Lecture Notes in Computer Science 1518
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

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

Michael Luby José Rolim
Maria Serna (Eds.)

Randomization and Approximation Techniques in Computer Science

Second International Workshop, RANDOM'98
Barcelona, Spain, October 8-10, 1998
Proceedings

Springer

Series Editors

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

Volume Editors

Michael Luby
International Computer Science Institute
1947 Center Street, Suite 600, Berkeley, CA 94704-1198, USA
E-mail: luby@icsi.berkeley.edu

José D.P. Rolim
University of Geneva, Computer Science Department
24, rue Général Dufour, CH-1211 Geneva 4, Switzerland
E-mail: rolim@cui.unige.ch

Maria Serna
University of Barcelona (UPC)
Jordi Girona Salgado 1-3, E-08034 Barcelona, Spain
E-mail: mjserna@lsi.upc.es

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Randomization and approximation techniques in computer science : second international workshop, RANDOM '98, Barcelona, Spain, October 8 - 10, 1998 ; proceedings / Michael Luby . . . (Hrsg.). - Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1998
(Lecture notes in computer science ; Vol. 1518)
ISBN 3-540-65142-X

CR Subject Classification (1991): F.2, G.1.2, G.1.6, G.2, G.3, E.1, I.3.5

ISSN 0302-9743
ISBN 3-540-65142-X Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10692778 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Foreword

The Workshop on *Randomization and Approximation Techniques in Computer Science*, **Random'98**, focuses on algorithmic and complexity aspects arising in the development of efficient randomized solutions to computationally difficult problems. It aims, in particular, at fostering the cooperation among practitioners and theoreticians and among algorithmic and complexity researchers in the field. RANDOM'98, held at the University of Barcelona (UPC), October 8–10, 1998, is the second in the series, after Bologna.

This volume contains all contributed papers accepted for presentation at the workshop, together with invited lectures by Josep Díaz (UPC Barcelona), Alan M. Frieze (Carnegie Mellon U.), Michael Luby (ICSI Berkeley), and Emo Welzl (ETH Zürich). The contributed papers were selected out of several dozen submissions received in response to the call for papers. All papers published in the workshop proceedings were selected by the program committee on the basis of referee reports. Considerable effort was devoted to the evaluation of the submissions by the program committee and a number of other referees. Extensive feedback was provided to authors as a result, which we hope has proven helpful to them.

We would like to thank all of the authors who responded to the call for papers, our invited speakers, the referees, and the members of the program committee:

Michael Luby, Chair, ICSI Berkeley
Andrei Broder, Digital Systems Research Center
Bernard Chazelle, Princeton U.
Andrea Clementi, U. of Rome
Anna Karlin, U. of Washington
Richard Karp, U. of Washington
Claire Kenyon, U. of Paris Sud
Michael Mitzenmacher, Digital Systems Research Center
Rajeev Motwani, Stanford U.
Prabhakar Raghavan, IBM
Maria Serna, UPC Barcelona
Alistair Sinclair, U. of California, Berkeley
Madhu Sudan, MIT
Avi Wigderson, Hebrew U.
Peter Winkler, Bell Labs

We gratefully acknowledge support from the European Association *INTAS*, the Comissionat per a Universitats i Recerca – Generalitat de Catalunya, and Universitat Politècnica de Catalunya. Finally, we would like to thank Helena Martinez, Carme Alvarez, Conrado Martinez, and Jordi Petit i Silvestre for their help in the preparation of the meeting.

August 1998 Michael Luby, José D. P. Rolim, Maria J. Serna

Contents

Disjoint Paths in Expander Graphs via Random Walks: a Short Survey

Alan M. Frieze*
Department of Mathematical Sciences
Carnegie-Mellon University
Pittsburgh
PA 15213
USA

Abstract

There has been a significant amount of research lately on solving the edge disjoint path and related problems on expander graphs. We review the random walk approach of Broder, Frieze and Upfal.

1 Introduction

The basic problem discussed in this paper can be described as follows: we are given a graph $G = (V, E)$ and a set of K disjoint pairs of vertices in V. If possible, find edge disjoint paths P_i that join a_i to b_i for $i = 1, 2, \ldots, K$. We call this the Edge Disjoint Paths problem. We also say that G is *K-routable* if such paths exist for *any* set of K pairs. For arbitrary graphs, deciding whether such paths exist is in $\mathcal{P}$ for fixed K – Robertson and Seymour [16], but is $\mathcal{NP}$-complete if K is part of the input, being one of Karp's original problems. This negative result can be circumvented for certain classes of graphs, see Frank [7]. In this paper we will focus on *expander graphs*. There have been essentially two bases for approaches to this problem in this context: (i) random walks and (ii) multicommodity flows. Our aim here is to provide a summary of the results known to us at present together with an outline of some of their proofs. We emphasise the random walk approach, see [11, 12, 13] for more detail on the multicommodity flow approach.

Expander Graphs For certain bounded degree expander graphs, Peleg and Upfal [15] showed that if G is a sufficiently strong expander then G is n^ϵ-routable for some small constant $\epsilon \ll 1/3$ that depends only on the expansion properties of the input graph. Furthermore there is a polynomial time algorithm for constructing such paths.

This result has now been substantially improved and there is only a small factor (essentially $\log n$) between upper and lower bounds for maximum routability.

*Supported in part by NSF grant CCR9530974. E-mail: **alan@random.math.cmu.edu**.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 1-14, 1998.
Springer-Verlag Berlin Heidelberg 1998

Using random walks, Broder, Frieze and Upfal [2] improved the result of [15] to obtain the same result for $K = n/(\log n)^\kappa$ where κ depends only on the expansion properties of the graph. More recently, they [3] improved this by replacing κ by $2 + \epsilon$ for any positive constant $\epsilon > 0$, at the expense of requesting greater expansion properties of G. More recently still, Leighton, Rao and Srinivasan [13], using the rival multi-commodity flow technology have improved on this by showing that the ϵ can be replaced by $o(1)$. In Section 4 we will show how the random walks approach can be improved to give the same result, Theorem 3. It is rather interesting that these, in many ways quite different, approaches seem to yield roughly the same results. We note that both approaches yield a non-constructive proof [3], [12] (via the local lemma) that in a sufficiently strong expander $K = \Omega(n/(\log n)^2)$ is achievable.

Random Graphs Random graphs are well known to be excellent expanders and so it is perhaps not surprising that they very highly "routable". Broder, Frieze, Suen and Upfal [4] and Frieze and Zhao [9] (see Theorems 7,8) show that they are K-routable where K is within a constant factor of a simple lower bound, something that has not yet been achieved for arbitrary expander graphs.

Low Congestion Path Sets One way of generalising the problem is to bound the number of paths that use any one edge, the *edge congestion*, by some value g in place of one. Bounds on the number of routable pairs in this case are given in Theorem 5.

Dynamic problem In the dynamic version of the problem each vertex receives an infinite stream of requests for paths starting at that vertex. The times between requests are random and paths are are only required for a certain time (until the communication terminates) and then the path is deleted. Again each edge in the network should not be used by more than g paths at once.

The random walk approach gives a simple and fully distributed solution for this problem. In [3] (see Theorem 6) we show that if the injection to the network and the duration of connections are both controlled by Poisson processes then there is an algorithm which achieves a steady state utilization of the network which is similar to the utilization achieved in the *static* case situation, Theorem 5.

Approximation Algorithm So far we have only considered the case where all requests for paths have to be filled. If this is not possible then one might be satisfied with filling as many requests as possible. Kleinberg and Rubinfeld [10] (see Theorem 10) prove that a certain greedy strategy provides has a worst-case performance ratio of order $1/(\log n \log\log n)$.

Vertex Disjoint Paths Finally, there is the problem of finding vertex disjoint paths between a given set of pairs of vertices. In the worst-case one cannot do better than the minimum degree of the graph. The interest therefore must be on graphs with degrees which grow with the size of the graph. In this context random graphs [5] have optimal routing properties, to within a constant factor.

The structure of the paper is now as follows: Section 3 discusses the problem of splitting an expander, a basic requirement for finding edge disjoint paths. Section 4 details the aforementioned results on expander graphs and outlines some of the proofs for the random walk approach. Section 5 details the results on random graphs and outlines the corresponding proofs. Section 6 describes the result of Kleinberg and Rubinfeld. A final section provides some open problems.

2 Preliminaries

There are various ways to define expander graphs; here we define them in terms of edge expansion (a weaker property than vertex expansion).

For a set of vertices $S \subset V$ let $\mathrm{out}(S)$ be the number of edges with one end-point in S and one end-point in $V \setminus S$, that is

$$\mathrm{out}(S) = \left| \left\{ \{u,v\} \mid \{u,v\} \in E, u \in S, v \notin S \right\} \right|.$$

Similarly,

$$\mathrm{in}(S) = \left| \left\{ \{u,v\} \mid \{u,v\} \in E, u,v \in S \right\} \right|.$$

Definition 1 *A graph $G = (V,E)$ is a β-expander, if for every set $S \subset V$, $|S| \leq |V|/2$, we have $\mathrm{out}(S) \geq \beta|S|$.*

For certain results we need expanders that have the property that the expansion of small sets is not too small. The form of definition given below is taken from [3].

Definition 2 *An r-regular graph $G = (V,E)$ is called an (α,β,γ)-expander if for every set $S \subset V$*

$$\mathrm{out}(S) \geq \begin{cases} (1-\alpha)r|S| & \text{if } |S| \leq \gamma|V| \\ \beta|S| & \text{if } \gamma|V| < |S| \leq |V|/2 \end{cases}$$

In particular random regular graphs and the (explicitly constructible) Ramanujan graphs of of Lubotsky, Phillips and Sarnak [14] are (α,β,γ)-expanders with $\alpha = O(\gamma + \frac{1}{r^{1/2}})$ and β close to $r/4$.

A *random walk* on an undirected graph $G = (V,E)$ is a Markov chain $\{X_t\} \subseteq V$ associated with a particle that moves from vertex to vertex according to the following rule: the probability of a transition from vertex i, of degree d_i, to vertex j is $1/d_i$ if $\{i,j\} \in E$, and 0 otherwise. (In case of a bi-partite graph we need to assume that we do nothing with probability 1/2 and move off with probability 1/2 only. This technicality is ignored for the remainder of the paper.) Its stationary distribution, denoted π, is given by $\pi(v) = d_v/(2|E|)$. Obviously, for regular graphs, the stationary distribution is uniform.

A trajectory W of length τ is a sequence of vertices $[w_0, w_1, \ldots, w_\tau]$ such that $\{w_t, w_{t+1}\} \in E$. The Markov chain $\{X_t\}$ induces a probability distribution on trajectories, namely the product of the probabilities of the transitions that define the trajectory.

Let P denote the transition probability matrix of the random walk on G, and let $P^{(t)}_{v,w}$ denote the probability that the walk is at w at step t given that it started at v. Let λ be the second largest eigenvalue of P. (All eigenvalues of P are real.) It is known that

$$|P^{(t)}_{v,w} - \pi(w)| \leq \lambda^t \sqrt{\pi(w)/\pi(v)}. \tag{1}$$

In particular, for regular graphs

$$P^{(t)}_{v,w} = \frac{1}{n} + O(\lambda^t). \tag{2}$$

To ensure rapid convergence we need $\lambda \le 1-\epsilon$ for some *constant* $\epsilon > 0$. This holds for all expanders (Alon [1]). In particular if G is a β-expander with maximum degree Δ respectively then Jerrum and Sinclair [17] show that

$$\lambda \le 1 - \frac{1}{2}\left(\frac{\beta}{\Delta}\right)^2 \tag{3}$$

It is often useful to consider the *separation* s of the distribution $P^{(t)}_{v,\cdot}$ from the limit distribution π given by

$$s(t) = \max_{v,w} \frac{\pi(w) - P^{(t)}_{v,w}}{\pi(w)}. \tag{4}$$

Then we can write

$$P^{(t)}_{v,\cdot} = (1-s(t))\pi + s(t)\sigma$$

where σ is a probability distribution. We can then imagine that the distribution $P^{(t)}_{v,\cdot}$ is producing by choosing either σ with probability $s(t)$ or π with probability $1-s(t)$. Hence if $\mathcal{E}$ is an event that depends only on the state of the Markov chain we have

$$(1-s(t))\mathbf{Pr}(\mathcal{E} \text{ under } \pi) + s(t) \ge \mathbf{Pr}(\mathcal{E} \text{ under } P^{(t)}_{v,\cdot}) \ge (1-s(t))\mathbf{Pr}(\mathcal{E} \text{ under } \pi). \tag{5}$$

We use this in the following scenario:

Experiment A: Choose $u_1 \in V$ with distribution π and do a random walk W_1 of length τ from u_1. Let v_1 be the terminal vertex of W_1.

Experiment B: Choose u_2 and v_2 independently from V with distribution π and do a random walk of length τ from u_2 to v_2.

We claim that for any event $\mathcal{E}$ depending on walks of length τ,

$$|\mathbf{Pr}((u_1,v_1,W_1) \in \mathcal{E}) - \mathbf{Pr}((u_2,v_2,W_2) \in \mathcal{E})| \le s(\tau). \tag{6}$$

This follows from the stronger claim that for any $u \in V$ and any event $\mathcal{E}$ depending on walks of length τ

$$|\mathbf{Pr}((u_1,v_1,W_1) \in \mathcal{E} \mid u_1 = u) - \mathbf{Pr}((u_2,v_2,W_2) \in \mathcal{E} \mid u_2 = u)| \le s(\tau),$$

which follows from (5).

The notation $B(m,p)$ stands for the binomial random variable with parameters $m =$ number of trials, and $p =$ probability of success.

3 Splitting an Expander

Most of the algorithms we describe work in phases. Each phase generates paths and it is important that the sets of paths produced in each phase remain edge disjoint. One way

of ensuring this is to insist that different phases work on different expander graphs. If the input consists of a single expander then we need a procedure for partitioning E into p sets, say $E_1, E_2, \ldots, E_p$, where the graphs $G_i = (V, E_i)$ are themselves expanders.

A natural way of trying to split G into expander graphs is to randomly partition E into p sets. The problem with this is that in a bounded degree expander this will almost surely lead to subgraphs with isolated vertices. We must find a partition which provides a high minimum degree in both graphs. The solution in [2, 3] is

Algorithm Partition(G, p)

1. Orient the edges of G so that $|outdegree(v) - indegree(v)| \leq 1$ for all $v \in V$.
2. For each $v \in V$ randomly partition the edges directed out of v into p sets $X_{v,1}, \ldots, X_{v,p}$ each of size $\lfloor r/2p \rfloor$ or $\lceil r/2p \rceil$. Let $E_i = \bigcup_{v \in V} X_{v,i}$, for $1 \leq i \leq p$.

Define $H(\gamma) = ((1-\gamma)^{1-\gamma}\gamma^{\gamma})^{-1}$ and $\psi(\epsilon) = (1-\epsilon)\ln(1-\epsilon) + \epsilon$.

Theorem 1 *Let $G = (V, E)$ be an r-regular n-vertex graph that is an (α, β, γ)-expander. If $\epsilon \in (0, 1)$ and $p \leq r/2$ are such that $\beta > p(\gamma\psi(\epsilon))^{-1}\ln(H(\gamma))$, then Partition splits the edge-set of G into p subgraphs. With probability at least* [1] *$1 - \exp(-n(\beta\gamma\psi(\epsilon)/p - \ln(H(\gamma)) - o(1)))$, all the p subgraphs span V and have edge-expansion at least*

- *$\lfloor r/(2p) \rfloor - \alpha r$ for sets of size at most γn.*
- *$(1-\epsilon)\beta/p$ for sets of size betweem γn and $n/2$.*

In particular each G_i is a ζ-expander where

$$\zeta = \min\{\lfloor r/(2p) \rfloor - \alpha r, (1-\epsilon)\beta/p\}. \tag{7}$$

This does not seem to be the best way to proceed, but it is the best we know constructively. Frieze and Molloy [8] have a stronger result which is close to optimal, but at present it is non-constructive. Let the edge-expansion $\eta(G)$ of G be defined by

$$\eta(G) = \min_{\substack{S \subseteq V \\ |S| \leq n/2}} \frac{\mathrm{out}(S)}{|S|}.$$

Theorem 2 *Let $p \geq 2$ be a positive integer and let $\epsilon > 0$ be a small positive real number. Suppose that*

$$\begin{aligned} \frac{r}{\log r} &\geq 7p\epsilon^{-2} \\ \eta(G) &\geq 4\epsilon^{-2}p\log r. \end{aligned}$$

Then there exists a partition $E = E_1 \cup E_2 \cup \cdots \cup E_p$ such that for $1 \leq i \leq p$

[1] The $o(1)$ term tends to 0 as $n \to \infty$.

(a)

$$\eta(G_i) \geq \frac{(1-\epsilon)\eta(G)}{p}.$$

(b)

$$\frac{(1-\epsilon)r}{p} \leq \delta(G_i) \leq \Delta(G_i) \leq \frac{(1+\epsilon)r}{p}.$$

4 Finding paths in Expander graphs

4.1 Edge Disjoint paths

We will first concentrate on showing how using random walks we can achieve the same bound on the number of routable pairs as given in [13].

Fix integer $s \geq 1$ and let $\log^{(s)}$ denote the natural logarithm iterated s times e.g. $\log^{(2)} n = \log\log n$.

Let $m_1 = \log^{(s)} n$ and $m_{i+1} = \lceil \frac{1}{4}\left(\frac{10}{3e}\right)^{m_i/10} \rceil$ for $i \geq 1$. Then let $K_i = \lfloor 2crn/(m_i(\log n)^2) \rfloor$ for $i \geq 1$ and $\sigma \leq s+2$ be the largest i such that $K_i > 0$. Here $c = O(\zeta^2/(r^2 s))$ is a positive constant – ζ as in (7).

Theorem 3 *Suppose G is an r-regular n-vertex graph that is an (α, β, γ)-expander. Suppose that $\zeta > 1$ above when $p = 5\sigma$. Then G is $crn/((\log n)^2 \log^{(s)} n)$ -routable.*

Proof We first split G into $p = 5\sigma$ expander graphs $G_1, G_2, \ldots, G_p$ using algorithm Partition. Note that the minimum degree in each G_i is at least $r/(2p)$ and maximum degree is at most $r/2$.

