. It is immediate to note that, for a xed value of j, the values of the theta function seem to slowly approach, as n grows, the lower bound (given by the clique number), which happens to be 2 almost always (obvious exceptions occur when 3 divides n and $j = \frac{n}{3}$.

Table 1. Some computed values of $\#(\overline{C}(n;j))$

	4	5	6	7	$\lfloor\frac{n}{4}\rfloor$	$\lceil\frac{n}{4}\rceil$	$\lfloor\frac{n}{3}\rfloor$	$\lfloor\frac{n}{3}\rfloor + 1$	$\frac{n-3}{2}$
51	2:2446	2:0474	2:1227	2:0838	2:1297	2:2446	3	2:0173	2:2446
101	2.2383	2.0121	2.1122	2.0228	2.2383	2.1162	2.2383	2.0044	2.2383
201	2.2366	2.0031	2.1103	2.0059	2.2366	2.1113	3	2.0011	2.2366
301	2.2363	2.0014	2.1099	2.0027	2.2363	2.1099	2.0005	2.2363	2.2363
401	2.2362	2.0008	2.11	2.0015	2.2362	2.1102	2.2362	2.0003	2.2362
501	2.2362	2.0005	2.11	2.001	2.2362	2.1102	3	2.0002	2.2362
1001	2.2361	2.0001	2.1099	2.0002	2.2361	2.1099	2.2361	2	2.2361
2001	2.2361	2	2.1099	2.0001	2.2361	2.1099	3	2	2.2361
3001	2.2361	2	2.1099	2	2.2361	2.1099	2	2.2361	2.2361
4001	2.2361	2	2.1099	2	2.2361	2.1099	2.2361	2	2.2361
5001	2.2361	2	2.1099	2	2.2361	2.1099	3	2	2.2361
10001	2.2361	2	2.1099	2	2.2361	2.1099	2.2361	2	2.2361

This is con rmed by the results in Table 2, which depicts the behavior of the relative distance d_{nj} of $\#(\overline{C}(n;j))$ from the clique number. We only consider odd values of j (so that the clique number is always 2); we also rule out the cases where $j = \frac{n}{3}$, for which we know there is a (relatively) large gap between clique number and theta function. More precisely, Table 2 shows: (1) the maximum relative distance $M = \max_{j;n} d_{nj}$, where n ranges over all odd integers from 9 to n; (2) the average relative distance $= \frac{1}{N_n} \sum_{j;n} d_{nj}$, where N_n is the number of admissible pairs $(n;j)$; (3) the average quadratic distance $= \frac{1}{N_n} \sum_{j;n} (d_{nj} - \;)^2$.

The regularities presented by the value of the theta function and by the geometric structure of the optimal lines suggest the possibilities of further analytic investigations. For instance, we have observed that, for $j = 4$, the formula $i = \lfloor \frac{n}{2} \arccos \frac{-1-\sqrt{5}}{4} \rfloor$ seems to correctly predict the index of the rst optimal line, in perfect agreement with the experimental results. In general, for j even and $j << n$, up to a value |, the optimal point seems to always correspond

Table 2. Relative distances of #($\overline{C}(n; j)$) from the clique number

n	M		
101	0.372402	0.056077	0.004343
201	0.372402	0.033712	0.002600
301	0.372402	0.024840	0.001876
401	0.372402	0.019897	0.001471
501	0.372402	0.016734	0.001214
1001	0.372402	0.009657	0.000653

to two consecutive indices. For j odd, the rst line giving the optimal point is almost always obtained at the index $\frac{n-1}{2}$; the second line varies with j, but with a regular behaviour.

5 Conclusions

This paper has provided a rst step towards extending the class of graphs for whose theta function either a formula or a very e cient algorithm is available. Work in progress by the authors [5] aims at nding an e cient algorithm for more general circulant graphs. We believe that the results of this paper together with the above mentioned more general results will contribute to shedding new lights on the properties of this fascinating function.

References

[1] Alizadeh, F., Haeberly, J.-P. A., Nayakkankuppam, M., Overton, M., Schmieta, S.: SDPPACK User's Guide.
http://www.cs.nyu.edu/faculty/overton/sdppack/sdppack.html
[2] Alon, N.: On the Capacity of Digraphs. European J. Combinatorics, **19** (1998) 1{5
[3] Alon, N., Orlitsky, A.: Repeated Communication and Ramsey Graphs. IEEE Trans. on Inf. Theory, **41** (1995) 1276{1289
[4] Ashley, Siegel: A Note on the Shannon Capacity of Run-Length-Limited Codes. IEEE Trans. on Inf. Theory, **33** (1987)
[5] Brimkov, V.E., Codenotti, B., Crespi, V., Leoncini, M.: E cient Computation of the Lovasz Number of Circulant Graphs. In preparation
[6] Farber, M.: An Analogue of the Shannon Capacity of a Graph. SIAM J. on Alg. and Disc. Methods, **7** (1986) 67{72
[7] Feige, U.: Randomized Graph Products, Chromatic Numbers, and the Lovasz #-Function. Proc. of the 27th STOC (1995) 635{640
[8] Haemers, W.: An Upper Bound for the Shannon Capacity of a Graph. Colloq. Math. Soc. Janos Bolyai, **25** (1978) 267-272
[9] Haemers, W.: On Some Problems of Lovasz Concerning the Shannon Capacity of Graphs. IEEE Trans. on Inf. Theory, **25** (1979) 231{232

[10] Knuth, D.E.: The Sandwich Theorem. Electronic J. Combinatorics, **1** (1994)

[11] Lovasz, L.: On the Shannon Capacity of a Graph. IEEE Trans. on Inf. Theory, **25** (1979) 1{7

[12] O'Rourke, J.: Computational Geometry in C. Cambridge University Press (1994)

[13] Rosenfeld, M.: On a Problem of Shannon. Proc. Amer. Math. Soc., **18** (1967) 315{319

[14] Shannon, C.E.: The Zero-Error Capacity of a Noisy Channel. IRE Trans. Inform. Theory, **IT-2** (1956) 8{19

[15] Szegedy, M.: A Note on the # Number of Lovasz and the Generalized Delsarte Bound. Proc. of the 35th FOCS, (1994) 36{41

Appendix

In this appendix we shall prove Theorem 1. Before that, we need to establish the following subsidiary result.