Let $H_i = G_{5(i-1)+1} \cup \cdots \cup G_{5i}$ for $1 \leq i \leq \sigma$. The algorithm runs in phases. Phase i is left to deal with at most $\lfloor K_i \rfloor$ pairs left over from Phases 1 to $i-1$, assuming these phases have all succeeded. Thus Phase σ should finish the job. We run Phase i on graph H_i and this keeps the paths edge disjoint. Denote the set of source-sink pairs for Phase i by $S_{A,i} = \{a_{1,i}, \ldots, a_{K_i,i}\}$ and $S_{B,i} = \{b_{1,i} \ldots, b_{K_i,i}\}$. Phase i is divided into 4 subphases.

Subphase i.a: The aim here is to choose w_j, W_j, $1 \leq j \leq 2K_i$ such that (i) $w_j \in W_j$, (ii) $|W_j| = m_i + 1$, (iii) the sets $W_j, 1 \leq j \leq 2K_i$ are pairwise disjoint and (iv) W_j induces a connected subgraph of $\Gamma_i = G_{5(i-1)+2}$.

As in [11] we can partition an arbitrary spanning tree T of Γ_i. Since T has maximum degree at most r we can find $2K_i$ vertex disjoint subtrees T_j, $1 \leq j \leq 2K_i$ of T, each containing between m_i+1 and $(r-1)m_i+2$ vertices. We can find T_1 as follows: choose an arbitrary root ρ and let $Q_1, Q_2, \ldots, Q_\sigma$ be the subtrees of ρ. If there exists l such that Q_l has between m_i+1 and $(r-1)m_i+2$ vertices then we take $T_1 = Q_l$. Otherwise we can search for T_1 in any Q_ℓ with more than $(r-1)m_i+2$ vertices. Since $T \setminus T_1$ is connected, we can choose all of the T_j's in this way. Finally, W_j is the vertex set of an arbitrary m_i+1 vertex subtree of T_j and w_j is an arbitrary member of W_j for $j = 1, 2, \ldots, 2K_i$.

Subphase i.b: Using a network flow algorithm in $G_{5(i-1)+1}$ connect in an arbitrary manner the vertices of $S_{A,i} \cup S_{B,i}$ to $W_i = \{w_1, \ldots, w_{2K_i}\}$ by $2K_i$ edge disjoint paths as follows:

- Assume that every edge in $G_{5(i-1)+1}$ has a capacity equal to 1.
- View each vertex in S_i as a source with capacity 1 and similarly every vertex in W_i as a sink with capacity equal 1.

The expansion properties of $G_{4(i-1)+1}$ ensure that such flows always exist.

Let $\tilde{a}_{k,i}$ (resp. $\tilde{b}_{k,i}$) denote the vertex in W_i that was connected to the original endpoint $a_{k,i}$ (resp. $b_{k,i}$). Our problem is now to find edge disjoint paths joining $\tilde{a}_{k,i}$ to $\tilde{b}_{k,i}$ for $1 \le k \le K_i$.

Subphase i**.c**: If w_t has been renamed as $\tilde{a}_{k,i}$ (resp. $\tilde{b}_{k,i}$) then rename the elements of W_t as $\tilde{a}_{k,\ell,i}$, (resp. $\tilde{b}_{k,\ell,i},$) $1 \le \ell \le m_i$. Choose $\xi_j, 1 \le j \le m_i K_i$ and $\eta_j, 1 \le j \le m_i K_i$ independently at random from the steady state distribution π_i of a random walk on G_{5i}. Using a network flow algorithm as in Subphase i.b, connect $\{\tilde{a}_{k,\ell,i} : 1 \le k \le K_i, 1 \le \ell \le m_i\}$ to $\{\xi_j : 1 \le j \le m_i K_i\}$ by edge disjoint paths in G_{5i-2}. Similarly, connect $\{\tilde{b}_{k,\ell,i} : 1 \le k \le K_i, 1 \le \ell \le m_i\}$ to $\{\eta_j : 1 \le j \le m_i K_i\}$ by edge disjoint paths in G_{5i-1}. Rename the other endpoint of the path starting at $\tilde{a}_{k,\ell,i}$ (resp. $\tilde{b}_{k,\ell,i}$) as $\hat{a}_{k,\ell,i}$ (resp. $\hat{b}_{k,\ell,i}$).

Once again the expansion properties of G_{5i-2}, G_{5i-1} ensure that flows exist.

Subphase i**.d**: Choose $\hat{x}_{k,\ell,i}, 1 \le k \le K_i, 1 \le \ell \le m_i$ independently at random from the steady state distribution π_i of a random walk on G_{5i}. Let $W'_{k,\ell,i}$ (resp. $W''_{k,\ell,i}$ be a random walk of length $\theta \log n$ from $\hat{a}_{k,\ell,i}$ (resp. $\hat{b}_{k,\ell,i}$) to $\hat{x}_{k,\ell,i}$. Here $\theta = r^2/(2\zeta^2))$ is chosen so that the separation (4) between π_i and the distribution of the terminal vertex of the walk is $O(n^{-3})$. ((3) gives $\lambda_i \le 1 - 2\zeta^2/r^2$ where λ_i is the second largest eigenvalue of a random walk on G_{5i}.) The use of this intermediate vertex $\hat{x}_{k,\ell,i}$ helps to break some conditioning caused by the pairing up of the flow algorithm.

Let $B'_{k,i}$ (resp. $B''_{k,i}$) denote the *bundle* of walks $W'_{k,\ell,i}, 1 \le \ell \le m_i$ (resp. $W''_{k,\ell,i}, 1 \le \ell \le m_i$). Following [13] we say that $W'_{k,\ell,i}$ is *bad* if there exists $k' \ne k$ such that $W'_{k,\ell,i}$ shares an edge with a walk in a bundle $B'_{k',i}$ or $B''_{k',i}$. Each walk starts at an independently chosen vertex and moves to an independently chosen destination. The steady state of a random walk is uniform on edges and so at each stage of a walk, each edge is equally likely to be crossed. Thus

$$\mathbf{Pr}(W'_{k,\ell,i} \text{ is bad}) \le \frac{10 m_i K_i \theta^2 (\log n)^2 \sigma}{rn} \le 1/10$$

for sufficiently small c.

We say that index k is bad if either $B'_{k,i}$ or $B''_{k,i}$ contain more than $m_i/3$ bad walks. If index k is not bad then we can find a walk from $\hat{a}_{k,\ell,i}$ to $\hat{b}_{k,\ell,i}$ through $\hat{x}_{k,\ell,i}$ for some ℓ which is edge disjoint from all other walks. This gives a walk

$$a_{k,i} \to \tilde{a}_{k,i} \to \tilde{a}_{k,\ell,i} \to \hat{a}_{k,\ell,i} \to \hat{x}_{k,\ell,i} \to \hat{b}_{k,\ell,i} \to \tilde{b}_{k,\ell,i} \to \tilde{b}_{k,i} \to b_{k,i},$$

which is edge-disjoint from all other such walks.

The probability that index k is bad is at most

$$2\mathbf{Pr}(B(m_i, .1) \ge 1/3) \le 2(3e/10)^{m_i/10}.$$

So with probability at least 1/2 the number of bad indices is no more than $\lfloor 4K_i(3e/10)^{m_i/10} \rfloor \le K_{i+1}$. By repetition we can ensure that Phase i succeeds **whp**. The theorem follows. □

4.1.1 Existence Result

In this section we describe the use of the Lovász Local Lemma [6] to prove the *existence* of a large number of edge disjoint paths in any r-regular (α, β, γ)-expander, [3]. At the present time we do not see how to make the argument constructive.

Theorem 4 *Given a bounded degree (α, β, γ)-expander graph there exists a parameter c that depends on α, β, γ, but not on n, such that any set of less than $cn/(\log n)^2$ disjoint pairs of vertices can be connected by edge disjoint paths.*

The proof starts by splitting G into 2 $\beta' > 1$ expanders and using the first to route $a_1, \dots, b_K$ to randomly chosen $\tilde{a}_1, \dots, \tilde{b}_K$ via edge disjoint paths found through a flow algorithm as in say, Subphase i.b of the algorithm of the previous section.

Then, for $1 \le i \le K$, $\tilde{a}_i$ is joined to $\tilde{b}_i$ via an $O(\log n)$ random walk W_i through a randomly chosen intermediate vertex x_i. We use the local lemma to show that $W_1, \dots, W_K$ are edge disjoint with positive probability. Ignoring several technical problems we consider *bad* event $\mathcal{E}_{i,j} = \{W_i \cap W_j \neq \emptyset\}$ and argue that $\mathcal{E}_{i,j}$ depends only on the $< 2K$ events of the form $\mathcal{E}_{i',j}$ or $\mathcal{E}_{i,j'}$. Since $\mathbf{Pr}(\mathcal{E}_{i,j}) = O((\log n)^2/n)$ we can follow through if $K(\log n)^2/n \ll 1$. This gives the theorem.

4.2 Low Congestion Paths

We discuss the following result from [3].

Theorem 5 *There is an explicit polynomial time algorithm that can connect any set of $K = \alpha(n)n/\log n$ pairs of vertices on a bounded degree expander so that no edge is used by more than g paths where*

$$g = \begin{cases} O\left(s + \left\lceil \dfrac{\log\log n}{\log(1/\hat{\alpha})} \right\rceil\right), & \text{for } \alpha < 1/2; \\ O(s + \alpha + \log\log n), & \text{for } \alpha \ge 1/2, \end{cases}$$

$\hat{\alpha} = \min(\alpha, 1/\log\log n)$, and s is the maximal multiplicity of a vertex in the set of pairs.

See [11] for similar results proved via multi-commodity flows.

The algorithm uses the same flow/random walk paradigm that we have already seen twice above. $a_1, \dots, b_K$ are joined to randomly chosen $\tilde{a}_1, \dots, \tilde{b}_K$ via edge disjoint paths found through a flow algorithm. Then, for $1 \le i \le K$, $\tilde{a}_i$ is joined to $\tilde{b}_i$ via an $O(\log n)$ random walk W_i through a randomly chosen intermediate vertex x_i. The number of paths which use an edge is bounded by the sum of two binomials. We then see that for a sufficiently large $\kappa > 0$ **whp** there are fewer than $n/(\log n)^\kappa$ edges which have congestion greater than g. We delete all of the paths through such edges

and re-join the corresponding pairs via edge disjoint paths, using the algorithm of [2]. □

Leighton, Rao and Srinvasan [12] generalise Theorem 4 by showing that for any $B \geq 1$, given enough expansion one can join $cn/(\log n)^{1+1/B}$ pairs with congestion at most B.

4.3 Dynamic Allocation of Paths

Broder, frieze and Upfal [3] discuus a stochastic model for studying a dynamic version of the circuit switching problem. In their model new requests for establishing paths arrive continuously at nodes according to a discrete Poisson process. Requests wait in the processor's queue until the requested path is established. The duration of a path is exponentially distributed.

Their model is characterized by three parameters:

- P_1 is an upper bound on the probability that a new request arrives at a given node at a given step.
- P_2 is the probability that a given existing path is terminated in a given step. A path *lives* from the time it is established until it is terminated.
- g is the maximum congestion allowed on any edge.
- The destinations of path requests are chosen uniformly at random among all the graph vertices

They study a simple and fully distributed algorithm for this problem. In the algorithm each processor at each step becomes active with a probability $P_1' > P_1$. An inactive processor does not try to establish a path even if there are requests in its queue. The algorithm can be succinctly described: Assume that a is active at step t, and the first request in a's queue is for b. Processor a tries to establish a path to b by choosing a random trajectory of length $\tau = c_0 \log n$ connecting a to b. If the path does not use any edge with congestion greater than $g-1$, the path is established, otherwise the request stays in the queue.

Theorem 6 *Let*

$$\Phi = \min\left\{\frac{1}{\log(grn)}, \frac{rg}{\tau^{(g+1)/g}}\right\}.$$

There exists a constant γ such that if $P_1 \leq \gamma \Phi P_2$, then the system is stable and the expected wait of a request in the queue is $O(1/P_1)$.

Before outlining the proof let us see the consequence of this theorem. Let $\mathbf{E}(N) = nP_1$ be the expected number of new requests that arrive at the system at a given step, and let $\mathbf{E}(D) = 1/P_2$ be the expected duration of a connection. For the system to be stable, the expected number of simultaneously active paths in the steady state must be at least $\mathbf{E}(N)\mathbf{E}(D) = nP_1/P_2$. Plugging $g = \log\log n/\log\omega$ for some ω in the range $[1, \log n]$ in the definition of Φ we get

$$\Phi = \Omega\left(\frac{1}{\omega \log n}\right).$$

Thus the theorem above implies that for such a congestion g, the system remains stable even if we choose P_1 and P_2 such that

$$\mathbf{E}(N)\mathbf{E}(D) = n\frac{P_1}{P_2} = \gamma n\Phi = \Omega\left(\frac{n}{w\log n}\right),$$

in which case the dynamic algorithm utilizes the edges of the network almost as efficiently as the static algorithm, Theorem 5 (there seems to be an efficiency gap of maximum order $\log\log\log n$ for $\omega \le \log\log n$).

In the proof of the theorem, time is partitioned into intervals of length $T \le 1/(4P_1)$. Let H_t denote the history of the system during the first t time intervals. Define the event

$$\mathcal{E}(v,t) = \left\{\begin{matrix}\text{If the queue of processor } v \text{ was not empty at the beginning of interval}\\ t \text{ then } v \text{ served at least one request during interval } t\end{matrix}\right\}$$

The goal is to show that for all v and t,

$$\mathbf{Pr}(\mathcal{E}(v,t) \mid H_{t-2}) > \frac{1}{2}. \tag{8}$$

Given this we conclude that in any segment of $2T$ steps processor v is serving at least one request with probability at least $1/2$. The number of new arrivals in this time interval has a Binomial distribution with expectation at most $2TP_1 < 1/2$. Thus, under these conditions the queue is dominated by an $M/M/1$ queue with expected inter-arrival distribution greater than $4T$, and expected service time smaller than $4T$. The queue is stable, and the expected wait in the queue is $O(1/T) = O(1/P_1)$.

To prove (8) we argue that with sufficiently high probability, (i) v becomes active at least once during an interval, (ii) there are no very old paths in the network, (iii) there are not too many paths in the network altogether and then (iv) we can argue that the first path that a processor v tries to establish is unlikely to use a fully loaded edge.

5 Random Graphs

We deal with two related models of a random graph. $G_{n,m}$ has vertex set $[n] = \{1,2,\ldots,n\}$ and and exactly m edges, all sets of m edges having equal probability. The random graph $G_{r-\text{reg}}$ is uniformly randomly chosen from the set of r-regular graphs with vertex set $[n]$.

Let D be the median distance between pairs of vertices in graph $G_{n,m}$. Clearly it is not possible to connect more than $O(m/D)$ pairs of vertices by edge-disjoint paths, for all choices of pairs, since some choice would require more edges than all the edges available. In the case of bounded degree expanders, this absolute upper bound on k is $O(n/\log n)$. The results mentioned above use only a vanishing fraction of the set of edges of the graph, thus are far from reaching this upper bound. In contrast, Broder, Frieze, Suen and Upfal [4] and Frieze and Zhao [9] show that for $G_{n,m}$ and $G_{r-\text{reg}}$ the absolute upper bound is achievable within a constant factor, and present algorithms that construct the required paths in polynomial time.

Theorem 7 *Let $m = m(n)$ be such that $d = 2m/n \geq (1 + o(1)) \log n$. Then, as $n \longrightarrow \infty$, with probability $1 - o(1)$, the graph $G_{n,m}$ has the following property: there exist positive constants α and β such that for all sets of pairs of vertices $\{(a_i, b_i) \mid i = 1, ..., K\}$ satisfying:*

(i) $K \leq \lceil \alpha m \log d / \log n \rceil$,

(ii) for each vertex v, $|\{i : a_i = v\}| + |\{i : b_i = v\}| \leq \min\{d_G(v), \beta d\}$,

there exist edge-disjoint paths in G, joining a_i to b_i, for each $i = 1, 2, \ldots, K$. Furthermore, there is an $O(nm^2)$ time randomized algorithm for constructing these paths.

Theorem 8 *Let r be a sufficiently large constant. Then, as $n \longrightarrow \infty$, the graph G_{r-reg} has the following property* **whp***: there exist positive absolute constants α, β such that for all sets of pairs of vertices $\{(a_i, b_i) \mid i = 1, ..., K\}$ satisfying:*

(i) $K \leq \lceil \alpha r n / \log_r n \rceil$,

(ii) for each vertex v, $|\{i : a_i = v\}| + |\{i : b_i = v\}| \leq \beta r$,

there exist edge-disjoint paths in G_{r-reg}, joining a_i to b_i, for each $i = 1, 2, \ldots, K$. Furthermore, there is an $O(n^3)$ time randomized algorithm for constructing these paths.

These results are best possible up to constant factors. Consider for example Theorem 7. For (i) note that the distance between most pairs of vertices in $G_{n,m}$ is $\Omega(\log n / \log d)$, and thus with m edges we can connect at most $O(m \log d / \log n)$ pairs. For (ii) note that a vertex v can be the endpoint of at most $d_G(v)$ different paths. Furthermore suppose that $d \geq n^\gamma$ for some constant $\gamma > 0$ so that $K \geq \lceil \alpha \gamma n d / 2 \rceil$. Let $\epsilon = \alpha\gamma/3$, $A = [\epsilon n]$, and $B = [n] \setminus A$. Now with probability 1-o(1) there are less than $(1 + o(1))\epsilon(1 - \epsilon)nd$ edges between A and B in $G_{n,m}$. However almost all vertices of A have degree $(1 + o(1))d$ and if for these vertices we ask for $(1 - \epsilon/2)d$ edge-disjoint paths to vertices in B then the number of paths required is at most $(1 + o(1))\epsilon(1 - \epsilon/2)nd < K$, but, without further restrictions, this many paths would require at least $(1 - o(1))\epsilon(1 - \epsilon/2)nd > (1 + o(1))\epsilon(1 - \epsilon)nd$ edges between A and B which is more than what is available. This justifies an upper bound of $1 - \epsilon/2$ for β of Theorem 7. A similar argument justifies the bounds in Theorem 8.

First consider Theorem 7. The edge set E is first split randomly into 5 sets $E_1, E_2, \ldots, E_5$. The graphs $G_i = (V, E_i)$ will all be good expanders **whp**. As usual, G_1 is used to connect $a_1, \ldots, b_K$ to randomly chosen $\tilde{a}_1, \ldots, \tilde{b}_K$ using network flows. A random walk is then done in G_2 starting at each of these latter vertices and ending at $\hat{a}_1, \ldots, \hat{b}_K$. After each walk, the edges are deleted from G_2 which keeps the walks edge disjoint. The length τ of these walks is $O(\log n / \log d)$ but long enough so that the endpoints $\hat{a}_1, \ldots, \hat{b}_K$ are (essentially) independent of their start points. This handles any possible conditioning introduced by the pairing of $\tilde{a}_i$ with $\tilde{b}_i$. Finally, $\hat{a}_i$ is joined to $\hat{b}_i$ directly by a random walk of length τ in G_3. G_4 and G_5 and the algorithm of [2] are used to connect the few pairs not successfully joined by the above process.

The success of this algorithm rests on the fact that if not too many walks, $O(m \log d / \log n)$, are deleted then $O(m)$ edges will be deleted and we will be easily

be able to ensure that the degrees of almost all vertices stay logarithmic in size. Thus the remaining graphs will be good expanders. The reader should notice that we only need a constant number of short random walks to connect each pair and this is why we are within a constant of optimal.

In Theorem 3 where degrees are bounded, we find that this argument breaks down because many (order n) vertices would become isolated through the deletion of the requested number of walks. The cure for this is to force the "action" to take place on a *core* of each subgraph (The k-core of a graph H is the largest subset of V which induces a subgraph of minimum degree at least k in H. It is unique and can be found by repeatedly removing vertices which have degree less than k.) This raises technical problems, such as what is to be done when one of the endpoints of a proposed walk drops out of the core. These problems are dealt with in [9].

The problem of finding vertex disjoint paths in random graphs is dealt with in Broder, Frieze, Suen and Upfal [5].

Theorem 9 *Suppose $m = \frac{n}{2}(\log n + \omega)$ where $\omega(n) \to \infty$. Then there exists $\alpha, \beta > 0$ such that* **whp** *for all $A = \{a_1, a_2, \ldots, a_K\}, B = \{b_1, b_2, \ldots, b_K\} \subseteq [n]$ satisfying*

(i) $A \cap B = \emptyset$

(ii) $|A| = |B| \leq \frac{\alpha n \ln d}{\log n}$

(iii) $|N(v) \cap (A \cup B)| \leq \beta |N(v)|$.

there are vertex disjoint paths P_i from a_i to b_i for $1 \leq i \leq K$. Furthermore these paths can be constructed by a randomised algorithm in $O(n^3)$ time.

This result is best possible up to constant factors.

6 Approximation Algorithm

Kleinberg and Rubinfeld [10] describe an on-line Bounded Degree (BGA) Approximation algorithm for the edge disjoint paths problem. BGA is defined by a parameter L as follows:

(i) Proceed through the terminal pairs in one pass.

(ii) When (a_i, b_i) is considered, check whether a_i and b_i can still be joined by a path of length at most L. If so, route (a_i, b_i) on such a path P_i. Delete P_i and iterate.

They prove the following:

Theorem 10 *Suppose G is an expander of maximum degree Δ. Then there exists $c > 0$ such that with $L = c\Delta \log n$, BGA is an $O(\log n \log\log n)$-approximation algorithm for the edge disjoint paths problem.*

7 Final Remarks

There has been a lot of progress since the first paper of Peleg and Upfal. The most interesting questions that remain to my mind are:

1. Can we take $K = \Omega(n/\log n)$, given sufficient expansion, in Theorem 3?
2. More modestly, can we remove the $\log^{(s)} n$ factor and make Theorem 4 constructive?
3. Can we achieve near optimal expander splitting as in Theorem 2, constructively?
4. Is there a constant factor approximation algorithm for the edge disjoint paths problem on expander graphs?
5. Can any of the above results be extended to digraphs?

References

[1] N.Alon, Eigenvalues and expanders, *Combinatorica* 6 (1986) 83-96.

[2] A.Z.Broder, A.M.Frieze and E.Upfal, Existence and construction of edge-disjoint paths on expander graphs, *SIAM Journal of Computing* 23 (1994) 976-989.

[3] A.Z.Broder, A.M.Frieze and E.Upfal, Existence and construction of edge low congestion paths on expander graphs, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, (1997) 531-539.

[4] A.Z.Broder, A.M.Frieze, S.Suen and E.Upfal, Optimal construction of edge disjoint paths in random graphs, *Proceedings of 4th Annual Symposium on Discrete Algorithms*, (1994) 603-612.

[5] A.Z.Broder, A.M.Frieze, S.Suen and E.Upfal, An efficient algorithm for the vertex-disjoint paths problem in random graphs, *Proceedings of 6th Annual Symposium on Discrete Algorithms*, (1996) 261-268.

[6] P.Erdös and L.Lovász, Problems and results on 3-chromatic hypergraphs and some related questions, In A.Hajnal et al., editors, *Infinite and Finite Sets*, Volume 11 of *Colloq. Math. Soc. J. Bolyai* (1975)609-627.

[7] A.Frank, Packing paths, cuts and circuits – a survey, in *Paths, Flows and VLSI Layout*, B.Korte, L.Lovász, H.J.Prömel and A.Schrijver Eds., Springer-Verlag, 1990.

[8] A.M.Frieze and M.Molloy, Splitting expanders.

[9] A.M.Frieze and L.Zhao, Edge disjoint paths in random regular graphs.

[10] J.Kleinberg and R.Rubinfeld, Short paths in expander graphs, *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, (1996) 86-95.

[11] T.Leighton and S.Rao, Circuit switching: a multicommodity flow based approach, Proceedings of a Workshop on Randomized Parallel Computing 1996.

[12] T.Leighton, S.Rao and A.Srinivasan, Multi-commodity flow and circuit switching, *Proceedings of the Hawaii International Conference on System Sciences*, 1998.

[13] T.Leighton, S.Rao and A.Srinivasan, New algorithmic aspects of the local lemma with applications to partitioning and routing.

[14] A.Lubotsky, R.Phillips, and P.Sarnak, Ramanujan graphs, *Combinatorica* 8 (1988) 261-277.

[15] D.Peleg and E.Upfal, Constructing disjoint paths on expander graphs, *Combinatorica*, 9, (1989) 289-313.

[16] N.Robertson and P.D. Seymour, Graph minors-XIII: The disjoint paths problem.

[17] A.Sinclair and M.Jerrum, Approximate counting, uniform generation and rapidly mixing Markov chains, *Information and Computation* 82(1) (1989) 93-133.

A Derandomization Using Min-Wise Independent Permutations

Andrei Z. Broder[1], Moses Charikar[2⋆], and Michael Mitzenmacher[1]

[1] Compaq SRC, 130 Lytton Avenue, Palo Alto, CA 94301, USA.
{broder,michaelm}@pa.dec.com
[2] Computer Science Dept., Stanford University, CA 94305, USA.
moses@cs.stanford.edu

Abstract. Min-wise independence is a recently introduced notion of limited independence, similar in spirit to pairwise independence. The later has proven essential for the derandomization of many algorithms. Here we show that approximate min-wise independence allows similar uses, by presenting a derandomization of the RNC algorithm for approximate set cover due to S. Rajagopalan and V. Vazirani. We also discuss how to derandomize their set multi-cover and multi-set multi-cover algorithms in restricted cases. The multi-cover case leads us to discuss the concept of *k-minima-wise* independence, a natural counterpart to k-wise independence.

1 Introduction

Carter and Wegman [6] introduced the concept of universal hashing in 1979, with the intent to offer an input independent, constant average time algorithm for table look-up. Although hashing was invented in the mid-fifties, when for the first time memory become "cheap" and therefore sparse tables became of interest, up until the seminal paper of Carter and Wegman the premise of the theory and practice of hashing was that either the input is chosen at random or the hash function is chosen uniformly at random among all possible hash functions. Both premises are clearly unrealistic: inputs are not random, and the space needed to store a truly random hash function would dwarf the size of the table. Carter and Wegman showed that, in order to preserve the desirable properties of hashing, it suffices to pick the hash function from what is now called a pair-wise independent family of hash functions. Such families of small size exist, and can be easily constructed.

Since then, pairwise independence and more generally k-wise independence have proven to be powerful algorithmic tools with significant theoretical and practical applications. (See the excellent survey by Luby and Wigderson [11]

⋆ Supported by the Pierre and Christine Lamond Fellowship and in part by an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 15-24, 1998.

and references therein.) One important theoretical application of pairwise independence is for the derandomization of algorithms. A well-known example is to find a large cut in a graph. One can color the vertices of a graph with $|E|$ edges randomly using two colors, the colors being determined by a pairwise independent hash function chosen at random from a small family. The colors define a cut, and on average the cut will have $|E|/2$ crossing edges. Hence, by trying every hash function in the family one finds a cut with at least the expected number of crossing edges, $|E|/2$.

Recently, we introduced an alternative notion of limited independence based on what we call min-wise independent permutations [4]. Our motivation was the connection to an approach for determining the resemblance of sets, which can be used for example to identify documents on the World Wide Web that are essentially the same [2, 3, 5]. In this paper we demonstrate that the notion of min-wise independence can also prove useful for derandomization. Specifically, we use a polynomial-sized construction of approximate min-wise independent permutations due to Indyk to derandomize the parallel approximate set cover algorithm of Rajagopalan and Vazirani [12]. (From now on, called the *RV-algorithm.*) This example furthers our hope that min-wise independence may prove a generally useful concept.

The paper proceeds as follows: in Section 2, we provide the definitions for min-wise and approximately min-wise independent families of permutations. We also state (without proof) Indyk's results. In Section 3, we provide the necessary background for the RV-algorithm. In particular, we emphasize how the property of min-wise independence plays an important role in the algorithm. In Section 4, we demonstrate that the RV-algorithm can be derandomized using a polynomial sized approximately min-wise independent family. Finally, in Section 5, we briefly discuss how to extend the derandomization technique to the set multi-cover and multi-set multi-cover algorithms proposed by Rajagopalan and Vazirani. This discussion motivates a generalization of min-wise independence to *k-minima-wise* independence, a natural counterpart to k-wise independence.

2 Min-wise independence

We provide the necessary definitions for min-wise independence, based on [4].

Let S_n be the set of all permutations of $[n]$. We say that $\mathcal{F} \subseteq S_n$ is *exactly min-wise independent* (or just *min-wise independent* where the meaning is clear) if for any set $X \subseteq [n]$ and any $x \in X$, when π is chosen at random [1] from $\mathcal{F}$ we have

$$\mathbf{Pr}\big(\min\{\pi(X)\} = \pi(x)\big) = \frac{1}{|X|}. \tag{1}$$

In other words we require that all the elements of any fixed set X have an equal chance to become the minimum element of the image of X under π.

[1] To simplify exposition we shall assume that π is chosen uniformly at random from $\mathcal{F}$, although it could be advatageous to use a another distribution instead. See [4].

We say that $\mathcal{F} \subseteq S_n$ is *approximately min-wise independent with relative error* ϵ (or just approximately min-wise independent where the meaning is clear) if for any set $X \subseteq [n]$ and any $x \in X$, when π is chosen at random from $\mathcal{F}$ we have

$$\left| \mathbf{Pr}\big(\min\{\pi(X)\} = \pi(x)\big) - \frac{1}{|X|} \right| \leq \frac{\epsilon}{|X|}. \tag{2}$$

In other words we require that all the elements of any fixed set X have only an almost equal chance to become the minimum element of the image of X under π.

Indyk has found a simple construction of approximately min-wise independent permutations with useful properties for derandomization [9]. The construction is derived from a family of hash functions that map $[n]$ to a larger set $[m]$ and have certain limited independence properties. A function $h : [n] \to [m]$ induces a permutation π of $[n]$ as follows: sort the n pairs $(h(x), x)$ for $x \in [n]$ in lexicographic order and define $\pi(x)$ to be the index of $(h(x), x)$ in the sorted order. Thus, a family of hash functions $\{h : [n] \to [m]\}$ induces a family of permutations of $[n]$. Indyk's results imply the following proposition.

Proposition 1. [Indyk] *There exists constants c_1 and c_2 such that, for any $c_1 \log(1/\epsilon)$-wise independent family $\mathcal{H}$ of hash functions from $[n]$ to $[c_2 n/\epsilon]$, the family of permutations on $[n]$ induced by $\mathcal{H}$ is approximately min-wise independent with relative error ϵ.*

Using the above proposition, an approximately min-wise independent family can be constructed as follows. Let $r = \lceil \log(c_2 n/\epsilon) \rceil$. We need a hash function that associates to each element in $[n]$ an r-bit string. We construct a string of length $n \cdot r$ bits, representing the concatenation of all the hash values, such that the bits are $rc_1 \log(1/\epsilon)$ independent. Thus the hash values for any $c_1 \log(1/\epsilon)$ elements are independent. Proposition 1 ensures that the family of permutations induced by this construction is approximately min-wise independent. Moreover, we can use the constructions of almost k-wise independent random variables due to Alon et. al. [1]. The fact that the bits will be only approximately k-wise independent can be absorbed into the relative error for the approximately min-wise independent family of permutations. As noted in [1], the construction of the appropriate approximately independent bit strings can be performed in NC, implying that the construction of an approximately min-wise independent family of permutations can be performed in NC. The size of the family of permutations obtained is $n^{O(\log(1/\epsilon))}$.

Hence in what follows we will use the fact that there exist NC-constructible approximately min-wise independent families of permutations of size $n^{O(\log(1/\epsilon))}$.

3 The parallel set cover algorithm

3.1 The problem

The set cover problem is as follows: given a collection of sets over a universe of n elements, and given an associated cost for each set, find the minimum cost

Preprocess.

Iteration:

For each not-covered element e, compute $value(e)$.

For each set S, include S in $\mathcal{L}$ if $\sum_{e \in U(S)} value(e) \geq C_S/2$.

Phase:

(a) Permute $\mathcal{L}$ at random.

(b) Each not-covered element e votes for the first set S in the random order such that $e \in S$.

(c) If $\sum_{e \text{ votes } S} value(e) \geq C_S/16$, then add S to the set cover.

(d) Remove from $\mathcal{L}$ any set not satisfying $\sum_{e \in U(S)} value(e) \geq C_S/2$.

Repeat until $\mathcal{L}$ is empty.

Iterate until all elements are covered.

Fig. 1. The RV-algorithm for parallel set cover

sub-collection of sets that covers all of the n elements. This problem (with unit costs) is included in Karp's famous 1972 list [10] of NP-complete problems. (See also [8].)

The natural greedy algorithm repeatedly adds to the cover the set that minimizes the average cost per newly added element. In other words, if the cost of set S is C_S, then at each step we add the set that minimizes $C_S/|U(S)|$, where $U(S)$ is the subset of S consisting of elements not yet covered. The greedy algorithm yields an H_n factor approximation. (H_n denotes the harmonic number $\sum_{1 \leq i \leq n} 1/i$.) For more on the history of this problem, see [12] and references therein. In particular Feige [7] has shown that improving this approximation is unlikely to be computationally feasible.

3.2 A parallel algorithm

The RV-algorithm is a natural modification of the greedy algorithm: instead of repeatedly choosing the set that covers elements at the minimum average current-cost, repeatedly choose some sets randomly from all sets with a suitably low minimum average current-cost. The intuition is that choosing several sets at a time ensures fast progress towards a solution; randomness is used in an ingenious way to ensure a certain amount of coordination so that not too many superfluous sets (that is, sets that cover few, if any, new elements) are used.

Define the *value* of an element to be:

$$value(e) = \min_{S \ni e} \frac{C_S}{|U(S)|}.$$

That is, the value of an element is the minimum possible cost to add it to the current cover. The algorithm of Rajagopalan and Vazirani is depicted in Figure 3.1.

The preprocessing step is used to guarantee that the costs C_S lie in a limited range; this is not of concern here since it does not involve any randomization. The randomization comes into play when the sets of $\mathcal{L}$ are randomly permuted, and each element votes for the first set in the random order. This property is exploited in the analysis of the algorithm in two ways:

1. The set that each element votes for is equally likely to be any set that contains it.
2. Given any pair of elements e and f, let N_e be the number of sets containing e but not f, let N_f be the number of sets containing f but not e, and let N_b be the number of sets that contain both. The probability that both e and f vote for the same set is
$$\frac{N_b}{N_e + N_b + N_f.}$$

Interestingly, both of these properties would hold if $\mathcal{L}$ were permuted according to a min-wise independent family of permutations; in fact, this is all that is required in the original analysis. Hence if we had a polynomial sized min-wise independent family, we could derandomize the algorithm immediately. Unfortunately, the lower bounds proven in [4] show that no such family exists; any min-wise independent family would have size exponential in $|\mathcal{L}|$.

We therefore consider what happens when we replace step (a) of the parallel set cover algorithm with the following step:

> (a') Permute $\mathcal{L}$ using a random permutation from an approximately min-wise independent family with error ϵ.

As we shall explain, for suitably small ϵ this replacement does not affect the correctness of the algorithm, and the running time increases at most by a constant factor. Using this fact, we will be able to derandomize the algorithm using Indyk's polynomial-sized construction.

4 The derandomization

We note that the proof of the approximation factor of the algorithm, as well as the bound on the number of iterations, does not change when we change how the permutation on $\mathcal{L}$ is chosen. Hence we refer the interested reader to the proofs in [12], and consider only the crux of the argument for the derandomization, namely the number of phases necessary for each iteration.

As in [12], we establish an appropriate potential function Φ, and show that its expected decrease $\Delta\Phi$ in each phase is $c\Phi$ for some constant c. The potential function is such that if it ever becomes 0 we are done. In [12], this was used

to show that $O(\log n)$ phases per round are sufficient, with high probability. By using a polynomial sized family of approximately min-wise independent permutations, we can try all possible permutations (on a sufficiently large number of processors) in each phase; in this way we ensure that in each phase the potential Φ decreases by a constant factor. This derandomizes the algorithm.

We review the argument with the necessary changes. The potential function Φ is $\sum_S U(S)$. The degree of an element e, denoted $\deg(e)$ is the number of sets containing it. A set-element pair (S, e) with $e \in U(S)$ is called *good* if $\deg(e) \geq \deg(f)$ for at least $3/4$ of the elements $f \in U(S)$. We show that on average a constant fraction of the good (S, e) pairs disappear in each phase (because sets are added to the cover), from which we can easily show that $\mathbf{E}(\Delta\Phi) \geq c\Phi$.