Theorem 3. *Let n be odd and n 7. Also, as in Section 3, let C be the intersection of the halspaces de ned by the inequalities z 0 and $f_i(x;y) - z$ 0, $i = 1;:::;\frac{n-1}{2}$. Then, C is a polyhedral cone with the apex at the origin. The 1-dimensional faces of C (i.e., the edges of C) are the intersections (in the half-space z 0) of \consecutive" pairs of planes P_i, $P_{s(i)}$, where P_i is de ned by the equation $f_i(x;y) - z = 0$ and*

$$s(i) = \begin{array}{ll} i+1 & \text{if } i < \frac{n-1}{2}; \\ 1 & \text{otherwise.} \end{array}$$

Proof. It is su cient to show that (1) for any $z_0 > 0$, Q_{z_0} is bounded[2] (i.e., is a polygon), so that the polyhedron is indeed a pointed cone with the apex at the origin, and (2) Q_{z_0} has exactly $\frac{n-1}{2}$ vertices, formed by the intersections of pairs of consecutive lines l_i and $l_{s(i)}$, where l_i has equation $f_i(x;y) - z_0 = 0$, $i = 1;::;\frac{n-1}{2}$. To this end, we rst establish a couple of preliminary results.

Given any two intersecting lines l and l^0 in the 2D space, we will say that *l^0 is clockwise from l* if the two lines can be overlapped by rotating l clockwise around the intersection point by an angle of less than =2 radians.

Lemma 2. *For n 11, l_{i+1} is clockwise from l_i, $i = 1;:::;\frac{n-1}{2} - 1$.*

Proof. The equation de ning l_i can be written as $y = m_j x + q_i$, where $m_t = -\frac{\cos\frac{2\ t}{n}}{\cos\frac{4\ t}{n}}$. Thus $-\frac{}{2} < \ '_i = \text{arctg}(m_i) < \frac{}{2}$ is the angle between the positive x-axis and l_i. It is clearly su cient to prove the following statements.

1. If $'_i\,'_{i+1} > 0$ then $'_i > \ '_{i+1}$;
2. if $'_i > 0$ and $'_{i+1} < 0$ then $'_i - \ '_{i+1} < \frac{}{2}$;
3. if $'_i < 0$ and $'_{i+1} > 0$ then $'_{i+1} - \ '_i > \frac{}{2}$.

[2] Recall that Q_{z_0} is the projection of $C \bigcap fz = z_0g$ onto the xy plane.

It is not di cult to see that Condition 1 (i.e., $'_i\,'_{i+1} > 0$) occurs if and only if $\frac{}{4}(k-1) < \cos\ _i < \cos\ _{i+1} < \frac{}{4}k$, for some $k\ 2\ f1;2;3;4g$. Since the denominator of m_t does not vanish when $t\ 2\ [i;i+1]$, then m_t is a continuous function. We shall then prove that $'_i > '_{i+1}$ by showing that, for $t\ 2\ [i;i+1]$, m_t is a monotone decreasing function of t. Indeed, we have

$$\begin{aligned}\frac{\mathrm{d}m_t}{\mathrm{d}t} &= -\frac{2}{n}\frac{-\sin\ _t\cos 2\ _t + 2\cos\ _t\sin 2\ _t}{\cos^2 2\ _t}\\ &= \frac{2}{n}\frac{\sin\ _t\cos 2\ _t - 4\sin\ _t\cos^2\ _t}{\cos^2 2\ _t}\\ &= \frac{2}{n}\frac{\sin\ _t(\cos 2\ _t - 4\cos^2\ _t)}{\cos^2 2\ _t}\\ &= \frac{2}{n}\frac{\sin\ _t(2\cos^2\ _t - 1 - 4\cos^2\ _t)}{\cos^2 2\ _t}\\ &= \frac{2}{n}\frac{\sin\ _t(-1 - 2\cos^2\ _t)}{\cos^2 2\ _t} < 0:\end{aligned}$$

The proof of statements 2 and 3 becomes simpler if we assume that n be large enough (although only statement 3 requires that n 11 in order to hold true). Suppose rst that $'_i > 0$ and $'_{i+1} < 0$. This only happens if $\frac{2\ i}{n} < \frac{}{2} < \frac{2\ (i+1)}{n}$ (i.e., if both angles are close to $\frac{}{2}$). For n large enough this clearly means that both m_i and $-m_{i+1}$ are positive and close to zero, which in turn implies that $'_i - '_{i+1}$ is close to 0.
The proof of statement 3 is similar. The condition $'_i < 0$ and $'_{i+1} > 0$ occurs if either $\frac{2\ i}{n} < \frac{}{4} < \frac{2\ (i+1)}{n}$ or $\frac{2\ i}{n} < \frac{3}{4} < \frac{2\ (i+1)}{n}$. It both cases, $-m_i$ and m_{i+1} approach in nity as n grows, which means that $'_{i+1} - '_i$ approaches .

As an example, in Fig. 3 we see (following the clockwise order) that, for $n = 13$ and $i = 1;2;3$, the line l_{i+1} is indeed clockwise from l_i.

Lemma 3. *For $i = 1;:::;\frac{n-1}{2} - 1$, let $v_i = l_i \bigcap l_{i+1}$ denote the intersection point between l_{i-1} and l_i. Then any two points v_i and v_{i+1}, for $i = 1;:::;\frac{n-1}{2} - 2$, together with the origin form a clockwise circuit.*

Proof. It is well known (see, e.g., [12]) that three arbitrary points $a = (a_0;a_1)$, $b = (b_0;b_1)$, and $c = (c_0;c_1)$ form a clockwise circuit if and only if

$$d(a;b;c) = \begin{matrix} a_0 & a_1 & 1\\ b_0 & b_1 & 1\\ c_0 & c_1 & 1\end{matrix} < 0:$$

In our case $c_0 = c_1 = 0$, so that the above determinant simpli es to $a_0b_1 - a_1b_0$, where a_0 and a_1 (resp., b_0 and b_1) are the coordinates of v_i (resp., v_{i+1}). To determine a_0 and a_1 we can solve the 2 2 linear system (where, for simplicity, we have set $z_0 = 1$)

$$\begin{aligned} f_i(x;y) - 1 &= 0\\ f_{i+1}(x;y) - 1 &= 0;\end{aligned}$$

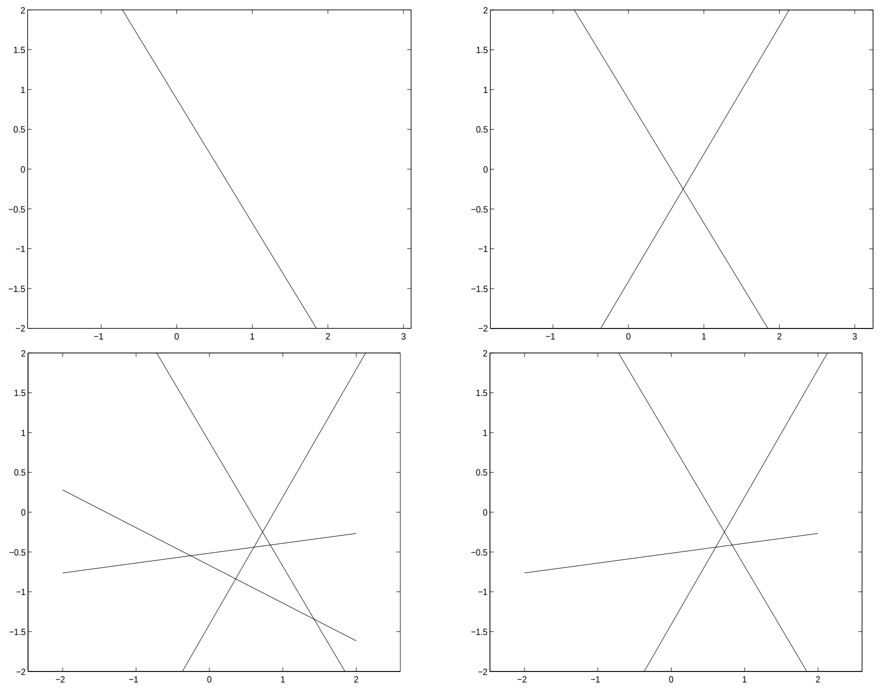

Fig. 3. Lines add in clockwise order ($n = 13$)

obtaining

$$x = \frac{\cos(2t(i+1)) - \cos(2ti)}{2(\cos(2t(i+1))\cos(ti) - \cos(2ti)\cos(t(i+1)))};$$

and

$$y = \frac{\cos(ti)\cos(2ti) - \cos(2ti)\cos(t(i+1))}{2\cos(2ti)(\cos(2t(i+1))\cos(ti) - \cos(2ti)\cos(t(i+1)))};$$

where $t = \frac{2}{n}$.

For v_{i+1} we clearly obtain similar values (the correspondence being is exactly given by replacing i with $i+1$ everywhere). After some simple but quite tedious algebra, the corresponding formula for the determinant simpli es to

$$d(v_i; v_{i+1}; O) = \frac{\cos(t)\cos(t(i+1)) - \cos(ti)}{(2\cos(ti)\cos(t(i+1)) + 1)(2\cos(t(i+1))\cos(t(i+2)) + 1)}:$$

We now prove that $d = d(v_i; v_{i+1}; O) < 0$. Consider the numerator of d. Since $\cos(ti) > \cos(t(i+1))$ (recall that $0 < ti < t(i+1) < t(i+2) <$), the numerator is clearly negative when $\cos(ti) > 0$. However, since $\cos(t)\cos(t(i+1)) - \cos(ti) = \cos(t(i+2)) - \cos(t)\cos(t(i+1))$ we can easily see that the numerator is also negative when $\cos(ti) < 0$. It remains to show that the denominator of d is positive. The denominator is the product of two terms, and the same argument applies to each of them. Clearly $2\cos(ti)\cos(t(i+1))+1>0$ when $\cos(ti)\cos(t(i+1))>0$. Hence the term might be negative only when $ti < \frac{}{2} < t(i+1)$. In this case, however, both angles are close to $\frac{}{2}$ and thus $j2\cos(ti)\cos(t(i+1))j$ is small compared to 1 (as in the proof of Lemma 2, this fact is obvious for large n, although it holds for any n 7). $\sqcap\!\!\!\!\sqcup$

We are now able to complete the proof of Theorem 3. As in Lemma 3, let v_i denote the intersection point of l_{i-1} and l_i, $i = 1;:::;\frac{n-1}{2}-1$. Also, let $v_{\frac{n-1}{2}}$ denote the intersection point of $l_{\frac{n-1}{2}}$ and l_1. By lemmas 2 and 3, we know that any three consecutive vertices of the closed polygon $L=[v_1;:::;v_{\frac{n-1}{2}}]$ make a right turn (except, possibly, for the two triples which include $v_{\frac{n-1}{2}}$ and v_1). We also know that the angle $'_i$, as a function of i, changes sign three times only, starting from a negative value for $i=1$. Hence the polygon L may possibly have the three shapes depicted in Fig. 4: (1) L is convex, (2) L is simple but not convex, (3) L is not simple.

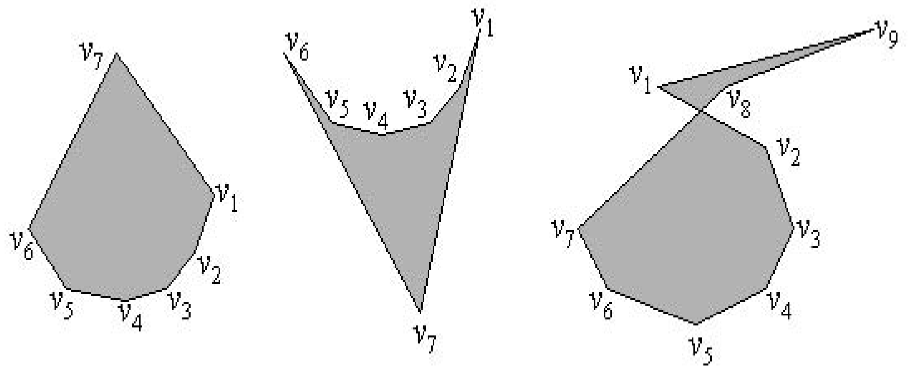

Fig. 4. Three possible forms for the polygon L of Theorem 3

Case 2 would clearly correspond to Q_{z_0} being unbounded, while case 3 would imply that the number of vertices of Q_{z_0} is less than $\frac{n-1}{2}$ and that not all of them are formed by the intersection of consecutive lines. Hence, to prove the result, we have to prove that only Case 1 indeed occurs. But this is easy. In fact, cases 2 and 3 can be ruled out simply by observing that that the three points $v_{\frac{n-1}{2}}$, v_1, and O make a left turn, while in case 1 they make a right turn. Computing the appropriate determinant d we get

$$d = \frac{-(2\ \ -1)(\ \ +1)(4\ ^2-2\ \ -1)}{2(4\ ^3-2\ \ -1)(32\ ^6-48\ ^4+20\ ^2-1)};$$

where $\ \ = \cos\frac{}{n}$. Now, the numerator is negative for $x > :8090169945$ (the largest root of $4x^2-2x-1=0$) while the denominator is positive for $x >$

:8846461772 (the unique real root of $4x^3 - 2x - 1 = 0$). But for $n = 7$ we already have = :9009688678; hence $d < 0$ for any n 7 and the three points make a right turn, as required.