Lemma 2. *Let $e, f \in U(S)$ with $\deg(e) \geq \deg(f)$. Then*

$$\mathbf{Pr}(f \textit{ votes for } S \mid e \textit{ votes for } S) > \frac{1-\epsilon}{2(1+\epsilon)}.$$

Proof. Let N_e be the number of sets containing e but not f, let N_f be the number of sets containing f but not e, and let N_b be the number of sets that contain both. The set S is chosen by both e and f if it the smallest choice for both of them; this happens with probability at least $\frac{1-\epsilon}{N_e+N_b+N_f}$, by the definition of approximate min-wise independence. Similarly, the set S is chosen by e with probability at most $\frac{1+\epsilon}{N_e+N_b}$. Hence

$$\mathbf{Pr}(f \text{ votes for } S \mid e \text{ votes for } S) \geq \frac{1-\epsilon}{1+\epsilon} \cdot \frac{N_e + N_b}{N_e + N_b + N_f} \geq \frac{1-\epsilon}{2(1+\epsilon)}.$$

The last inequality follows from the fact that $N_e \geq N_f$.

The above lemma suggests that if (S, e) is good, and e votes for S, then S should get many votes. Indeed, this is the case.

Lemma 3. *If (S, e) is good then*

$$\mathbf{Pr}(S \textit{ is picked } \mid e \textit{ votes for } S) > \frac{1-4\epsilon}{15}.$$

Proof. Clearly $\mathit{value}(f) \leq C_S/|U(S)|$ for any $f \in U(S)$, so

$$\sum_{\substack{f \in U(S) \\ \deg(f) > \deg(e)}} \mathit{value}(f) \leq \frac{C_S}{4} \ .$$

But if $S \in \mathcal{L}$, then $\sum_{f \in U(S)} \mathit{value}(f) \geq C_S/2$. Therefore

$$\sum_{\substack{f \in U(S) \\ \deg(f) \leq \deg(e)}} \mathit{value}(f) \geq \frac{C_S}{4} \ .$$

By Lemma 2, if e votes for S, then each f with $\deg(f) \leq \deg(e)$ votes for S with probability at least $(1-\epsilon)/(2(1+\epsilon))$. Hence, conditioned on e voting for S, the expected total value of all elements that vote for S is at least $C_S(1-\epsilon)/\big(8(1+\epsilon)\big)$. Let p be the probability that S is picked in this case. Then as the total value from all elements that vote for S is at most C_S, clearly

$$pC_S + (1-p)\frac{C_S}{16} \geq \frac{C_S(1-\epsilon)}{8(1+\epsilon)}.$$

From this we obtain that $p > (1-4\epsilon)/15$.

From the Lemma above we show that the expected decrease in the potential function is a constant fraction per round.

Lemma 4. $\mathbf{E}(\Delta\Phi) \geq \frac{(1-5\epsilon)}{60}\Phi$.

Proof. As in [12], we estimate the decrease in Φ due to each pair (S, e) when e votes for S and S joins the cover. The associated decrease is $\deg(e)$ since Φ decreases by one for every remaining set that contains e. Hence

$$\begin{aligned}
\mathbf{E}(\Delta\Phi) &\geq \sum_{(S,e):e\in U(S)} \mathbf{Pr}(e \text{ voted } S \text{ and } S \text{ was picked }) \cdot \deg(e) \\
&\geq \sum_{(S,e) \text{ good}} \mathbf{Pr}(e \text{ voted } S) \cdot \mathbf{Pr}(S \text{ was picked} \mid e \text{ voted } S) \cdot \deg(e) \\
&\geq \sum_{(S,e) \text{ good}} \frac{1-\epsilon}{\deg(e)} \cdot \frac{1-4\epsilon}{15} \cdot \deg(e) \geq \sum_{(S,e) \text{ good}} \frac{1-5\epsilon}{15} \\
&\geq \sum_{(S,e):e\in U(S)} \frac{1-5\epsilon}{60} \geq \frac{1-5\epsilon}{60}\Phi.
\end{aligned}$$

If initially we have n sets and m elements, then initially $\Phi \leq mn$, and hence we may conclude that at most $O(\log nm)$ phases are required before an iteration completes. Given the results of [12], we may conclude:

Theorem 5. *The algorithm* Parallel Set Cover *can be derandomized to an NC^3 algorithm that approximates set cover within a factor of $16H_n$ using a polynomial number of processors.*

One may trade of the number of processors and a constant factor in the running time by varying the error ϵ. However, the family must be sufficiently large so that ϵ is small enough for the analysis to go through. Having $\epsilon < 1/5$ is sufficient (this can be improved easily, at least to $\epsilon < 1/3$).

5 Extensions

Besides the parallel set cover algorithm, Rajagopalan and Vazirani also provide algorithms for the more general set multi-cover and multi-set multi-cover problems. In the set multi-cover problem, each element has a requirement r_e, and it

must be covered r_e times. In the multi-set multi-cover problem, multi-sets are allowed. These algorithms follow the same basic paradigm as the parallel set cover algorithm, except that during the algorithm an element that still needs to be covered $r(e)$ more times gets $r(e)$ votes. (Note $r(e)$ is dynamic; $r(e) = r_e$ initially.)

Our derandomization approach using approximately min-wise independent families of permutations generalizes to these extensions as well, subject to a technical limitation that the initial requirements r_e must be bounded by a fixed constant. We need slightly more than approximate min-wise independence, however. The following properties are sufficient[2]:

- the ordered $r(e)$-tuple of the first $r(e)$ sets containing an element e in the random order is equally likely to be any ordered $r(e)$-tuple of sets that contain e,
- for any pair of elements e and f both in some set S, the ordered $(r(e) + r(f) - 1)$-tuple of the first $r(e) + r(f) - 1$ sets containing either e or f in the random order is equally likely to be any ordered $(r(e) + r(f) - 1)$-tuple of sets that contain either e or f.

Note that when $r(e) = r(f) = 1$, these conditions are implied by min-wise independence, as we would expect.

These requirements suggest a natural interpretation of min-wise independence: suppose that not just any element of a set X was equally likely to be the first after applying a permutation, but that any ordered set of k elements of a set X are equally likely to be the first k elements (in the correct order) after applying a permutation to X. Let us call this k-minima-wise independence. Then the properties above correspond to $\max_{e,f}(r(e) + r(f) - 1)$-minima-wise independence; if $\max_e r(e)$ is a fixed constant, then we require a k-minima-wise independent family of permutations for some constant k. In fact, as with the parallel set cover problem, we require only approximate k-minima-wise independence, and the construction of Indyk can be generalized to give us an appropriate family of polynomial size when k is a constant.

We note in passing that for estimating the resemblance of documents as in [2] and [5] with a "sketch" of size k we need one sample from a k-minima-wise independent family, while for the method presented in [3], we need k separate samples from a min-wise independent family.

There is an interesting meta-principle behind our derandomizations, which appears worth emphasizing here.

Remark 6. Let $\mathcal{E}$ be an event that depends only on the order of the first k elements of a random permutation. Then any bound on the probability of $\mathcal{E}$ that holds for random permutations also holds for any k-minima-wise independent family. Moreover, for any approximately k-minima-wise independent family, a suitable small correction to the bound holds.

[2] In fact they are more than is necessary; however, stating the properties in this form is convenient.

For example, many of the lemmata in [12] prove bounds for events assuming that the random permutations are generated by assigning each set a uniform random variable from $[0, 1]$ and then sorting. Because the events these lemmata bound depend only on the first $(r(e)+r(f)-1)$ sets of the permutation, the lemmata still hold when using $(r(e) + r(f) - 1)$-minima-wise independent families, and only minor corrective terms need to be introduced for $(r(e) + r(f) - 1)$-minima-wise independent families. Hence given the results of [12], the derandomizations follow with relatively little work.

6 Conclusion

We have demonstrated a novel derandomization using the explicit construction of approximate min-wise independent families of permutations of polynomial size. We expect that this technique may prove useful for further derandomizations.

The question of how to best construct small approximately min-wise independent families of permutations remains open. Improvements in these constructions would lead to improvements in the number of processors required for our derandomizations here, and more generally may enhance the utility of this technique.

References

1. N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.
2. A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences 1997*, pages 21–29. IEEE Computer Society, 1988.
3. A. Z. Broder. Filtering near-duplicate documents. In *Proceedings of FUN 98*, 1998. To appear.
4. A. Z. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 327–336, 1998.
5. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of the Sixth International World Wide Web Conference*, pages 391–404, 1997.
6. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, Apr. 1979.
7. U. Feige. A threshold of $\ln n$ for approximating set cover (preliminary version). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 314–318, Philadelphia, Pennsylvania, 22–24 May 1996.
8. M. R. Garey and D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.
9. P. Indyk, A small approximately min-wise independent family of hash functions. manuscript, 1998.
10. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

11. M. Luby and A. Wigderson. Pairwise independence and derandomization. Technical Report TR-95-035, International Computer Science Institute, Berkeley, California, 1995.
12. S. Rajagopalan and V. V. Vazirani. Primal-dual RNC approximation algorithms for (multi)-set (multi)-cover and covering integer programs. In *34th Annual Symposium on Foundations of Computer Science*, pages 322–331, Palo Alto, California, 3–5 Nov. 1993. IEEE. Journal version to appear in *SIAM Journal of Computing*.

An Algorithmic Embedding of Graphs via Perfect Matchings

Vojtech Rödl, Andrzej Ruciński, and Michelle Wagner

Department of Mathematics and Computer Science
Emory University, Atlanta, Georgia, USA
{rodl, andrzej, wagner}@mathcs.emory.edu

Abstract. Recently Komlós, Sárközy, and Szemerédi proved a striking result called the blow-up lemma that, loosely speaking, enables one to embed any bounded degree graph H as a spanning subgraph of an ϵ-regular graph G. The first proof given by Komlós, Sárközy, and Szemerédi was based on a probabilistic argument [8]. Subsequently, they derandomized their approach to provide an algorithmic embedding in [9]. In this paper we give a different proof of the algorithmic version of the blow-up lemma. Our approach is based on a derandomization of a probabilistic proof of the blow-up lemma given in [13]. The derandomization utilizes the Erdős-Selfridge method of conditional probabilities and the technique of pessimistic estimators.

1 Introduction

Given a graph G and two disjoint subsets U and W of its vertex set $V(G)$, denote by $e_G(U,W)$ the number of edges of G with one endpoint in U and the other in W. Define the *density* $d_G(U,W)$ of the pair (U,W) in G by $d_G(U,W) = \frac{e_G(U,W)}{|U||W|}$. Let $\epsilon > 0$, and let $G = (V_1, V_2; E)$ be a bipartite graph. G is called ϵ-*regular* if for every pair of sets (U,W), $U \subset V_1$, $W \subset V_2$, $|U| \geq \epsilon|V_1|, |W| \geq \epsilon|V_2|$,

$$|d_G(U,W) - d_G(V_1,V_2)| < \epsilon. \tag{1}$$

Let $\epsilon > 0$ and $0 < d < 1$. A bipartite graph $G = (V_1, V_2; E)$ is called *super* (d,ϵ)-*regular* if the following conditions hold:

(i) G is ϵ-regular,
(ii) for each $v \in V_i$, $(d-\epsilon)|V_{3-i}| \leq deg(v) \leq (d+\epsilon)|V_{3-i}|$, $i = 1, 2$.

Super (d,ϵ)-regular graphs with $|V_1| = |V_2| = n$ and $d > 2\epsilon$ satisfy the Hall condition and thus contain a perfect matching. In fact, as it has been shown in [2], the number of perfect matchings in such graphs is close to $d^n n!$.

Given two graphs H and G on the same number of vertices, we call a bijection $f : V(H) \to V(G)$ an *embedding of* H *into* G if f maps every edge of H onto an edge of G. In other words, f is an isomorphism between H and a subgraph of G. In [8] the following result was proved.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 25-34, 1998.

Theorem 1 (Blow Up Lemma). *For every choice of positive integers r and Δ, and $0 < d < 1$, there exists an $\delta > 0$ and an integer $n = n(\delta)$ such that the following holds. Let G be an r-partite graph with all partition sets $V_1, ..., V_r$ of order n and all $\binom{r}{2}$ bipartite subgraphs $G[V_i, V_j]$ super (d, δ)-regular. Then, for every r-partite graph H with maximum degree $\Delta(H) \leq \Delta$ and all partition sets $X_1, ..., X_r$ of order n, there exists an embedding f of H into G that maps X_i onto V_i, $i = 1, 2, \ldots, r$.*

Hence, it is possible to embed any graph H with bounded maximum degree as a *spanning* subgraph of a dense, ϵ-regular graph G. The Blow-up Lemma together with the Regularity Lemma of Szemerédi [15] enable one to tackle and solve difficult problems like the Pósa-Seymour conjecture on powers of hamiltonian cycles [10] or the Alon-Yuster conjecture on perfect F-matchings [11]. Since the Regularity Lemma has been already made algorithmic in [1], it is very important to provide a constructive version of the Blow Up Lemma too.

The idea of the original proof of Theorem 1 (cf. [8]) involves sequentially embedding the vertices of H into G while the sets of candidates for images of unembedded vertices are not threatened. When all but a small fraction of the vertices are already embedded, the remaining vertices are all embedded at once using the König-Hall theorem. In [13] an alternative proof, using random perfect matchings of super regular graphs, where H is embedded into G in only a constant number of rounds, was proposed. In this paper we present an algorithmic version of that proof. (An algorithmic version of the original proof of Theorem 1 can be found in [9].)

Theorem 2. *There is an algorithm EMBED which, in time polynomial in n, does the following. Given an r-partite graph G with all partition sets $V_1, ..., V_r$ of order n and all $\binom{r}{2}$ bipartite subgraphs $G[V_i, V_j]$ super (d, δ)-regular, and given an r-partite graph H with maximum degree $\Delta(H) \leq \Delta$ and all partition sets $X_1, ..., X_r$ of order n, where r and Δ are arbitrary positive integers, $0 < d < 1$, and $\delta = \delta(r, \Delta, d) > 0$ is sufficiently small, EMBED constructs an embedding f of H into G that maps X_i onto V_i, $i = 1, ..., r$.*

The algorithm EMBED consists of two phases. In the preliminary Phase 1, outlined in Sect. 2, refinements of the partitions of $V(H)$ and $V(G)$ are obtained and new edges are added to both H and G. Then in the main Phase 2, described in Sect. 3, H is embedded into G with the partition sets mapped accordingly. At the end of Sect. 3 a more formal description of algorithm EMBED is provided. As a crucial ingredient, we need to derandomize a probabilistic result on random perfect matchings of a super regular graph (the Catching Lemma). This part, which we believe is of independent interest, is treated in Sect. 4.

2 Partitions and Enlargements

In this section we describe how to construct finer partitions of $V(H)$ and $V(G)$. This will be done by the algorithm PARTITION. We will only outline its two

major steps. As an input, H and G are the graphs described in Theorem 2 with initial partitions $V(H) = X_1 \cup \cdots \cup X_r$ and $V(G) = V_1 \cup \cdots \cup V_r$, and $|X_i| = |V_i| = n$, $i = 1, \ldots, r$.

PARTITION $V(H)$:

Let l be the unique positive integer so that

$$2\Delta^2 + 1 < 2^l \leq 4\Delta^2, \tag{2}$$

Partition each X_i into $t = 2^l$ sets $X_{i,j}$, $j = 1, \ldots, t$, of size m or $m+1$ where $m = \lfloor 2^{-l}n \rfloor$ so that each pair of distinct sets $(X_{i_1,j_1}, X_{i_2,j_2})$ spans in H a (possibly empty) matching.

To obtain the finer partition of $V(H)$, we will use an algorithmic version of a result on graph packing by Sauer and Spencer given in [14]. Given two graphs Γ and Γ' on the same n-vertex set V, we say that a bijection $\pi : V \to V$ is a *packing of Γ and Γ'* if $E(\Gamma) \cap \pi(E(\Gamma')) = \emptyset$, where $\pi(E(\Gamma')) = \{(\pi(u), \pi(v)) : (u, v) \in E(\Gamma')\}$.

Lemma 1. *Let Γ and Γ' be two graphs on an n-vertex set V satisfying $2\Delta(\Gamma)\Delta(\Gamma') < n$. There exists a polynomial-time algorithm PACK that finds a packing of Γ and Γ'.*

We apply algorithm PACK as follows. Given a graph Γ, the *square of Γ*, denoted by Γ^2, is the graph obtained from Γ by joining each pair of distinct vertices of Γ whose distance is at most 2 by an edge. Denote by Γ_i the subgraph of H^2 induced by X_i. Set $y = n - 2^l m$ and let Γ' be the vertex-disjoint union of $2^l - y$ cliques of order m and y cliques of order $m+1$. Note that $\Delta(\Gamma_i) \leq \Delta^2$ and $\Delta(\Gamma') = m$, so we have $2\Delta(\Gamma_i)\Delta(\Gamma') < n$. Hence, applying the algorithm PACK to Γ_i and Γ' as defined above yields a finer partition of X_i into 2^l sets $X_{i,j}$, of size m or $m+1$, which are independent in H^2. Clearly, the edges of H which go between any two such sets are pairwise disjoint. Thus, each pair of distinct sets $(X_{i_1,j_1}, X_{i_2,j_2})$ spans in H a (possibly empty) matching.

PARTITION $V(G)$:

Partition each V_i into $t = 2^l$ sets $V_{i,j}$, $j = 1, 2, \ldots, t$, of size m or $m+1$ so that $|V_{i,j}| = |X_{i,j}|$, $i = 1, 2, \ldots, r$, $j = 1, 2, \ldots, t$, and so that each pair of distinct sets $(V_{i_1,j_1}, V_{i_2,j_2})$ with $i_1 \neq i_2$ spans a super $(d, 2(\Delta^2+1)\delta)$-regular subgraph.

As the resulting partition sets have size at least $m \geq n/(2\Delta^2 + 2)$, the $2(\Delta^2+1)\delta$-regularity of all pairs follows immediately from the assumption that all pairs (V_i, V_j) are δ-regular. To establish the super regularity we have to control how the degree of every vertex splits between the new partition sets. For this we recall a slight extension of a result of Alon and Spencer given in [3]. It derandomizes a standard application of Chernoff's bound.

Lemma 2. *Let $\mathcal{F}$ be a family of k subsets of an n-element set Ω. There exists a polynomial-time algorithm HALF that partitions Ω into $\Omega = \Omega^+ \cup \Omega^-$ such that for each $F \in \mathcal{F}$,*

$$\frac{|F|}{2} - \frac{\beta}{2} \leq |F \cap \Omega^+|, |F \cap \Omega^-| \leq \frac{|F|}{2} + \frac{\beta}{2}, \tag{3}$$

where $\beta = \sqrt{2nlog(2k)}$.