As the last observation, we recall that the proof holds for odd n 11 (because of Lemma 2). However, the result is true for any odd n 7, as can be seen by directly checking the cases $n = 7$ and $n = 9$ (see Fig. 5).

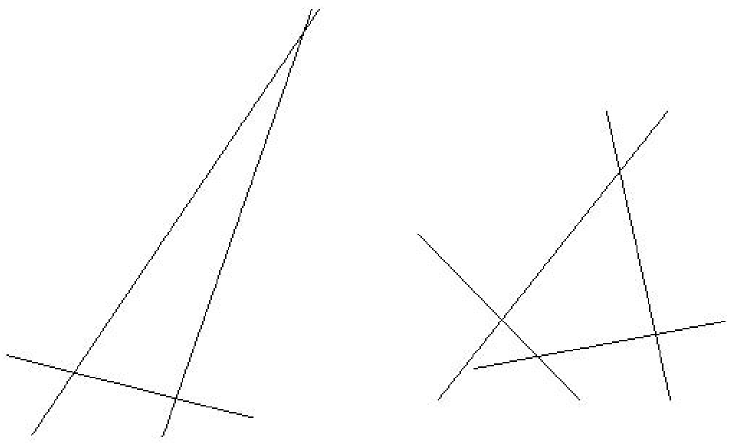

Fig. 5. Projection Q_{z_0} for $n = 7$ (left) and $n = 9$

Proof of Theorem 1. Consider the linear system (3) with $j = 2$. By Theorem 3, we know that $= \frac{2\ i}{n}$ and $= \frac{2\ (i+1)}{n}$, for some i 2 1; :::; $\frac{n-1}{2} - 1$. The solution to (3) is given by $x = \frac{n}{2}\frac{\cos\ + \cos}{(\cos\ - 1)(\cos\ - 1)}$, $y = \frac{-n}{4}\frac{1}{(\cos\ - 1)(\cos\ - 1)}$, and $z = n + 2x + 2y = n\ 1 - \frac{\frac{1}{2} - \cos\ - \cos}{(\cos\ - 1)(\cos\ - 1)}$. Our goal is now to determine the value of i which minimizes z. More precisely, we will compute the minimum, over the set 1; 2; :::; $\frac{n-1}{2} - 1$, of the following function

$$g_n(x) = \frac{\cos\frac{2\ x}{n} + \cos\frac{2\ (x+1)}{n} - \frac{1}{2}}{(\cos\frac{2\ x}{n} - 1)(\cos\frac{2\ (x+1)}{n} - 1)}:$$

$g_n(x)$ is a continuous function in the open interval (0; 2), with $\lim_{x\,!\,0^+} = \lim_{x\,!\,2\ -} = +1$. Computing the derivative we obtain

$$g_n^0(x) = -\frac{\csc\frac{x}{n}\csc\frac{(1+x)}{n}^{\ 3}}{8n}h_n(x);$$

where $h_n(x) = \sin\frac{(2x-1)}{n} + \sin\frac{(2x+3)}{n} + \sin\frac{(4x+1)}{n} + \sin\frac{(4x+3)}{n}$. Clearly, $g_n^0(x) = 0$ if and only if $h_n(x) = 0$. As a rst rough argument (which is useful just to locate the zero $\bar{x}$ of h_n), we note that $h_n(x) > 0$ if $\frac{(4x+3)}{n}$. This implies that $\bar{x}$ must be greater than $\frac{n-3}{4}$. But then, for n large enough, we may \approximate" $h_n(x)$ with $h_n(x) = 2\sin\frac{2x}{n} + 2\sin\frac{4x}{n}$ which vanishes at $x = \frac{n}{3}$. So $\bar{x}$ $\frac{n}{3}$ and we see that $\frac{}{2} < 2\bar{x} - 1 < 2\bar{x} + 3 <$ and $< 4\bar{x} + 1 < 4\bar{x} + 3 < \frac{3}{2}$. We now use this result to obtain tight bounds for $\bar{x}$. We observe that $h_n(x)$ is positive if

$$\sin\ 2\ - \frac{(4x+3)}{n} = \max\ \sin\frac{(4x+3)}{n}\ ;\ \sin\frac{(4x+1)}{n}$$
$$< \min\ \sin\frac{(2x-1)}{n}; \sin\frac{(2x+3)}{n}$$
$$= \sin\frac{(2x+3)}{n};$$

which amounts to saying that $\bar{x}$ cannot be less than $\frac{n}{3} - 1$. Analogously, $h_n(x)$ is negative if

$$\sin\ 2\ - \frac{(4x+1)}{n} = \min\ \sin\frac{(4x+3)}{n}\ ;\ \sin\frac{(4x+1)}{n}$$
$$> \max\ \sin\frac{(2x-1)}{n}; \sin\frac{(2x+3)}{n}$$
$$= \sin\frac{(2x-1)}{n};$$

which implies that $\bar{x}$ cannot be larger than $\frac{n}{3}$. This fact allows us to conclude that the integer value which minimizes $g_n(x)$ (and hence z) is one among $b\frac{n}{3}c - 1$, $b\frac{n}{3}c$ and $d\frac{n}{3}e$. We now prove that the value sought is $b\frac{n}{3}c$ by showing that $g_n(b\frac{n}{3}c - 1) - g_n(b\frac{n}{3}c)$ and $g_n(d\frac{n}{3}e) - g_n(b\frac{n}{3}c)$ are both positive. For simplicity, we shall use the following notation: $f_{bc-1} = \cos\frac{2\ (bn{=}3c-1)}{n}$, $f_{bc} = \cos\frac{2\ (bn{=}3c)}{n}$, $f_{de} = \cos\frac{2\ (dn{=}3e)}{n}$, and $f_{de+1} = \cos\frac{2\ (dn{=}3e+1)}{n}$. We have

$$g_n(d\frac{n}{3}e) - g_n(b\frac{n}{3}c) = \frac{f_{de} + f_{de+1} - 1{=}2}{(f_{de}-1)(f_{de+1}-1)} - \frac{f_{bc} + f_{de} - 1{=}2}{(f_{bc}-1)(f_{de}-1)}$$
$$= \frac{f_{de}f_{bc} - f_{de} + f_{de+1}f_{bc} - f_{de+1} - \frac{1}{2}f_{bc} + \frac{1}{2}}{(f_{bc}-1)(f_{de}-1)(f_{de+1}-1)} -$$
$$\frac{f_{bc}f_{de+1} - f_{bc} + f_{de}f_{de+1} - f_{de} - \frac{1}{2}f_{de+1} + \frac{1}{2}}{(f_{bc}-1)(f_{de}-1)(f_{de+1}-1)}$$
$$= \frac{(f_{bc} - f_{de+1})(\frac{1}{2} + f_{de})}{(f_{bc}-1)(f_{de}-1)(f_{de+1}-1)}:$$

The last expression is positive since the denominator is negative, $f_{bc} - f_{de+1} > 0$, and $f_{de} < -\frac{1}{2}$. Similarly,

$$g_n(b\frac{n}{3}c - 1) - g_n(b\frac{n}{3}c) = \frac{f_{bc-1} + f_{bc} - 1{=}2}{(f_{bc-1}-1)(f_{bc}-1)} - \frac{f_{bc} + f_{de} - 1{=}2}{(f_{bc}-1)(f_{de}-1)}$$

$$= \frac{f_{bc-1}f_{de} - f_{bc-1} + f_{bc}f_{de} - f_{bc} - \frac{1}{2}f_{de} + \frac{1}{2}}{(f_{bc-1} - 1)(f_{bc} - 1)(f_{de} - 1)} - \frac{f_{bc}f_{bc-1} - f_{bc} + f_{de}f_{bc-1} - f_{de} - \frac{1}{2}f_{bc-1} + \frac{1}{2}}{(f_{bc-1} - 1)(f_{bc} - 1)(f_{de} - 1)}$$

$$= \frac{(f_{de} - f_{bc-1})(\frac{1}{2} + f_{bc})}{(f_{bc-1} - 1)(f_{bc} - 1)(f_{de} - 1))};$$

and again the numerator is negative since $f_{de} - f_{bc-1} < 0$ and $f_{bc} > -\frac{1}{2}$. $\square$

Speeding Up Pattern Matching by Text Compression

Yusuke Shibata[1], Takuya Kida[1], Shuichi Fukamachi[2], Masayuki Takeda[1], Ayumi Shinohara[1], Takeshi Shinohara[2], and Setsuo Arikawa[1]

[1] Dept. of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
`fyusuke, kida, takeda, ayumi, arikawag@i.kyushu-u.ac.jp`
[2] Dept. of Arti cial Intelligence, Kyushu Institute of Technology,
Iizuka 320-8520, Japan
`ffukamati, shinog@ai.kyutech.ac.jp`

Abstract. Byte pair encoding (BPE) is a simple universal text compression scheme. Decompression is very fast and requires small work space. Moreover, it is easy to decompress an arbitrary part of the original text. However, it has not been so popular since the compression is rather slow and the compression ratio is not as good as other methods such as Lempel-Ziv type compression.
In this paper, we bring out a potential advantage of BPE compression. We show that it is very suitable from a practical view point of *compressed pattern matching*, where the goal is to nd a pattern directly in compressed text without decompressing it explicitly. We compare running times to nd a pattern in (1) BPE compressed les, (2) Lempel-Ziv-Welch compressed les, and (3) original text les, in various situations. Experimental results show that pattern matching in BPE compressed text is even faster than matching in the original text. Thus the BPE compression reduces not only the disk space but also the searching time.

1 Introduction

Pattern matching is one of the most fundamental operations in string processing. The problem is to nd all occurrences of a given pattern in a given text. A lot of classical or advanced pattern matching algorithms have been proposed (see [8,1]). The time complexity of pattern matching algorithm is measured by the number of symbol comparisons between pattern and text symbols. The Knuth-Morris-Pratt (KMP) algorithm [19] is the rst one which runs in linear time proportional to the sum of the pattern length m and the text length n. The algorithm requires additional memory proportional to the pattern length m. One interesting research direction is to develop an algorithm which uses only constant amount of memory, preserving the linear time complexity (see [11,7,5,13,12]). Another important direction is to develop an algorithm which makes a sublinear number of comparisons on the average, as in the Boyer-Moore (BM) algorithm [4] and its variants (see [24]). The lower bound of the average case time complexity is known to be $O(n \log m{=}m)$ [27], and this bound is achieved by the algorithm presented in [6].

From a practical viewpoint, the constant hidden behind O-notation plays an important role. Horspool's variant [14] and Sunday's variant [22] of the BM algorithm are widely known to be very fast in practice. In fact, the former is incorporated into a software package `Agrep`, which is understood as the fastest pattern matching tool developed by Wu and Manber [25].

Recently, a new trend for accelerating pattern matching has emerged: *speeding up pattern matching by text compression*. It was rst introduced by Manber [20]. Contrary to the traditional aim of text compression | to reduce space requirement of text les on secondary disk storage devices | , text is compressed in order to speed up the pattern matching process.

It should be mentioned that the problem of pattern matching in compressed text without decoding, which is often referred to as *compressed pattern matching*, has been studied extensively in this decade. The motivation is to investigate the complexity of this problem for various compression methods from the viewpoint of combinatorial pattern matching. It is theoretically interesting, and in practice some algorithms proposed are indeed faster than a regular decompression followed by a simple search. In fact, Kida et al. [18,17] and Navarro et al. [21] independently presented compressed pattern matching algorithms for the Lempel-Ziv-Welch (LZW) compression which run faster than a decompression followed by a search. However, the algorithms are slow in comparison with pattern matching in uncompressed text if we compare the CPU time. In other words, the LZW compression did not speed up the pattern matching.