We construct the finer partition of each V_i in l iterations where l is defined in (2). In the first iteration, we apply HALF to $\mathcal{F} = \{V_i\} \cup \{N(v) \cap V_i : v \notin V_i\}$. Thus, V_i and all of the neighborhoods in $\mathcal{F}$ get roughly halved and two new families are generated. In the second iteration, we apply HALF twice, once to each family. This generates four new families, etc.

More generally, the j-th iteration will consist of applying HALF once to each of the 2^{j-1} families that are generated in the $(j-1)$-st iteration. After the l-th iteration, each V_i is partitioned into $t = 2^l$ sets so that each set has size roughly m and also each neighborhood has shrunk in the same proportion. To ensure that the convention $|V_{i,j}| = |X_{i,j}|$ is satisfied for all $i = 1, \ldots, r$ and $j = 1, \ldots, t$, we arbitrarily move vertices around. This affects the degrees very little. The cumulative error introduced by this procedure is only $O(\beta) = o(n)$. The super $(d, 2(\Delta^2 + 1)\delta)$- regularity of the subgraphs spanned by pairs of these refined partition sets follows easily.

Below we outline a simple procedure ENLARGE which is performed merely for convenience. It smoothes out the recursive embedding described in the next section.

Given a bipartite graph $\Gamma = (U, V; E)$, a matching $M \subset E$ is called *saturating* if $|M| = \min(|U|, |V|)$.

ENLARGE $E(H)$ and $E(G)$:

(i) Add edges to form a supergraph H' of H so that each pair $(X_{i_1,j_1}, X_{i_2,j_2})$ spans a saturating matching.

(ii) Add edges between the pairs $(V_{i_1,j_1}, V_{i_2,j_2})$ with $i_1 = i_2$ (i.e. pairs with density 0) to form a supergraph G' of G so that each such pair spans in G' a super $(d, 2(\Delta^2 + 1)\delta)$-regular subgraph.

In step (ii), we may insert between each pair $(V_{i_1,j_1}, V_{i_2,j_2})$, $i_1 = i_2$, the same (our favorite) super $(d, 2(\Delta^2 + 1)\delta)$-regular graph.

Note that every embedding of H' into G' that maps $X_{i,j}$ onto $V_{i,j}$ for all i and j yields a desired embedding of H into G.

3 Embedding

After the finer partitions of $V(H)$ and $V(G)$ have been obtained, we rename the sets $X_{i,j}$ and $V_{i,j}$ to Y_i and W_i and restore the notation H and G for H'

and G'. More precisely, with $s = rt$, H is now an s-partite graph with partition $V(H) = Y_1 \cup \cdots \cup Y_s$, where $m+1 \geq |Y_1| \geq ... \geq |Y_s| \geq m$, so that each pair of distinct sets (Y_i, Y_j) spans a saturating matching. Similarly, G is now an s-partite graph with partition $V(G) = W_1 \cup \cdots \cup W_s$, $|W_j| = |Y_j|$ for each $j = 1, \ldots, s$, so that each pair of distinct sets (W_i, W_j) spans a super (d, ϵ)-regular graph where $\epsilon = 2(\Delta^2 + 1)\delta$.

The goal of this section is to outline how algorithm EMBED from Theorem 2 constructs an embedding of H into G so that each Y_j is mapped onto the set W_j.

Before describing the embedding, we introduce some definitions and notation. For every $1 \leq j \leq s-1$ and each vertex $x \in Y_{j+1} \cup \cdots \cup Y_s$, let $N_j(x)$ denote the set of precisely j neighbors of x which belong to $Y_1 \cup \cdots \cup Y_j$. Given a bijection f_j between $Y_1 \cup \cdots \cup Y_j$ and $W_1 \cup \cdots \cup W_j$, let $M_j(x) = f_j(N_j(x))$. Given f_j, for each $t = j+1, \ldots, s$, we define a bipartite auxiliary graph A_j^t with bipartition (Y_t, W_t) and edge set

$$E(A_j^t) = \{xv : x \in Y_t, v \in W_t \text{ and } uv \in E(G) \text{ for each } u \in M_j(x)\}. \qquad (4)$$

We call the graphs A_j^t *candidacy graphs* because the edges of A_j^t join a given vertex $x \in Y_t$ to all vertices of W_t which, after f_j embeds $Y_1 \cup \cdots \cup Y_j$ onto $W_1 \cup \cdots \cup W_j$, are still good candidates for the image of x.

We will embed H into G recursively. Let f_1 be any bijection between Y_1 and W_1. Assuming that there exists an embedding f_{j-1} of $H[Y_1 \cup \cdots \cup Y_{j-1}]$ into $G[W_1 \cup \cdots \cup W_{j-1}]$ so that the candidacy graphs A_{j-1}^t, $t = j, \ldots, s$, are super $(d^{j-1}, \epsilon_{j-1})$-regular, extend f_{j-1} to f_j by constructing a perfect matching $\sigma_j : Y_j \to W_j$ in A_{j-1}^j which makes the graphs A_j^t, $t = j+1, \ldots, s$ super (d^j, ϵ_j)-regular, and set $f_j(x) = f_{j-1}(x)$ if $x \in Y_1 \cup ... \cup Y_{j-1}$ and $f_j(x) = \sigma_j(x)$ if $x \in Y_j$. This operation of extending f_{j-1} to f_j will be designated by $f_j \leftarrow f_{j-1} + \sigma_j$. (Here and throughout we view a perfect matching as a bijection along the edges of a bipartite graph rather than as a set of edges.)

Note that the instance $j = s$ yields the desired embedding. The request to make all future graphs A_j^t super regular serves to carry over the recursion. The existence of the perfect matching σ_j follows from a result called the Four Graphs Lemma given in [13]. Its probabilistic ingredient is Lemma 4 below which we will refer to as the Catching Lemma. We will first explain how the Four Graphs Lemma works. Then, in the final section, we will outline how to derandomize the Catching Lemma.

Let $2 \leq j < t \leq s$ be fixed and consider the following graphs:

1. Let $\Gamma_1^j = A_{j-1}^j$.
2. Let $\Gamma_2 = \Gamma_2^{j,t}$ be the bipartite graph spanned by the pair (W_j, W_t) in G.
3. Let $\Gamma_3 = \Gamma_3^{j,t}$ denote a bipartite graph with bipartition (Y_j, W_t) such that $xw \in E(\Gamma_3^{j,t})$ if and only if $yw \in E(A_{j-1}^t)$, where y is the unique neighbor of x in Y_t. (If $|Y_j| = |Y_t| - 1$ then there is one vertex x_t in Y_j with no match in Y_t; in such a case we arbitrarily join x_t to $\lfloor d^{j-1}(m+1) \rfloor$ vertices of W_t; however, for clarity of exposition we do assume that $|Y_j| = |Y_t|$.)

4. Given a perfect matching σ in Γ_1^j, let $A_\sigma = A_\sigma^{j,t}$ be a subgraph of $\Gamma_3^{j,t}$ so that $xw \in E(A_\sigma^{j,t})$ if and only if both $xw \in E(\Gamma_3^{j,t})$ and $\sigma(x)w \in E(\Gamma_2^{j,t})$.

Observe that $\Gamma_3^{j,t}$ is isomorphic to A_{j-1}^t and $A_\sigma^{j,t}$ is isomorphic to A_j^t. Observe also that by the induction hypothesis, both Γ_1^j and $\Gamma_3^{j,t}$ are super $(d^{j-1}, \epsilon_{j-1})$-regular (in the "ugly" case when $|Y_j| = |Y_t| - 1$, adding the vertex x_t and its neighbors could slightly affect the super regularity of $\Gamma_3^{j,t}$; to be on the safe side one would need to double ϵ_{j-1}). Finally, by our assumption on G, $\Gamma_2^{j,t}$ is super (d, ϵ)-regular.

The Four Graph Lemma asserts (under one additional assumption) that most of the perfect matchings σ of Γ_1^j are such that the graphs $A_\sigma^{j,t}$, $t = j+1, ..., s$, are all super (d^j, ϵ_j)-regular, where $\epsilon_j = h(\epsilon_{j-1})$ and h is an easily computable function which decreases to 0 when its argument does so. The additional demand is that the endpoints of each edge of Γ_1^j have about $d^j m$ common neighbors in each Y_t. Precisely, for every edge $vu \in \Gamma_1^j$ and for each $t = j+1, ..., s$, we require that

$$(d^j - \epsilon)m < |N_{\Gamma_2}(v) \cap N_{\Gamma_3}(u)| < (d^j + \epsilon)m \ . \tag{5}$$

It can easily be conformed with by throwing away the edges of Γ_1^j which fail to have this property. We omit the description of algorithm PEEL which does it. The residual subgraph $\bar{\Gamma}_1^j$ is still super regular with slightly bigger second parameter (c.f. Fact 1 in [13]). From now on we will be assumimg that Γ_1^j does satisfy this additional assumption, i.e. we set $\Gamma_1^j \leftarrow \bar{\Gamma}_1^j$. This constraint guarantees that for every perfect matching σ of Γ_1^j , and for each t, the degree $d_{A_\sigma}(x)$ of each vertex $x \in Y_j$ is close to $d^j m$. Observe that $d_{A_\sigma}(v)$ for $v \in W_t$ is precisely equal to the number of edges of σ which connect a vertex of $N_{\Gamma_2}(v)$ with a vertex of $N_{\Gamma_3}(v)$. The Catching Lemma assures that this number is right for most of the σ's. Hence, provided we can derandomize the Catching Lemma, the second condition in the definition of a super regular graph is taken care of. To establish the ϵ_j-regularity alone of $A_\sigma^{j,t}$ we rely on the following criterion from [1].

Lemma 3. *If $\Gamma = (U, V; E)$, $|U| = |V| = m$, is a bipartite graph with at least $(1 - 5\epsilon)m^2/2$ pairs of vertices $w_1, w_2 \in U$ satisfying*

(i) $deg(w_1), deg(w_2) > (d - \epsilon)m$, and
(ii) $|N(w_1) \cap N(w_2)| < (d + \epsilon)^2 m$,

then Γ is $(16\epsilon)^{1/5}$-regular.

Note that $|N_{A_\sigma}(v_1) \cap N_{A_\sigma}(v_2)|$ equals the number of edges of σ which connect a vertex of $N_{\Gamma_2}(v_1) \cap N_{\Gamma_2}(v_2)$ with a vertex of $N_{\Gamma_3}(v_1) \cap N_{\Gamma_3}(v_2)$. For most pairs $v_1, v_2 \in W_t$ neither of these sets is too large (call these pairs *good*) and, by the Catching Lemma again, there cannot be too many edges of a random σ between them. The probability of failure is exponentially small, so that we are in position

to demand that a random perfect matching of Γ_1^j simultaneously inserts the right number of edges between each pair $N_{\Gamma_2}(v),\ N_{\Gamma_3}(v)$ for all $v \in\ W_t$, $t = j+1, ..., s$, as well as between each pair $N_{\Gamma_2}(v_1) \cap\ N_{\Gamma_2}(v_2),\ N_{\Gamma_3}(v_1) \cap\ N_{\Gamma_3}(v_2)$ for all good pairs $v_1, v_2 \in W_t$, $t = j+1, ..., s$.

In the next section we describe an algorithm CATCH which constructs the desired perfect matching. We wrap up this section by a description of algorithm EMBED.

Algorithm EMBED

Input: r-partite graphs H and G as in Theorem 2.
Output: An embedding f of H into G so that each X_i is mapped onto V_i, $i = 1, \ldots, r$.

Phase 1:

1. Apply PARTITION to $V(H)$ and $V(G)$ and denote the output by $V(H) = Y_1 \cup \cdots \cup Y_s$ and $V(G) = W_1 \cup \cdots \cup W_s$, where $m \le |Y_1| = |W_1| \le ... \le |Y_s| = |W_s| \le m+1$.
2. Apply ENLARGE to H and G with output H' and G'; $H \leftarrow H'$ and $G \leftarrow G'$.

Phase 2:

1. Let f_1 be any bijection from Y_1 to W_1.
2. $j \leftarrow 2$; WHILE $j \le s$ DO:
   (a) Apply PEEL to Γ_1^j; denote output by $\bar{\Gamma}_1^j$; $\Gamma_1^j \leftarrow \bar{\Gamma}_1^j$.
   (b) Apply CATCH to the graph Γ_1^j and to the pairs $(N_{\Gamma_2^{j,t}}(v),\ N_{\Gamma_3^{j,t}}(v))$, $v \in W_t$, $t = j+1, ..., s$, $(N_{\Gamma_2^{j,t}}(v_1) \cap N_{\Gamma_2^{j,t}}(v_2),\ N_{\Gamma_3^{j,t}}(v_1) \cap N_{\Gamma_3^{j,t}}(v_2))$ for all good pairs $v_1, v_2 \in W_t$, $t = j+1, ..., s$; denote the output by σ_j.
   (c) $f_j \leftarrow f_{j-1} + \sigma_j$, $j \leftarrow j+1$.
3. $f \leftarrow f_s$

4 Catching

The Catching Lemma appeared first in [13] (c.f Lemma 1). The name comes from the fact that a pair of sets catches in between a number of edges of a random perfect matching which is close to its expectation. In our application to the Four Graphs Lemma described in Sect. 3, we actually have $k = \Theta(m^2)$ of these pairs.

Lemma 4 (Catching Lemma). *For every choice of three real numbers $0 < d, d_1, d_2 < 1$ there exists $\epsilon > 0$, $c = c(\epsilon)$, $0 < c < 1$, $m_0(\epsilon)$ and a function $g(x) \to 0$ as $x \to 0$ such that the following holds. Let Γ be a super (d, ϵ)-regular*

graph with bipartition (V_1, V_2), $|V_1| = |V_2| = m > m_0(\epsilon)$, *and let* $S_i \subseteq V_1$, $d_1 m \leq |S| = s_i \leq d_2 m$, *and* $T_i \subseteq V_2$, $d_1 m \leq |T| = t_i \leq d_2$, $i = 1, 2, ..., k$, *where* $k = O(m^2)$. *Then, for a perfect matching* σ *of* Γ, *drawn randomly with the uniform distribution, the event that for each* $i = 1, 2, ..., k$

$$s_i t_i/m - g(\epsilon)m < |\sigma(S_i) \cap T_i| < s_i t_i/m + g(\epsilon)m \tag{6}$$

holds with probability at least $1 - c^m$.

As it was explained in Sect. 3 we need to design a subroutine CATCH, which derandomizes the above lemma simultaneously for all pairs (S_i, T_i), i.e. a procedure which constructs the desired perfect matching of Γ.

How to derandomize this lemma, i.e. how to effectively find the required perfect matching? The most straightforward approach seems to be the Erdős-Selfridge method of conditional probabilities (see [4]) and the technique of pessimistic estimators (see [12]). In basic terms, pessimistic estimators are easily computable functions that are used to provide an upper bound on probabilities that cannot be computed efficiently. Here, the situation calls for just such a technique, since it is not known how to compute the exact probability of the event that there exists an i such that $|\sigma(S_i) \cap T_i|$ falls outside the required interval. However, as these upper bounds hold true only as long as the size of the residual subgraph of Γ remains reasonably large, at some point we need to stop the procedure and complete the current matching by any matching we can possibly find in the leftover graph. The problem we immediately face is that this leftover subgraph, though ϵ'-regular for some ϵ', does not need to be super regular and therefore may not have a perfect matching at all.

To overcome this difficulty we invoke a new idea. Take a subgraph Γ' of Γ on half of the vertices so that the other half is super regular and every set S_i and T_i is roughly halved, and find a suitable almost perfect matching in Γ'. By suitable we mean one which satisfies (6) with respect to the halves of the sets S_i and T_i, and with $g(\epsilon)$ replaced by $\frac{1}{3}g(\epsilon)$. We then move the leftover vertices to the other half which still remains super regular and repeat as long as the remaining part is larger than $\frac{1}{3}g(\epsilon)m$. Finally we find a perfect matching in the leftover subgraph of Γ which is so small that it cannot affect the required property of the constructed perfect matching σ. (The third $1/3$ serves as a cushion for all the inaccuracies we commit along the way.)

More formally, we prove the following theorem.

Theorem 3. *There is a polynomial time algorithm CATCH which for a given super* (d, ϵ)*-regular bipartite graph* Γ *and* $k = O(n^2)$ *pairs of sets* (S_i, T_i), *as in Lemma 4, finds a perfect matching* σ *of* Γ *such that for each* $i = 1, ..., k$, *the inequalities (6) hold.*

The algorithm CATCH uses 3 subroutines: HALF, MATCH and HALL. Procedure HALL just finds a perfect matching in any bipartite graph satisfying Hall's condition. We may use, for instance, the algorithm of Hopcroft and Karp, given in [6], of complexity $m^{5/2}$.

Procedure HALF is defined in Lemma 2.

Procedure MATCH does the same job as CATCH, but in the space of almost perfect matchings, i.e. matchings covering all but $\sqrt{\epsilon}n$ vertices on each side. With the same input as CATCH it outputs an almost perfect matching σ' which satisfies condition (6) with $g(\epsilon)$ replaced by $\frac{1}{3}g(\epsilon)$, along with a pair of leftover sets L_U and L_V, both of size $\sqrt{\epsilon}n$.

Algorithm MATCH is based on a probabilistic lemma very similar to Lemma 4 above. To derandomize that lemma we indeed apply the Erdős-Selfridge method of conditional probabilities. As we are now after an *almost* perfect matching, the problem of finishing it off no longer exists. For details see the full size paper.

We conclude this extended abstract by a description of the algorithm CATCH. Given two disjoint matchings σ and σ', their union will be denoted by $\sigma + \sigma'$.

Algorithm CATCH

Input: A super (d, ϵ)-regular graph $\Gamma = (U, V; E)$ with $|U| = |V| = m$, and sets (S_i, T_i), $i = 1, ..., k$, as in Theorem 3.

Output: A perfect matching $\sigma : U \to V$ that satisfies, for each $i = 1, 2, \ldots, k$, inequalities (6).