When searching text les stored in secondary disk storage, the running time is the sum of le I/O time and CPU time. Obviously, text compression yields a reduction in the le I/O time at nearly the same rate as the compression ratio. However, in the case of an adaptive compression method, such as Lempel-Ziv family (LZ77, LZSS, LZ78, LZW), a considerable amount of CPU time is devoted to an extra e ort to keep track of the compression mechanism. In order to reduce both of le I/O time and CPU time, we have to nd out a compression scheme that requires no such extra e ort. Thus we must re-estimate the performance of existing compression methods or develop a new compression method in the light of the new criterion: *the time for nding a pattern in compressed text directly*.

As an e ective tool for such re-estimation, we introduced in [16] a unifying framework, named *collage system*, which abstracts various dictionary-based compression methods, such as Lempel-Ziv family, and the static dictionary methods. We developed a general compressed pattern matching algorithm for strings described in terms of collage system. Therefore, any of the compression methods that can be described in the framework has a compressed pattern matching algorithm as an instance.

Byte pair encoding (BPE, in short) [10], included in the framework of collage systems, is a simple universal text compression scheme based on the pattern-substitution [15]. The basic operation of the compression is to substitute a single character which did not appear in the text for a pair of consecutive two characters which frequently appears in the text. This operation will be repeated until either all characters are used up or no pair of consecutive two characters appears frequently. Thus the compressed text consists of two parts: the substitution table, and the substituted text. Decompression is very fast and requires small work space. Moreover, partial decompression is possible, since the compression depends only on the substitution. This is a big advantage of BPE in comparison with adaptive dictionary based methods. Despite such advantages, the BPE method has received little attention, until now. The reason for this is mainly the following two disadvantages: the compression is terribly slow, and the compression ratio is not as good as other methods such as Lempel-Ziv type compression.

In this paper, we pull out a potential advantage of BPE compression, that is, we show that BPE is very suitable for speeding up pattern matching. Manber [20] also introduced a little simpler compression method. However since its compression ratio is not so good and is about 70% for typical English texts, the improvement of the searching time cannot be better than this rate. The compression ratio of BPE is about 60% for typical English texts, and is near 30% for biological sequences. We propose a compressed pattern matching algorithm which is basically an instance of the general one mentioned above. Experimental results show that, in CPU time comparison, the performance of the proposed algorithm running on BPE compressed les of biological sequences is better than that of Agrep running on uncompressed le of the same sequences. This is not the case for English text les. Moreover, the results show that, in elapsed time comparison, the algorithm drastically defeats Agrep even for English text les.

It should be stated that Moura et al. [9] proposed a compression scheme that uses a word-based Hu man encoding with a byte-oriented code. The compression ratio for typical English texts is about 30%. They presented a compressed pattern matching algorithm and showed that it is twice faster than Agrep on uncompressed text in the case of exact match. However, the compression method is not applicable to biological sequences because they cannot be segmented into words. For the same reason, it cannot be used for natural language texts written in Japanese in which we have no blank symbols between words.

Recall that the key idea of the Boyer-Moore type algorithms is to skip symbols of text, so that they do not read all the text symbols on the average. The algorithms are intended to avoid ‘redundunt’ symbol comparisons. Analogously, our algorithm also skips symbols of text in the sense that more than one symbol is encoded as one character code. In other words, our algorithm avoids processing of redundant information about text. Note that the redundancy varies depending on the pattern in the case of the Boyer-Moore type algorithms, whereas it depends only on the text in the case of speeding up by compression.

The rest of the paper is organized as follows. In Section 2, we introduce the byte pair encoding scheme, discuss its implementation, and estimate its performance in comparison with Compress and Gzip. Section 3 is devoted to compressed pattern matching in BPE compressed les, where we have two implementations using the automata and the bit-parallel approaches. In Section 4, we report our experimental results to compare practical behaviors of these algorithms performed. Section 5 concludes the discussion and explains some of future works.

2 Byte Pair Encoding

In this section we describe the byte pair encoding scheme, discuss its implementation, and then estimate the performance of this compression scheme in comparison with widely-known compression tools `Compress` and `Gzip`.

2.1 Compression Algorithm

The BPE compression is a simple version of pattern-substitution method [10]. It utilizes the character codes which did not appear in the text to represent frequently occurring strings, namely, strings of which frequencies are greater than some threshold. The compression algorithm repeats the following task until all character codes are used up or no frequent pairs appear in the text:

> *Find the most frequent pair of consecutive two character codes in the text, and then substitute an unused code for the occurrences of the pair.*

For example, suppose that the text to be compressed is

$$T_0 = \mathtt{ABABCDEBDEFABDEABC}.$$

Since the most frequent pair is AB, we substitute a code G for AB, and obtain the new text

$$T_1 = \mathtt{GGCDEBDEFGDEGC}.$$

Then the most frequent pair is DE, and we substitute a code H for it to obtain

$$T_2 = \mathtt{GGCHBHFGHGC}.$$

By substituting a code I for GC, we obtain

$$T_3 = \mathtt{GIHBHFGHI}.$$

The text length is shorten from $|T_0| = 18$ to $|T_3| = 9$. Instead we have to encode the substitution pairs $\mathtt{AB} \to \mathtt{G}$, $\mathtt{DE} \to \mathtt{H}$, and $\mathtt{GC} \to \mathtt{I}$.

More precisely, we encode a table which stores for every character code what it represents. Note that a character code can represent either (1) the character itself, (2) a code-pair, or (3) nothing. Let us call such table *substitution table*. In practical implementations, an original text file is split into a number of fixed-size blocks, and the compression algorithm is then applied to each block. Therefore a substitution table is encoded for each block.

2.2 Speeding Up of Compression

In [10] an implementation of BPE compression is presented, which seems quite simple. It requires $O(‘N)$ time, where N is the original text length and ‘ is the number of character codes. The time complexity can be improved into $O(‘ + N)$ by using a relatively simple technique, but this improvement did not reduce the compression time in practice. Thus, we decided to reduce the compression time with sacri ces in the compression ratio.

The idea is to use a substitution table obtained from a small part of the text (e.g. the rst block) for encoding the whole text. The disadvantage is that the compression ratio decreases when the frequency distribution of character pairs varies depending on parts of the text. The advantage is that a substitution table is encoded only once. This is a desirable property from a practical viewpoint of compressed pattern matching in the sense that we have to perform only once any task which depends on the substitution table as a preprocessing since it never changes.