1. $\sigma \leftarrow \emptyset$.
2. WHILE $|U| > \frac{1}{3}g(\epsilon)m$ DO:
   (a) Apply HALF to $\{U, S_i, i = 1, ..., k, N_\Gamma(v), v \in V\}$. Denote the output by U_1 and U_2.
   (b) Apply HALF to $\{V, T_i, i = 1, ..., k, N_\Gamma(u), u \in U\}$. Denote the output by V_1 and V_2.
   (c) Apply MATCH to $\Gamma[U_1, V_1]$ and $(S_i \cap U_1, T_i \cap V_1)$, $i = 1, ..., k$. Denote the output by σ', L_U and L_V.
   (d) $\sigma \leftarrow \sigma + \sigma'$,
   $U \leftarrow U_2 \cup L_U$, $V \leftarrow V_2 \cup L_V$,
   $\Gamma \leftarrow \Gamma[U, V]$,
   $S_i \leftarrow S_i \cap U$, $T_i \leftarrow T_i \cap V$, $i = 1, ..., k$.
3. Apply HALL to $\Gamma[U, V]$. Denote the output by σ';
   $\sigma \leftarrow \sigma + \sigma'$.

References

1. N. Alon, R. Duke, H. Leffman, V. Rödl, and R. Yuster, "The algorithmic aspects of the regularity lemma", *Journal of Algorithms*, vol. 16 (1994), pp. 80-109.
2. N. Alon, V. Rödl, and A. Ruciński, "Perfect matchings in ϵ-regular graphs", *The Electronic J. of Combin.*, vol. 5(1) (1998), # R13.
3. N. Alon and J. Spencer, *The Probabilistic Method*, Wiley, New York, 1992.
4. P. Erdös and J. L. Selfridge, "On a combinatorial game", *Journal of Combinatorial Theory, Series A*, vol. 14 (1973), pp. 298-301.
5. A. Hajnal and E. Szemerédi, "Proof of a conjecture of Erdős", Combinatorial Theory and its Applications, vol. II (P. Erdős, A. Rényi, and V.T. Sós eds), *Colloq. Math. Soc. J. Bolyai 4*, North Holland, Amsterdam, 1970, pp. 601-623.

6. J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM J. Comput.*, vol. 2 (1973), pp. 225-231.
7. S. Janson, T. Łuczak, and A. Ruciński, *Topics in Random Graphs*, Wiley, New York, 1999.
8. J. Komlós, G. N. Sárközy, and E. Szemerédi, "Blow-up lemma", *Combinatorica*, vol. 17 (1997), pp. 109-123.
9. J. Komlós, G. N. Sárközy, and E. Szemerédi, "An algorithmic version of the blow-up lemma", *Random Structures and Algorithms*, vol. 12 (1998), pp. 297-312.
10. J. Komlós, G. N. Sárközy and E. Szemerédi "On the Pósa-Seymour conjecture J. Graph Theory", *Journal of Graph Theory*, to appear
11. J. Komlós, G. N. Sárközy and E. Szemerédi "Proof of the Alon-Yuster conjecture" submitted
12. P. Raghavan, "Probabilistic construction of deterministic algorithms: Approximating packing integer programs", *Journal of Computer and System Sciences*, vol. 37 (1988), pp. 130-143.
13. V. Rödl and A. Ruciński, "Perfect matchings in ϵ-regular graphs and the blow-up lemma", submitted.
14. N. Sauer and J. Spencer, "Edge disjoint placement of graphs", *J. Comb. Th. B*, vol. 25 (1978), pp. 295-302.
15. E.Szemerédi " Partitions of graphs" *Problems Combin. et Theorie des graphes, Edition du C.N.R.S.* vol. 260 (1978), pp. 399-402.

Deterministic Hypergraph Coloring and Its Applications

Chi-Jen Lu

Computer Science Department
University of Massachusetts at Amherst
cjlu@cs.umass.edu

Abstract. Given a hypergraph and a set of colors, we want to find a vertex coloring to minimize the size of any monochromatic set in an edge. We give deterministic polynomial time approximation algorithms with performances close to the best bounds guaranteed by existential arguments. This can be applied to support divide and conquer approaches to various problems. We give two examples. For deterministic approximate DNF counting, this helps us explore the importance of a previously ignored parameter, the maximum number of appearance of any variable, and construct algorithms that are particularly good when this parameter is small. For partially ordered sets, we are able to constructivize the dimension bound given by Füredi and Kahn [5].

1 Introduction

A hypergraph $H(V, E)$ consists of a set V of nodes and a set E of edges, where each edge is a subset of nodes. An undirected graph is just a hypergraph where each edge contains exactly two nodes. There are four parameters associated with a hypergraph:

- $n = |V|$, number of nodes.
- $m = |E|$, number of edges.
- $t = \max\{|e| : e \in E\}$, size of the largest edge. We will write $E \subseteq V^{\leq t}$.
- $d = \max\{deg(v) : v \in V\}$, where $deg(v) = |\{e \in E : v \in e\}|$. d is called the degree of H.

Given k colors, we want to color nodes so that no color appears more than c times in any edge. Clearly $c \geq \mu \equiv \frac{t}{k}$, and we want to have $c = \mu\alpha$ for α as small as possible. This problem was studied before by Srinivasan [11], who gave the following (nonconstructive) existential bound for c:

$$c = \begin{cases} O(\mu) & \text{if } \mu = \Omega(\log d)) \\ O(\frac{\log d}{\log((\log d)/\mu)}) & \text{otherwise.} \end{cases}$$

We give a deterministic polynomial time algorithm for finding a coloring with

$$c = \begin{cases} O(\mu) & \text{if } \mu = \Omega(\log(td)) \\ O(\frac{\log(dt)}{\log((\log(dt))/\mu)}) & \text{otherwise.} \end{cases} \quad (1)$$

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 35-46, 1998.

When $t = d^{O(1)}$ or $\mu = \Omega(\log(td))$, our bound for c is within a constant factor of the bound given by Srinivasan. Also, for our applications below, our bound suffices. In fact, a much more involved method can actually constructivize Srinivasan's bound for c, but we leave it to a later paper.

Notice that such a k-coloring partitions the original hypergraph into k sub-hypergraphs, one for each color. In each sub-hypergraph, every edge now has at most c edges. This turns out to support some divide and conquer approaches. We will give two examples.

Our first application is to the deterministic DNF approximate counting problem. Given a DNF formula F of n variables and m terms, we want to estimate its *volume*, defined as

$$vol(F) = P_{x \in \{0,1\}^n}[F(x) = 1],$$

within an additive error ϵ. Luby, Velicković, and Wigderson [7], following the work of Nisan [8][9], gave a deterministic $2^{O(\log^4 \frac{nm}{\epsilon})}$ time algorithm. Luby and Velicković [6] gave a deterministic $2^{(\log \frac{m \log n}{\epsilon})(\frac{1}{\epsilon})(2^{O(\sqrt{\log \log \frac{m}{\epsilon}})})}$ time algorithm, which is good when large error ϵ is allowed. Note that a DNF formula F can be naturally modeled by a hypergraph, with nodes corresponding to variables and edges corresponding to terms. Now the degree d of the hypergraph indicates the maximum number of times a variable is read in F. This parameter has not received attention before for this problem, and is our focus here. We construct deterministic algorithms with running times $2^{O((\log \frac{mn}{\epsilon})(\log^3 \frac{d \log m}{\epsilon}))}$ and $2^{(\log \frac{m \log n}{\epsilon})(\log \frac{1}{\epsilon})(2^{O(\sqrt{\log(d \log \frac{m}{\epsilon})})})}$ respectively. Note that d is at most m, so our first algorithm is never worse than that of Luby, Velicković, and Wigderson [7], and is particularly good when d is small and ϵ is large. Our second algorithm is better than that of Luby and Velicković [6] when $d \leq 2^{\frac{1}{2}\log^2 \frac{1}{\epsilon}}$, and is better than our first algorithm when $d \leq 2^{4(\log \log \frac{m}{\epsilon})^2}$.

Our second application is to dimensions of partially ordered sets (posets). Let $(P, <)$ be a poset. Its dimension, denoted as $dim(P)$, is defined to be the minimum number of linear extensions $L_1, \ldots, L_d$ such that $P = L_1 \cap \cdots \cap L_d$ (i.e., $x < y$ iff $x <_{L_i} y$ for all i). For $x \in P$, let $U(x) = \{y \in P : y \geq x\}$ be the set of upper bounds for x, and let $C(x) = \{y \in P : y \geq x \text{ or } y \leq x\}$ be the set of elements comparable to x. Füredi and Kahn [5] gave the following existential bound: for some constants c_1 and c_2,

$$dim(P) \leq r \equiv \min\{c_1 t \log^2 t, c_2 u \log |P|\},$$

where $t = \max_{x \in P} |C(x)|$ and $u = \max_{x \in P} |U(x)|$. One key ingredient in their proof is the hypergraph coloring problem, where a poset $(P, <)$ is modeled by a hypergraph $H(V, E)$ with $V = P$ and $E = \{U(x) : x \in P\}$. Using our coloring algorithm, together with other ideas, we are able to constructivize their existence bound. That is, we give a deterministic polynomial time algorithm for finding $O(r)$ linear extensions with intersection equal to the given poset.

We believe that there should be more applications of our hypergraph coloring algorithm.

2 Hypergraph Coloring

Consider a hypergraph $H(V, E)$ with $n = |V|$, $m = |E|$, $E \subseteq V^{\leq t}$, and degree d. We want to color nodes with k colors such that no edge contains c nodes of the same color. For a coloring, call an edge *bad* if it contains c nodes of the same color, and call a set of edges bad if all edges in it are bad. Our goal is to find a *good* k-coloring γ such that no edge is bad. If we choose γ randomly, then for an edge e,

$$P_\gamma[e \text{ is bad}] \leq \binom{t}{c}(\frac{1}{k})^c k \leq (\frac{3t}{ck})^c k.$$

From the Lovász local lemma [4], a good k-coloring existes provided $(\frac{3t}{ck})^c kdt \leq \frac{1}{4}$. However, a random k-coloring sometimes is good with exponentially small probability, and it is not obvious how to find such a good coloring, even probabilistically. Beck [2] had the first success in derandomizing the local lemma, and Alon [1] later adapted Beck's idea to derandomize more applications of the local lemma. We will follow their approach closely.

Our main result in this section is a deterministic polynomial time algorithm to find a good k-coloring, satisfying

$$(\frac{9t}{ck})^c \frac{k}{3}(dt)^4 \leq \frac{1}{4},$$

which leads to the bound for c given in Eq (1). For this value of c, a random $\frac{k}{3}$-coloring turns an edge e bad with probability less than $p \equiv (\frac{1}{dt})^4$. We first give a randomized algorithm and then we derandomize it.

2.1 A Randomized Algorithm

There will be at most three phases, each using a distinct set of $\frac{k}{3}$ colors. The intuition is that a random $\frac{k}{3}$-coloring is unlikely to have a large cluster of bad edges and that each bad cluster can be recolored separately, for a proper definition of " cluster". For a hypergraph H, its line graph L_H is the graph where nodes are the edges of H and two nodes are adjacent iff the corresponding edges in H intersect. Let $L_H^{(a,b)}$ be the graph with the same node set but now two nodes are adjacent iff their distance is exactly a or b in L_H. Call a set of edges in H an (a, b)-tree if the corresponding nodes in $L_H^{(a,b)}$ are connected. Our algorithm consists of phases. In the first phase, we find a $\frac{k}{3}$-coloring such that all bad $(1, 2)$-trees have size $O(dt \log m)$. Then we try to recolor each bad $(1, 2)$-tree separately, using a new set of $\frac{k}{3}$ colors. If m is small, the recoloring can be done in the second phase. Otherwise we need another phase, using another set of $\frac{k}{3}$ colors.

Phase 1:

In this phase, we will find a $\frac{k}{3}$-coloring such that all bad $(1, 2)$-trees have size $O(dt \log m)$. First we need the following lemma:

Lemma 1. *For some $u = O(\log m/\log(dt))$, the probability that a random $\frac{k}{3}$-coloring has a bad $(2,3)$-tree of size u is $(1/m)^{\Omega(1)}$.*

Proof: Any two edges of a $(2,3)$-tree have no node in common, and the events of each being bad are independent. As there are at most $\frac{m}{((dt)^3-1)u+1}\binom{(dt)^3u}{u}$ $(2,3)$-trees of size u, and each one is bad with probability at most p^u, the probabilty that a random $\frac{k}{3}$-coloring has a bad $(2,3)$-tree of size u is at most

$$m(3(dt)^3p)^u \leq m(\frac{3}{dt})^u = (1/m)^{\Omega(1)}.$$

□

As a $(1,2)$-tree of size dtu must contain a $(2,3)$-tree of size u, a random $\frac{k}{3}$-coloring with high probabilty will have no $(1,2)$-tree of size $dtu = O(dt\log m)$. In the next section, we will show how to find such a $\frac{k}{3}$-coloring deterministically, by using the standard technique of conditional probability with a pessimistic estimator.

Phase 2:

Suppose we have found a $\frac{k}{3}$-coloring with no bad $(1,2)$-tree of size dtu. Then we try to recolor these bad $(1,2)$-trees . Let $T = (V_T, E_T)$ be a bad $(1,2)$-tree . When we recolor nodes in T, those good edges intersecting T are also affected, and we want to make sure that they won't turn bad after the recoloring. So together with T, we also take into account those good edges but with nodes not in T removed, and consider the coloring problem for this hypergraph S. More precisely, $S = (V_S, E_S)$ where $V_S = V_T$ and $E_S = \{e \cap V_S : e \in E_H, e \cap V_S \neq \emptyset\}$. It's easy to see that the condition of the local lemma still holds and a good $\frac{k}{3}$-coloring exists for S. We will use a different set of $\frac{k}{3}$ colors in this phase. If we find a good $\frac{k}{3}$-coloring for S, then after this recoloring, no edge of H intersecting T is bad. Now as each edge of H intersects at most one bad $(1,2)$-tree , we can repeat this recoloring process for each bad $(1,2)$-tree , using the same new set of $\frac{k}{3}$ colors.

Note that now $|E_S| \leq (dt)^2u$. Suppose $\sqrt{\log m/\log\log m} \leq dt$. Then the probability that a random $\frac{k}{3}$-coloring has a bad edge in S is at most

$$|E_S|p < (dt)^4(\frac{1}{dt})^4 = 1.$$

We can find a good $\frac{k}{3}$-coloring in deterministic polymonial time, using again the technique of conditional probability.

Otherwise, when $dt < \sqrt{\log m/\log\log m}$, we can find a $\frac{k}{3}$-coloring such that all bad $(1,2)$-trees have size at most

$$O(dt\log((dt)^2\log m)/\log(dt)) = O(\sqrt{\log m\log\log m}/\log(dt)),$$

similarly to phase 1. Then we enter pahse 3.

Phase 3:

Now as $t \leq \sqrt{\log m / \log\log m}$, each bad $(1,2)$-tree has $O(\log m / \log(dt))$ nodes, and we can use an exhaustive search to find a good $\frac{k}{3}$-coloring in deterministic $(\frac{k}{3})^{O(\log m/\log(dt))} = m^{O(1)}$ time.

2.2 Derandomization of Phase 1

Let R denote the set of all $(2,3)$-trees of size $u = O(\frac{\log m}{\log(dt)})$, and let B denote the event that some tree in R is bad. From Lemma 1, we know that the bad event B is unlikely to happen under a random $\frac{k}{3}$-coloring . But how do we find a good coloring deterministically? The idea is to use the standard technique of conditional probability with a pessimistic estimator, introduced by Raghavan [10]. We want to color nodes one by one. The color of each node is chosen to minimize the probability of having the bad event B if we randomly $\frac{k}{3}$-color the remaining nodes. The hope is that the final coloring is a good one because the final conditional probability, which is either 0 or 1, is at most the origional unconditional one, which is less than 1. However, it's not easy to compute the exact conditional probability at each step here. So we use a pessimistic estimator instead.

Suppose that we have already assigned colors $\gamma_1, \ldots, \gamma_i$ to nodes $v_1, \ldots, v_i$. We will overestimate the conditional probability

$$P_i(\gamma_1, \ldots, \gamma_i) \equiv P_{\gamma_{i+1},\ldots,\gamma_n}[B \mid \gamma_1, \ldots, \gamma_i],$$

by the following pessimistic estimator:

$$A_i(\gamma_1, \ldots, \gamma_i) = \sum_{T \in R} \prod_{e \in T} \sum_{I \subseteq e, |I|=c} P_{\gamma_{i+1},\ldots,\gamma_n}[\mathrm{mono}(I) \mid \gamma_1, \ldots, \gamma_i],$$

where $\mathrm{mono}(I)$ denotes the event that I is monochromatic. It's easy to see that $P_i(\gamma_1, \ldots, \gamma_i) \leq A_i(\gamma_1, \ldots, \gamma_i)$ for all i and all $\gamma_1, \ldots, \gamma_i$, and also that $A_0 = (1/m)^{\Omega(1)}$ from Lemma 1. Now,

$$\begin{aligned} A_i(\gamma_1, \ldots, \gamma_i) &= \sum_{T \in R} \prod_{e \in T} E_{\gamma_{i+1}} \sum_{I \subseteq e, |I|=c} P_{\gamma_{i+2},\ldots,\gamma_n}[\mathrm{mono}(I) \mid \gamma_1, \ldots, \gamma_{i+1}] \\ &= \sum_{T \in R} E_{\gamma_{i+1}} \prod_{e \in T} \sum_{I \subseteq e, |I|=c} P_{\gamma_{i+2},\ldots,\gamma_n}[\mathrm{mono}(I) \mid \gamma_1, \ldots, \gamma_{i+1}] \\ &= E_{\gamma_{i+1}} A_i(\gamma_1, \ldots, \gamma_{i+1}), \end{aligned}$$

where the second equality is because each edge e in T intersects no other edges in T. We pick γ_{i+1} to minimize $A_i(\gamma_1, \ldots, \gamma_i, \gamma_{i+1})$. Then we have

$$1 > A_0 \geq A_1(\gamma_1) \geq A_2(\gamma_1, \gamma_2) \geq \cdots \geq A_n(\gamma_1, \ldots, \gamma_n) \geq P_n(\gamma_1, \ldots, \gamma_n).$$

$P_n(\gamma_1, \ldots, \gamma_n)$ is either 1 or 0 dependending on whether the coloring $\gamma_1, \ldots, \gamma_n$ results in a bad $(2,3)$-tree of size u. As $P_n(\gamma_1, \ldots, \gamma_n) < 1$, there is no bad $(2,3)$-tree of size u, and we have found a good $\frac{k}{3}$-coloring .

It remains to show that for any i and any $\gamma_1, \ldots, \gamma_i$, $A_i(\gamma_1, \ldots, \gamma_i)$ can be computed efficiently. There are $m^{O(1)}$ $(2,3)$-trees of size u in R. It can be shown that enumerating all of them takes polynomial time. We omit the proof here. For each $(2,3)$-tree T and any edge e in T, $\sum_{I \subseteq e, |I|=c} P_{\gamma_{i+1},\ldots,\gamma_n}[\text{mono}(I) \mid \gamma_1, \ldots, \gamma_i]$ also can be easily computed. So $A_i(\gamma_1, \ldots, \gamma_i)$ can be computed in deterministic polynomial time.

3 DNF Approximate Counting

Each finite set is associated with a natural distribution, the uniform distribution over its elements, and we won't make the distinction between a set and its natural distribution when it's clear from the context. Given a DNF formula F on n variables, we'd like to know its volume, $vol(F) = P_{x \in \{0,1\}^n}[F(x) = 1]$. Valiant [12] has shown that it's #P-complete to compute the exact value, so we settle for an approximation. The standard approach is to find a pseudorandom distribution using many fewer random bits that can still fool F.

Definition 1 *A function $g : \{0,1\}^r \to \{0,1\}^n$ is called an ϵ-generator for a boolean function $F : \{0,1\}^n \to \{0,1\}$ if*

$$|P_{x \in \{0,1\}^n}[F(x) = 1] \Leftrightarrow P_{y \in \{0,1\}^r}[F(g(y)) = 1]| \leq \epsilon.$$

A function g is called an ϵ-generator for a class of boolean functions if it is an ϵ-generator for each function in this class.

So the algorithm for approximating $vol(F)$ is to find an ϵ-generator g for F and then compute $\frac{1}{2^r}\sum_{y \in \{0,1\}^r} F(g(y))$, the expected value of F over the pseudorandom distribution generated by g. The running time is proportional to 2^r, and the key point is to reduce r. Notice that we can have different function g for different F.

Clearly there are three important parameters that help determine the difficulty of this problem: the number n of variables, the number m of terms, and the error ϵ allowed. We discover the importance of another parameter d, the maximum number of terms that a variable can appear. Let DNF_d denote the set of DNF formulas with each variable appearing in at most d terms. Such formulas are usually called *read-d-times* DNF formulas. In the following, we will also assume that each term in a formula F contains at most $t = \log \frac{m}{\epsilon}$ literals. This is because we can always remove those terms containing more than t literals to get another formula F' such that $|P_x[F(x) = 1] \Leftrightarrow P_x[F'(x) = 1]| \leq \epsilon$, and then consider the formula F' instead. Let $t\text{DNF}_d$ denote the set of DNF_d formulas with no term containing more than t literals. This is the class of formulas we consider in this section. For convenience, we also assume that $n = m^{\Theta(1)}$.

A $t\text{DNF}_d$ formula F of n variables and m terms can be seen as a hypergraph $H(V,E)$ with $|V| = n$, $|E| = m$, $E \subseteq V^{\leq t}$ and degree d. We can use the algorithm in the previous section to find a k-coloring of variables such that no c literals are monochromatic in a term, for some k and c to be chosen later.

Let V_i, $1 \leq i \leq k$, be the set of variables with color i. If we fix values to all variables not in V_i, we get a $c\text{DNF}_d$ formula on V_i. Suppose that for $1 \leq i \leq k$, $g_i : \{0,1\}^{r_i} \to \{0,1\}^{|V_i|}$ is an ϵ-generator for all $c\text{DNF}_d$ formulas on the variable set V_i. Define $g : \{0,1\}^r \to \{0,1\}^n$, with $r = r_1 + \cdots + r_k$ and $n = |V_1| + \cdots + |V_k|$, such that those bits corresponding to V_i are generated by g_i.

Lemma 2. *The function g defined above is a $k\epsilon$-generator for F.*

Proof: For $1 \leq i \leq k$, let U_i denote the uniform distribution over $\{0,1\}^{|V_i|}$ for the variables in V_i, and let S_i be the corresponding pseudorandom distribution generated by g_i. Let D_i denote the distribution $S_1 \times \cdots \times S_i \times U_{i+1} \times \cdots \times U_k$, and let D'_i denote the distribution $S_1 \times \cdots \times S_{i-1} \times U_{i+1} \times \cdots \times U_k$. For $y \in D'_i$ let F_y denote the resulting formula from F by assigning the value y to the corresponding variables. F_y is a $c\text{DNF}_d$ formula on variable set V_i. Then

$$\begin{aligned}
|vol(F) \Leftrightarrow P_{x \in D_k}[F(x) = 1]| &= |P_{x \in D_0}[F(x) = 1] \Leftrightarrow P_{x \in D_k}[F(x) = 1]| \\
&\leq \sum_{i=0}^{k-1} |P_{x \in D_i}[F(x) = 1] \Leftrightarrow P_{x \in D_{i+1}}[F(x) = 1]| \\
&\leq \sum_{i=0}^{k-1} E_{y \in D'_i} |P_{z \in S_i}[F_y(z) = 1] \Leftrightarrow P_{z \in U_i}[F_y(z) = 1]| \\
&\leq k\epsilon.
\end{aligned}$$

D_k is the pseudorandom distribution generated by g. So g is a $k\epsilon$-generator for F. □

It remains to find such ϵ-generators for $c\text{DNF}_d$. We will give two constructions according to two different values of k and c.

3.1 Construction I, $c = O(\log \frac{dt}{\epsilon})$ and $k = O(\frac{t}{c})$

For a DNF formula G, a subformula of G is a formula with some of G's terms removed. Let $l = 2^c \ln \frac{1}{\epsilon} = (\frac{dt}{\epsilon})^{O(1)}$ and $m' = d(l \Leftrightarrow 1)c = (\frac{dt}{\epsilon})^{O(1)}$. We will see that any $c\text{DNF}_d$ formula G has a subformula of at most m' terms with almost the same volume. This suggests the following lemma.

Lemma 3. *Suppose that G is a $cDNF_d$ formula and g is an ϵ-generator for all subformulas of G with at most m' terms. Then g is a 2ϵ-generator for G.*

Proof: When G, after simplification, has at most m' terms, g is certainly an ϵ-generator for G. When G has more than m' terms, it has l disjoint terms because otherwise some variable would appear more than $m'/((l \Leftrightarrow 1)c) = d$ times. Let T denote the OR of those l disjoint terms. Then

$$1 \geq vol(G) \geq vol(T) > 1 \Leftrightarrow (1 \Leftrightarrow 2^{-c})^l \geq 1 \Leftrightarrow e^{-2^{-c} 2^c \ln 1/\epsilon} = 1 \Leftrightarrow \epsilon.$$

As g is an ϵ-generator for T, we have

$$1 \geq P_y[G(g(y)) = 1] \geq P_y[T(g(y)) = 1] \geq vol(T) \Leftrightarrow \epsilon \geq 1 \Leftrightarrow 2\epsilon.$$

Then $|vol(G) \Leftrightarrow P_y[G(g(y)) = 1]| \leq 2\epsilon$. □

It remains to show how to find such an ϵ-generator for all subformulas with at most m' terms from any $G \in c\mathrm{DNF}_d$. We will show that the generator of Luby, Velicković, and Wigderson [7] can be slightly modified to suit this purpose. For more detail, please refer to [7].

First, we fix the following parameters:

- $n' = cm' = (\frac{dt}{\epsilon})^{O(1)}$,
- $b = \log \frac{4n'm'}{\epsilon} = O(\log \frac{dt}{\epsilon})$,
- $s = b^2 = O(\log^2 \frac{dt}{\epsilon})$,
- $r = 24cb^3 = O(\log^4 \frac{dt}{\epsilon})$, and
- $\delta = \frac{\epsilon}{4m'n'(3cr)^{2b}} = (\frac{1}{2})^{o(\log^2 \frac{dt}{\epsilon})}$.

Following [7], we want to construct a set system. For any subformula T of of G with m' terms, we call a family of n subsets $S_1, \ldots, S_n \subseteq \{1, \ldots, r\}$ *good* for T if they satisfy the following two conditions:

- for any variable x_i of T, $|S_i| \leq s$, and
- for any variable x_i of T and any term of T with variables $\{x_j : j \in B\}$, $|S_i \cap (\cup_{j \in B \setminus \{i\}} S_j)| < b$.

We will choose them randomly from an approximate $2b$-wise independent space.

Definition 2 *An (n, k, p, δ) space consists of a sequence of n binary random variables $x_1, \ldots, x_n$, such that for any $I \subseteq \{1, \ldots, n\}$ with $|I| \leq k$,*

$$|P[\forall i \in I, x_i = 1] \Leftrightarrow p^{|I|}| \leq \delta.$$

Let y_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq r$, be sampled from an $(nr, 2b, \frac{2s}{r}, \delta)$ space. It can be efficiently sampled using $v = O(\log\log n + b\log\frac{r}{s} + \log\frac{1}{\delta}) = O(\log^2 \frac{dt}{\epsilon})$ random bits [3]. Let $S_i(y) = \{j : y_{ij} = 1\}$. Let T be any subformula of G with at most m' terms and n' variables. Then, by choosing y randomly, the probability that $S_1(y), \ldots, S_n(y)$ are not good for T is at most

$$\begin{aligned}
& n'(\frac{1}{s^b} + r^{2b}\delta) + m'n'\binom{r}{b}c^b((\frac{2s}{r})^{2b} + \delta) \\
\leq\ & \frac{n'}{s^b} + m'n'(\frac{12cs^2}{rb})^b + n'r^{2b}\delta + m'n'(\frac{3cr}{b})^b\delta \\
\leq\ & \frac{\epsilon}{4} + \frac{\epsilon}{4} + \frac{\epsilon}{4} + \frac{\epsilon}{4} \\
\leq\ & \epsilon,
\end{aligned}$$

Consider the generator $g : \{0,1\}^{r+v} \to \{0,1\}^n$, defined as the following:

$$g(w, y) \equiv (\bigoplus_{j \in S_1(y)} w_j, \ldots, \bigoplus_{j \in S_n(y)} w_j).$$

The proof in [7] can be used to show that g is an ϵ-generator for all subfomulas of G with at most m' terms. From Lemma 3, g is a 2ϵ-generator for $G \in c\text{DNF}_d$. From Lemma 2, we can get a $2k\epsilon$-generator for $F \in \text{DNF}_d$. The number of random bits used is

$$k(r+v) = O(\frac{t}{c}\log^4\frac{dt}{\epsilon}) = O(\log\frac{m}{\epsilon}\log^3\frac{d\log m}{\epsilon}).$$

With ϵ replaced by $\frac{\epsilon}{2k}$, we have the following:

Lemma 4. *Given $F \in DNF_d$, we can construct an ϵ-generator for F using $O(\log\frac{m}{\epsilon}\log^3\frac{d\log m}{\epsilon})$ random bits.*

To summarize, given a DNF formula F, we do the following:

- Remove those terms with more than $t = \log\frac{m}{\epsilon}$ variables.
- Determine the parameter d, the maximum number of appearances of any variable.
- Run the hypergraph coloring algorithm to partition F into k $c\text{DNF}_d$ formulas.
- Construct generators $g_1, \ldots, g_k$ and the generator g.
- Compute the average of F under the pseudorandom distribution generated by g.

So we have the following theorem.

Theorem 1 *Given a DNF formula F, we can approximate its volume with error ϵ in deterministic $2^{O(\log\frac{m}{\epsilon}\log^3\frac{d\log m}{\epsilon})}$ time.*

3.2 Construction II, $k = t2^{O(\sqrt{\log(dt)})}$ and $c = O(\sqrt{\log(dt)})$

This is the framework used by Luby and Velicković [6], but we use our hypergraph coloring algorithm instead. Let $l = \lceil\log\frac{8k}{\epsilon}\rceil c2^c$ and $\delta = \frac{\epsilon}{8k2^l}$. Let $h : \{0,1\}^r \to \{0,1\}^n$ be the mapping that generates the $(n, l, 1/2, \delta)$ space. Then

Lemma 5. *[6] h is an $\frac{\epsilon}{k}$-generator for cDNF.*

From Lemma 2, we have an ϵ-generator using $kr = O(k(l+\log\log n+\log 1/\delta)) = O(kl)$ random bits. So we have the following:

Lemma 6. *Given $F \in DNF_d$, we can construct an ϵ-generator for F using $\log\frac{m}{\epsilon}\log\frac{1}{\epsilon}2^{O(\sqrt{\log(d\log\frac{m}{\epsilon})})}$ random bits.*

Similarly, we have the following theorem.

Theorem 2 *Given a DNF formula F, we can approximate its volume with error ϵ in deterministic $2^{\log\frac{m}{\epsilon}\log\frac{1}{\epsilon}2^{O(\sqrt{\log(d\log\frac{m}{\epsilon})})}}$ time.*

4 Dimensions of Posets

In this section, we will constructivize existential bounds, given by Füredi and Kahn [5], on the dimensions of posets.

Definition 3 *[5] Let $(P, <)$ be a poset. Its dimension, denoted as $dim(P)$, is defined to be the minimum number of permutations $L_1, \ldots, L_d$ such that $P = L_1 \cap \cdots \cap L_d$, i.e.,*

$$\forall x, y \in P, \;\; x < y \Leftrightarrow \forall i, x <_{L_i} y. \tag{2}$$

Definition 4 *[5] For $x \in P$, define $U(x) \equiv \{y \in P : y \geq x\}$, $L(x) \equiv \{y \in P : y \leq x\}$, and $C(x) \equiv U(x) \cup L(x)$. Define $u \equiv \max\{U(x) : x \in P\}$, $l \equiv \max\{L(x) : x \in P\}$, and $t \equiv \max\{C(x) : x \in P\}$.*

Lemma 7. *[5] The dimension of $(P, <)$ is equal to the minimum number of permutations $\pi_1, \ldots, \pi_d$, such that*

$$\forall x, y \in P, \;\; y \not\leq x \Rightarrow \exists i, x <_{\pi_i} U(y). \tag{3}$$

In addition, there is a deterministic polynomial time algorithm for converting a set of d permutations satifying condition (3) to a set of d permutations satifying condition (2).

Füredi and Kahn [5] gave a simple upper bound: $dim(P) = O(u \log |P|)$. Their argument can be turned into a deterministic algorithm.

Theorem 3 *Given any poset $(P, <)$, we can find a set of $O(u \log |P|)$ permutations satisfying condition (2) in deterministic polynomial time.*

Proof: We say that a pair $(x, y) \in P^2$ is killed by a permutation π if $x <_\pi U(y)$. We want to pick $d = O(u \log |P|)$ permutations one by one, in d phases. In phase i, find π_i that can kill at least $\frac{1}{u+1}$ fraction of those pairs not killed by $\pi_1, \ldots, \pi_{i-1}$. Such a π_i exists because the expected fraction killed by a random permutation is $\frac{1}{u+1}$. We can find such a π_i by fixing components one by one, in $|P|$ steps, again using the technique of conditional probability. Then after d phases, the number of pairs not killed yet is less than $|P|^2(1 \Leftrightarrow \frac{1}{u+1})^d < 2^{2 \log |P| - d/(u+1)} \leq 1$, for some $d = O(u \log |P|)$. These d permutations satisfy condition (3) and can be converted to d permutations satisfying condition (2). It's easy to see that the whole process can be done in deterministic polynomial time. □

Füredi and Kahn [5] gave another upper bound: $dim(P) = O(t \log^2 t)$. Again, their argument can be turned into a deterministic algorithm. Assume without loss of generality that $t^{\log t} \leq |P|$ (otherwise, we can just use the previous theorem).

First, a simple lemma.

Lemma 8. *[5] Given a hypergraph $H(V, E)$ with $E \subseteq V^{\leq a}$ and degree at most b, there exists a coloring with $(a \Leftrightarrow 1)b + 1$ colors such that no color appears more than once in an edge.*

Such a coloring can easily be found using a greedy algorithm.

Theorem 4 *Given any poset $(P,<)$, we can find a set of $O(t\log^2 t)$ permutations satisfying condition (2) in deterministic polynomial time.*

Proof: Consider the hypergraph $H(V,E)$ where $V=P$ and $E=\{U(y): y\in P\}\subseteq V^{\leq t}$, with degree at most t. Using our hypergraph coloring algorithm, we can color nodes using $k=O(t/c)$ colors such that no color appears more than $c=\Theta(\log t)$ times in an edge. H is now partitioned into k hypergraphs $H_1(V_1,E_1),\ldots,H_k(V_k,E_k)$ in the obvious way.

We will have k groups, $G_1,\ldots,G_k$, of permutations, with permutations in the group G_i designed to kill those pairs $(x,y)\in P^2$ with $x\in V_i$. Permutations in G_i place V_i ahead of $V\setminus V_i$, and use an arbitrary permutation T_i for $V\setminus V_i$. It remains to guarantee that for each $(x,y)\in P^2$ with $x\in V_i$ and $y\neq x$, there exists a permutation $\pi\in G_i$ such that $x<_\pi U(y)\cap V_i$. Notice that we have reduced the original problem to k subproblems.

For H_i, use Lemma 8 to color nodes in V_i with $r=O(ct)=O(t\log t)$ colors such that all colors in an edge are distinct. Let V_{ij}, $1\leq j\leq r$, denote those nodes in V_i with color j. Notice that the order among V_{ij} does not matter, and we fix an arbitrary permutation $R_{i,j}$ on V_{ij}, together with its converse $R^c_{i,j}$. It remains to find a set of permutations on r colors such that for any set S of $c\Leftrightarrow 1$ colors and any color $\alpha\notin S$, some permutation puts α ahead of S. A random collection of $s=O(c^2\log r)$ permutations will fail with probability at most

$$r\binom{r\Leftrightarrow 1}{c\Leftrightarrow 1}(1\Leftrightarrow\frac{1}{c})^s<1.$$

Using a similar idea to that in Theorem 3, a good set of permutations $\gamma_1,\ldots,\gamma_s$ can be found one by one, each in deterministic $r^{O(c)}=|P|^{O(1)}$ time. Then for each i and l with $1\leq i\leq k$ and $1\leq l\leq s$, we define two permutations:

$$\pi_{i,l}=(R_{i,\gamma_l(1)},R_{i,\gamma_l(2)},\ldots,R_{i,\gamma_l(r)},T_i),\text{ and}$$
$$\pi'_{i,l}=(R^c_{i,\gamma_l(1)},R^c_{i,\gamma_l(2)},\ldots,R^c_{i,\gamma_l(r)},T_i).$$

So the total number of permutations is $2sk=O(t\log^2 t)$, and they can be found in deterministic polynomial time. □

Note that a more careful analysis shows that actually we can find a set of $O(t\log u\log l)$ of permutations satisfying condition (2).