Fast execution of the substitutions according to the table is achieved by an e cient multiple key replacement technique [2,23], in which a one-way sequential transducer is built from a given collection of replacement pairs which performs the task in only one pass through a text. When the keys have overlaps, it replaces the longest possible rst occurring key. The running time is linear in the total length of the original and the substituted text.

2.3 Comparison with `Compress` and `Gzip`

We compared the performance of BPE compression with those of `Compress` and `Gzip`. We implemented the BPE compression algorithm both in the standard way described in [10] and in the modi ed way stated in Section 2.2. The `Compress` program has an option to specify in bits the upper bound to the number of strings in a dictionary, and we used `Compress` with speci cation of 12 bits and 16 bits. Thus we tested ve compression programs.

We estimated the compression ratios of the ve compression programs for the four texts shown in Table 1. The results are shown in Table 2. We can see that the compression ratios of BPE are worse than those of `Compress` and `Gzip`,

Table 1. Four Text Files.

le	annotation
Brown corpus (6.4 Mbyte)	A well-known collection of English sentences, which was compiled in the early 1960s at Brown University, USA.
Medline (60.3 Mbyte)	A clinically-oriented subset of Medline, consisting of 348,566 references.
Genbank1 (43.3 Mbyte)	A subset of the GenBank database, an annotated collection of all publicly available DNA sequences.
Genbank2 (17.1 Mbyte)	The le obtained by removing all elds other than accession number and nucleotide sequence from the above one.

especially for English texts. We also estimated the CPU times for compression and decompression. Although we omit here the results because of lack of space, we observed that the BPE compression was originally very slow, and it is drastically accelerated by the modi cation stated in Section 2.2. In fact, the original BPE compression is 4 5 times slower than `Gzip`, whereas the modi ed one is 4 5 times faster than `Gzip` and is competitive with `Compress` with 12 bit option.

Thus, BPE is not so good from the traditional criteria. This is the reason why it has received little attentions, until now. However, it has the following properties which are quite attractive from the practical viewpoint of compressed pattern matching: (1) No bit-wise operations are required since all the codes are of 8 bits; (2) Decompression requires very small amount of memory; and (3) Partial decompression is possible, that is, we can decompress any portion of compressed text.

In the next section, we will show how we can perform compressed pattern matching e ciently in the case of BPE compression.

Table 2. Compression Ratios (%).

		BPE		Compress		Gzip
		standard	modi ed	12bit	16bit	
Brown corpus	(6.8Mb)	51.08	59.02	51.67	43.75	39.04
Medline	(60.3Mb)	56.20	59.07	54.32	42.34	33.35
Genbank1	(43.3Mb)	46.79	51.36	43.73	32.55	24.84
Genbank2	(17.1Mb)	30.80	32.50	29.63	26.80	23.15

3 Pattern Matching in BPE Compressed Texts

For searching a compressed text, the most naive approach would be the one which applies any string matching routine with expanding the original text on the fly. Another approach is to encode a given pattern and apply any string matching routine in order to nd the encoded pattern directly in the compressed text. The problem in this approach is that the encoded pattern is not unique. A solution due to Manber [20] was to devise a way to restrict the number of possible encodings for any string.

The approach we take here is basically an instance of the general compressed pattern matching algorithm for strings described in terms of collage system [16]. As stated in Introduction, collage system is a unifying framework that abstracts most of existing dictionary-based compression methods. In the framework, a string is described by a pair of a dictionary D and a sequence S of tokens representing phrases in D. A dictionary D is a sequence of assignments where the basic operations are concatenation, repetition, and prefix (suffix) truncation. A text compressed by BPE is described by a collage system with no truncation operations. For a collage system with no truncation, the general compressed pattern matching algorithm runs in $O(\|D\|+|S|+m^2+r)$ time using $O(\|D\|+m^2)$ space, where $\|D\|$ denotes the size of the dictionary D and $|S|$ is the length of the sequence S.

The basic idea of the general algorithm is to simulate the move of the KMP automaton for input D and S. Note that one token of sequence S may represent a string of length more than one, which causes a series of state transitions. The idea is to substitute just one state transition for each such consecutive state transitions. More formally, let $\delta : Q \times \Sigma \to Q$ be the state transition function of the KMP automaton, where Σ is the alphabet and Q is the set of states. Extend δ into the function $\hat{\delta} : Q \times \Sigma^* \to Q$ by

$$\hat{\delta}(q, \varepsilon) = q \qquad \text{and} \qquad \hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a),$$

where $q \in Q$, $u \in \Sigma^*$, and $a \in \Sigma$. Let D be the set of phrases in dictionary. Let *Jump* be the limitation of $\hat{\delta}$ to the domain $Q \times D$.

By identifying a token with the phrase it represents, we can define the new automaton which takes as input a sequence of tokens and makes state transition by using *Jump*. The state transition of the new machine caused by a token corresponds to the consecutive state transitions of the KMP automaton caused by the phrase represented by the token. Thus, we can simulate the state transitions of the KMP automaton by using the new machine. However, the KMP automaton may pass through the final state during the consecutive transitions. Hence the new machine should be a Mealy type sequential machine with output function *Output* $: Q \times D \to 2^N$ defined by

$$Output(q, u) = \{i \in N \mid 1 \le i \le |u| \text{ and } \hat{\delta}(q, u[1..i]) \text{ is the final state}\},$$

where N denotes the set of natural numbers, and $u[1..i]$ denotes the length i prefix of string u.

In [16] efficient realizations of the functions *Jump* and *Output* were discussed for general case. In the case of BPE compression, a simpler implementation is possible. We take two implementations. One is to realize the state transition function *Jump* defined on $Q \times D$ as a two-dimensional array of size $|Q| \times |D|$. The array size is not critical since the number of phrases in D is at most 256 in BPE compression. This is not the case with LZW, in which $|D|$ can be the compressed text size.

Another implementation is the one utilizing the bit parallel paradigm in a similar way that we did for LZW compression [17]. Technical details are omitted because of lack of space.

4 Experimental Results

We estimated the running time of the proposed algorithms running on BPE compressed files. We tested the two implementations mentioned in the previous section. For comparisons, we tested the algorithm [17] in searching LZW compressed les. We also tested the KMP algorithm, the Shift-Or algorithm [26,3], and Agrep (the Boyer-Moore-Horpspool algorithm) in searching uncompressed les. The performance of the BM type algorithm strongly depends upon the pattern length m, and therefore the running time of Agrep was tested for $m = 4; 8; 16$. The performance of each algorithm other than Agrep is independent of the pattern length. The text les we used are the same as the four text les mentioned in Section 2. The machine used is a PC with a Pentium III processor at 500MHz running TurboLinux 4.0 operating system. The data transfer speed was about 7.7 Mbyte/sec.

The results are shown in Table 3, where we included the preprocessing time. In this table, (a) and (b) stand for the automata and the bit-parallel implementations stated in the previous section, respectively.

Table 3. Performance Comparisons.

		BPE		LZW	uncompressed				
		(a)	(b)	[17]	KMP	Shift-Or	Agrep		
							$m = 4$	$m = 8$	$m = 16$
CPU time (sec)	Brown Corpus	0.09	0.16	0.94	0.13	0.11	0.09	0.07	0.07
	Medline	1.03	1.43	6.98	1.48	1.28	0.85	0.69	0.63
	Genbank1	0.52	0.89	4.17	0.81	0.76	0.72	0.58	0.53
	Genbank2	0.13	0.22	1.33	0.32	0.29	0.27	0.32	0.32
elapsed time (sec)	Brown Corpus	0.59	0.54	1.17	0.91	1.01	0.91	0.90	0.90
	Medline	4.98	4.95	7.53	8.38	8.26	8.01	7.89	7.99
	Genbank1	3.04	2.95	4.48	6.26	6.32	6.08	5.67	5.64
	Genbank2	0.76	0.73	1.46	2.28	2.33	2.19	2.18	2.14

First of all, it is observed that, in CPU time comparison, the automata-based implementation of the proposed algorithm in searching BPE compressed le is faster than each of the routines except Agrep. Comparing with Agrep, it is good for Genbank1 and Genbank2, but not so for other two les. The reason for this is that the performance of the proposed algorithm depends on compression ratio. Recall that the compression ratios for Genbank1 and Genbank2 are relatively high in comparison with those of Brown corpus and Medline.

From a practical viewpoint, the running speed in elapsed time is also important, although it is not easy to measure accurate values of elapsed time. Table 3 implies that the proposed algorithm is the fastest in the elapsed time comparison.

5 Conclusion

We have shown potential advantages of BPE compression from a viewpoint of compressed pattern matching.

The number of tokens in BPE is limited to 256 so that all the tokens are encoded in 8 bits. The compression ratio can be improved if we raise the limitation to the number of tokens. A further improvement is possible by using variable-length codewords. However, it is preferable to use xed-length codewords with 8 bits from the viewpoint of compressed pattern matching since we want to keep the search on a byte level for e ciency.

One future direction of this study will be to develop approximate pattern matching algorithms for BPE compressed text.

References

1. A. Apostolico and Z. Galil. *Pattern Matching Algorithm*. Oxford University Press, New York, 1997.
2. S. Arikawa and S. Shiraishi. Pattern matching machines for replacing several character strings. *Bulletin of Informatics and Cybernetics*, 21(1{2):101{111, 1984.
3. R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Comm. ACM*, 35(10):74{82, 1992.
4. R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. ACM*, 20(10):62{72, 1977.
5. D. Breslauer. Saving comparisons in the Crochemore-Perrin string matching algorithm. In *Proc. of 1st European Symp. on Algorithms*, pages 61{72, 1993.
6. M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string-matching algorithm. *Algorithmica*, 12(4/5):247{267, 1994.
7. M. Crochemore and D. Perrin. Two-way string-matching. *J. ACM*, 38(3):651{675, 1991.
8. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.
9. E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Direct pattern matching on compressed text. In *Proc. 5th International Symp. on String Processing and Information Retrieval*, pages 90{95. IEEE Computer Society, 1998.
10. P. Gage. A new algorithm for data compression. *The C Users Journal*, 12(2), 1994.
11. Z. Galil and J. Seiferas. Time-space-optimal string matching. *J. Comput. System Sci.*, 26(3):280{294, 1983.
12. L. Gasieniec, W. Plandowski, and W. Rytter. Constant-space string matching with smaller number of comparisons: Sequential sampling. In *Proc. 6th Ann. Symp. on Combinatorial Pattern Matching*, pages 78{89. Springer-Verlag, 1995.
13. L. Gasieniec, W. Plandowski, and W. Rytter. The zooming method: a recursive approach to time-space e cient string-matching. *Theoret. Comput. Sci*, 147(1/2):19{30, 1995.
14. R. N. Horspool. Practical fast searching in strings. *Software-Practice and Experience*, 10:501{506, 1980.
15. G. C. Jewell. Text compaction for information retrieval. *IEEE SMC Newsletter*, 5, 1976.
16. T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. A unifying framework for compressed pattern matching. In *Proc. 6th International Symp. on String Processing and Information Retrieval*, pages 89{96. IEEE Computer Society, 1999.

17. T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Shift-And approach to pattern matching in LZW compressed text. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, pages 1{13. Springer-Verlag, 1999.
18. T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In J. A. Atorer and M. Cohn, editors, *Proc. Data Compression Conference '98*, pages 103{112. IEEE Computer Society, 1998.
19. D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput*, 6(2):323{350, 1977.
20. U. Manber. A text compression scheme that allows fast searching directly in the compressed le. In *Proc. Combinatorial Pattern Matching*, volume 807 of *Lecture Notes in Computer Science*, pages 113{124. Springer-Verlag, 1994.
21. G. Navarro and M. Ra not. A general practical approach to pattern matching over Ziv-Lempel compressed text. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, pages 14{36. Springer-Verlag, 1999.
22. D. M. Sunday. A very fast substring search algorithm. *Comm. ACM*, 33(8):132{142, 1990.
23. M. Takeda. An e cient multiple string replacing algorithm using patterns with pictures. *Advances in Software Science and Technology*, 2:131{151, 1990.
24. B. W. Watson and G. Zwaan. A taxonomy of sublinear multiple keyword pattern matching algorithms. *Sci. of Comput. Programing.*, 27(2):85{118, 1996.
25. S. Wu and U. Manber. Agrep { a fast approximate pattern-matching tool. In *Usenix Winter 1992 Technical Conference*, pages 153{162, 1992.
26. S. Wu and U. Manber. Fast text searching allowing errors. *Comm. ACM*, 35(10):83{91, October 1992.
27. A. C.-C. Yao. The complexity of pattern matching for a random string. *SIAM J. Comput.*, 8(3):368{387, 1979.

Author Index