5 Acknowledgements

We would like to thank David Mix Barrington for some helpful comments.

References

1. N. Alon, A parallel algorithmic version of the local lemma, *Random Structures and Algorithms*, 2(4), pages 367-378, 1991

2. J. Beck, An algorithmic approach to the Lovász local lemma, *Random Structures and Algorithms*, 2(4), pages 343-365, 1991
3. G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Velicković, Approximations of general independent distributions. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 10-16, 1992.
4. P. Erdös and L. Lovász, problems and results on 3-chromatic hypergraphs and some related questions, in A. Hajnal et. al. Eds, *Infinite and Finite Sets*, North Holland, 1975, pages 609-628.
5. Z. Füredi and J. Kahn, On the dimensions of ordered sets of bounded degree, *Order*, 3, pages 15-20, 1986.
6. M. Luby and B. Velicković, On deterministic approximate counting of DNF, In *Proceedings of the 23nd Annual ACM Symposium on Theory of Computing*, pages 430-438, 1991.
7. M. Luby, B. Velicković, and A. Wigderson, Deterministic approximate counting of depth-2 circuits, In *Proceedings of the Second Israeli Symposium on Theory of Computing and Systems*, 1993.
8. N. Nisan, Pseudo-random bits for constant depth circuits, *Combinatorica*, 11(1), pages 63-70, 1991
9. N. Nisan and A. Wigderson, Hardness vs. randomness, In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 2-11, 1988.
10. P. Raghavan, Probabilistic construction of deterministic algorithm: Approximating packing integer programs, *Journal of Computer and System Sciences*, 38, pages 683-707, 1994.
11. A. Srinivasan, An extension of the Lovász Local Lemma, and its applications to integer programming, In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 6-15, 1996.
12. L. G. Valiant, The complexity of enumeration and reliability problems, *SIAM Journal on Computing*, 8, pages 410-421, 1979.

On the Derandomization of Space-Bounded Computations

Roy Armoni

Institute of Computer Science
The Hebrew University of Jerusalem
aroy@cs.huji.ac.il
http://www.cs.huji.ac.il/~aroy

Abstract. We construct a pseudo-random generator for space bounded computations using the extractor of Zuckerman [7]. For machines that use S space and $R \leq 2^{S^{1-\epsilon}}$ random bits for $\epsilon > 0$, the generator uses a seed of length $O((S \log R)/\log S)$ which is shorter than the seed of both the generator of Nisan [4] and the generator of Nisan and Zuckerman [5]. We then use this generator to derandomize these machines in space $O(S\sqrt{(\log R)/\log S})$ which is better than the derandomization of [6].

1 Introduction

One of the important resources that complexity theory tries to measure besides time and space is the number of random bits needed to solve a given problem using a certain probabilistic model. Upper bounds are usually gained by introducing a deterministic algorithm G that gets a short random input s and outputs a long string of length $R \gg |s|$ that fools the model. By fooling a probabilistic computational model we mean that its behavior on a truly random input of length R is statistically very close to its behavior on the output of G when s is truly random. The algorithm G is called pseudo-random generator for that model.

The computational model that we try to fool here is space-bounded computations, i.e., the complexity class $BPSPACE(S)$ which contains all the problems solvable by a Turing machine in space S using random bits with a bounded two sided error. The two best pseudo-random generators known today for space-bounded computations are [4] and [5]. In [4] Nisan shows a generator that fools any space S machine that requires $R \leq 2^{O(S)}$ random bits using a seed of length $O(S \log R)$. However, when $R = S^{O(1)}$ Nisan and Zuckerman [5] introduce a more powerful generator which requires only $O(S)$ random bits for its seed. Our generator combines ideas from the two generators of [5] and [3], which almost interpolates the whole gap between the two results above. For the rest of the paper denote $\epsilon(z) = 4^{\log^* z} \log z$. We show that for $R \leq 2^{O(S)}$ there exists a pseudo-random generator for space-S computations with seed of length $O(\frac{(S+\epsilon(R)) \log R}{\max\{1, \log S - \log\log R\}})$ and statistical error $R^{-\Omega(1)}$. This result is better than both earlier results for R such that $S^{\omega(1)} < R < 2^{S^{1-\epsilon}}$ for $\epsilon > 0$. For $R = S^k$,

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 47-59, 1998.

the generator of Nisan and Zuckerman uses a seed of length $2^{\Omega(k)}S$ while our generator G uses a seed of length $O(kS)$, and for $R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$, the length of the seed is only $O(\frac{S \log R}{\log S})$, while the generator of Nisan needs a seed of length $\Theta(S \log R)$. Note also, that for $S^{\omega(1)} < R < 2^{o(\log^2 S)}$, our generator gives an estimation to the accepting probability of a $BPSPACE(S)$ machine in less space than [6], since they use $\Theta(S\sqrt{\log R})$ space which is larger than our seed.

We would like to remark here that ideally, the "error term" $e(R)$ should simply be $\log R$, but unfortunately this $e(R)$ term arises in the known constructions of extractors, and no better one is known yet.

The best derandomization known so far for $BPSPACE(S)$ is the one of Saks and Zhou [6]. They proved that any problem that is randomly solvable in space S using $R \leq 2^{O(S)}$ random bits, can be solved deterministically in space $O(S\sqrt{\log R})$ as well. In this paper we show that this task can be done in space $O(\frac{(S+e(R))\sqrt{\log R}}{\sqrt{\max\{1, \log S - \log\log R\}}})$. For $R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$, the space required according to our result is only $O(S\sqrt{\frac{\log R}{\log S}})$ which is less by a factor of $\sqrt{\log S}$ than the space used by Saks and Zhou [6].

2 Preliminaries

2.1 Definitions and Notations

Let $x \in \mathbb{R}^l$ be a vector, then its ith entry is denoted by $x[i]$. The L_1-norm of x is $\|x\| = \sum_{i=1}^{l} |x[i]|$. Let M be a $l \times l$ matrix over $\mathbb{R}$, then its L_1-norm is $\|M\| = \sup\{xM : x \in \mathbb{R}^l, \|x\| = 1\}$. The statistical distance between two distributions D and D' on the space $\{0,1\}^l$ is $\|D - D'\|$.

Let A_1 and A_2 be two events in a probability space. We denote by $\Pr[A_1 : A_2]$ the probability of A_1 conditioned upon A_2. Let $f : W \to R$ be a function, and let X be a random variable distributed on W, then $\mathbf{E}_{x \in X}[f(x)]$ denotes the expectation of $f(X)$. We denote by D_f the distribution over R of the random variable $f(Y)$ where Y is a random variable uniformly distributed over W. If $f : W \to R$ and $g : U \to W$, then $f \circ g$ is the composition of f and g, thus, $D_{f \circ g}$ is the distribution over R of the random variable $f(g(Y))$ where Y is a random variable uniformly distributed over U.

2.2 Extractors

Definition 1. *[5] A distribution D on $\{0,1\}^l$ is called a δ-source if for all $x \in \{0,1\}^l$, $D(x) \leq 2^{-\delta l}$.*

Definition 2. *[7, Definition 1.4] A function $E : \{0,1\}^k \times \{0,1\}^t \to \{0,1\}^m$ is called a $(k, \delta, t, m, \epsilon)$-extractor if for every δ-source D on $\{0,1\}^k$, if X is a random variable distributed according to D and Y uniformly distributed on $\{0,1\}^t$, the distribution of $E(X,Y) \circ Y$ is within statistical distance of ϵ from the uniform distribution on $\{0,1\}^{m+t}$.*

Lemma 1. *There is a constant $c_1 > 0$ such that for every integer k and every $\epsilon > 2^{-\frac{k}{4^{\log^* k}}}$ there exists a $(k, \frac{1}{2}, c_1 \log \frac{k}{\epsilon}, \frac{k}{4}, \epsilon)$- extractor which runs in NC.*

For the proof, refer to [7, Theorem 3.13,Remark 3.14] and substitute $n = k$, $\delta = \frac{1}{2}$ and $\alpha = \frac{1}{4}$. Note that although the original proof uses a different definition for statistical distance between distributions, which differs from ours by a factor of 2, the Lemma still holds.

2.3 Deterministic Sampling

Definition 3. *An* (ϵ, γ) sampler *(oblivious sampler)* $A : \{0,1\}^m \times [k] \to \{0,1\}^l$ *is a deterministic algorithm such that for every function* $f : \{0,1\}^l \to [0,1]$,

$$\Pr_{y \in \{0,1\}^m}[|\mathbf{E}_{i \in [k]}[f(A(y,i))] - \mathbf{E}_{x \in \{0,1\}^l}[f(x)]| > \epsilon] \leq \gamma$$

An efficient construction of a (non-oblivious) sampler is presented in [2]. For the proof of the following Lemma, refer to [7, Theorem 5.5] and substitute $d = k$, $\alpha = \frac{1}{2}$ and $\gamma = 2^{-l}$.

Lemma 2. *For every integer l, for every constant $b < 1$ and for every $\epsilon \geq 2^{-l^b}$ there is an $(\epsilon, 2^{-l})$ oblivious sampler $A : \{0,1\}^{3l} \times [k] \to \{0,1\}^l$, running in NC, where $k = (\frac{l}{\epsilon})^{O(1)}$.*

2.4 Space Bounded Computations and Branching Programs

In this subsection we show how to simulate a space bounded computation by branching programs (BPs) in order to conclude that a pseudo-random generator for BPs fools space bounded machines as well. For that purpose we need the following definition of oblivious read-once BP.

Definition 4. *A (w, n, r)-BP (branching program) is a graph of $n+1$ layers, indexed by $0, \ldots, n$, with w vertices at each layer (length n and width w). For each vertex v at layer $i < n$ there are 2^r outgoing multi-edges to vertices at layer $i+1$ where each of these edges is labeled by a distinct string from $\{0,1\}^r$. Upon receiving an input string from $\{0,1\}^{rn}$ which can be viewed as n blocks of length r, denoted $R_1, \ldots, R_n$, all edges are deleted from the BP, except for the edges between layer $i-1$ and i labeled by R_i, for every $i \leq n$. Let s be one of the vertices at layer 0 - we call s the initial vertex. The value computed by the BP is the index (in $[w]$) of the last vertex in the path that was started from s. Thus, a (w, n, r)-BP can be viewed as a function from $\{0,1\}^{rn}$ to $[w]$.*

Now, we define a pseudo-random generator for BPs.

Definition 5. *A function $G : \{0,1\}^l \to \{0,1\}^{rn}$ is called a γ-pseudo-random generator for (w, n, r)-BPs if for every (w, n, r)-BP P, $\|D_P - D_{P \circ G}\| \leq \gamma$.*

Definition 6. *A function $G : \{0,1\}^l \to \{0,1\}^R$ is called a γ-pseudo-random generator for space-S machines that use R random bits, if for every machine M that runs in space S, uses R random bits and outputs $\{accept, reject\}$, then $\|D_M - D_{M \circ G}\| \leq \gamma$.*

Lemma 3. *Let $S, R > 0$ be integers and let $\alpha > 0$. Then, if $G : \{0,1\}^l \to \{0,1\}^R$ is an α-pseudo-random generator for $(2^S, R, 1)$-BPs, then G is an α-pseudo-random generator for space-S machines that use R random bits.*

Proof: Let M be a space-S machine that uses R random bits. Following [3], we simulate M by a $(2^S, R, 1)$-BP P. At each layer of P, the 2^S different vertices will correspond to the different configurations of M. An edge from vertex v at layer i, labeled by $x \in \{0,1\}$ will end up at vertex u at layer $i+1$ iff the input x on the random tape of M takes it from configuration v to u. Let s be the initial configuration of M. It follows that P computes the ending configuration of P. Now, since the output of M is a mapping from configurations to $\{accept, reject\}$, we get that for every $G : \{0,1\}^l \to \{0,1\}^R$, $\|D_M - D_{M \circ G}\| \leq \|D_P - D_{P \circ G}\|$. We conclude that if G is α-pseudo-random for $(2^S, R, 1)$-BPs, then G is α-pseudo-random for space-S machines that use at most R random bits. □

Remark 1. If $G : \{0,1\}^l \to \{0,1\}^{rn}$ is γ-pseudo-random generator for (w, n, r)-BPs, and if for some $r' < r$, $G' : \{0,1\}^l \to \{0,1\}^{r'n}$ is defined to output the first r' bits of every output block of G, then G' is γ-pseudo-random generator for (w, n, r')-BPs.

3 Pseudo-randomness

In this section we show a pseudo-random generator that fools any (w, n, r)-BP P. Our building block will be the extractor from sub-section 2.2.

3.1 Pseudo-randomness for Branching Programs

Definition 7. *Let c_1 be the constant of Lemma 1. Given n, w, r and γ, fix the following parameters: Let $\epsilon \stackrel{\text{def}}{=} \frac{\gamma}{2n}$, $k \stackrel{\text{def}}{=} \max\{4r, 2\log w + e(\frac{1}{\epsilon})\}$ (we remind the reader that $e(z) = 4^{\log^* z} \log z$) and $t \stackrel{\text{def}}{=} c_1 \log \frac{k}{\epsilon}$. Let E be the $(k, \frac{1}{2}, t, r, \epsilon)$-extractor of Lemma 1. Define a generator $\hat{G} : \{0,1\}^{k+nt} \to \{0,1\}^{nr}$ as follows:*

$$\forall x \in \{0,1\}^k, y_1, \ldots, y_n \in \{0,1\}^t, \quad \hat{G}(x, y_1, \ldots, y_n) \stackrel{\text{def}}{=} E(x, y_1), \ldots, E(x, y_n)$$

For every $y_1, \ldots, y_n \in \{0,1\}^t$, we denote $\hat{G}^{(y_1, \ldots, y_n)} : \{0,1\}^k \to \{0,1\}^{nr}$ as $\hat{G}^{(y_1, \ldots, y_n)}(x) = \hat{G}(x, y_1, \ldots, y_n)$.

Lemma 4. *For every n, w, r and γ, for every (w, n, r)-BP P and for every random variable Y uniformly distributed on $\{0,1\}^{tn}$, $\mathbf{E}_{y \in Y}[\|D_P - D_{P \circ \hat{G}^{(y)}}\|] \leq \gamma$.*

A similar version of this Lemma has already been proven by [5, Lemma 2] for space-bounded machine, and we state our Lemma for BPs. Another difference is that their Lemma is weaker, since they only proved that

$$\|\mathbf{E}_{y\in Y}[D_P - D_{P\diamond\hat{G}^{(y)}}]\| = \|D_P - D_{P\diamond\hat{G}}\| \leq \gamma$$

Proof: Let P be a (w,n,r)-BP. We denote by $S_i(u,v) \subseteq \{0,1\}^r$ the set of labels of edges going from vertex u at layer $i-1$ to vertex v at layer i of P. We define the following $w \times w$ stochastic matrices. M_i is the matrix with $\frac{|S_i(u,v)|}{2^r}$ at its (u,v) coordinate and for every $x \in \{0,1\}^k$ and $y \in \{0,1\}^t$ we define the matrix $\tilde{M}_i(x,y)$ with 1 at its (u,v) coordinate if $E(x,y) \in S_i(u,v)$ and 0 otherwise. Let u_0 be the initial state of P, then let e be a unit vector of length w with 1 at its u_0 coordinate and 0 elsewhere.

Let X be a random variable uniformly distributed on $\{0,1\}^k$. Notice that $D_P = e\prod_{i=1}^n M_i$ and that for every $y_1,\ldots,y_n \in \{0,1\}^t$,

$$D_{P\diamond\hat{G}^{(y_1,\ldots,y_n)}} = \mathbf{E}_{x\in X}[e\prod_{i=1}^{n}\tilde{M}_i(x,y_i)]$$

Let $Y_1,\ldots,Y_n$ be random variables uniformly distributed on $\{0,1\}^t$. We will show by induction on j that

$$\mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e((\prod_{i=1}^{j}M_i) - \mathbf{E}_{x\in X}[\prod_{i=1}^{j}\tilde{M}_i(x,y_i)])\|] \leq \frac{j\gamma}{n}$$

For $j=0$ we trivially get equality. For $j>0$, assume correctness for $j-1$ and prove for j.

$$\begin{aligned}
&\mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e((\prod_{i=1}^{j}M_i) - \mathbf{E}_{x\in X}[\prod_{i=1}^{j}\tilde{M}_i(x,y_i)])\|] \leq \\
&\leq \mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e((\prod_{i=1}^{j-1}M_i) - \mathbf{E}_{x\in X}[\prod_{i=1}^{j-1}\tilde{M}_i(x,y_i)])M_j\|] + \\
&\quad + \mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e\mathbf{E}_{x\in X}[(\prod_{i=1}^{j-1}\tilde{M}_i(x,y_i))(M_j - \tilde{M}_j(x,y_j))]\|] \\
&\leq \frac{(j-1)\gamma}{n} + \mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|\mathbf{E}_{x\in X}[v(x,y_1,\ldots,y_{j-1})(M_j - \tilde{M}_j(x,y_j))]\|]
\end{aligned}$$

where in the last inequality, the first summand is bounded by the induction hypothesis and $\|M_j\| = 1$ as it is stochastic, and in the second summand, $v(x,y_1,\ldots,y_{j-1}) = e\prod_{i=1}^{j-1}\tilde{M}_i(x,y_i)$. Note that $v(x,y_1,\ldots,y_{j-1})$ is a distribution vector over $[w]$. To conclude the proof, it is enough to show that for every $y_1,\ldots,y_{j-1} \in \{0,1\}^t$,

$$\mathbf{E}_{y_j\in Y_j}[\|\mathbf{E}_{x\in X}[v(x,y_1,\ldots,y_{j-1})(M_j - \tilde{M}_j(x,y_j))]\|] \leq \frac{\gamma}{n}$$

For the rest of the proof we fix the values of $y_1, \ldots, y_{j-1}$.

Let $\mathcal{V}$ be a random variable distributed on $[w]$ according to the distribution $\mathbf{E}_{x \in X}[v(x, y_1, \ldots, y_{j-1})]$. Let $L = \{v \in [w] : \Pr[\mathcal{V} = v] > \frac{\epsilon}{w}\}$.

For $v \in L$, $\Pr[X = x : \mathcal{V} = v] \leq \frac{\Pr[X=x]}{\Pr[\mathcal{V}=v]} \leq \frac{w}{\epsilon} \cdot \Pr[X = x] \leq 2^{\frac{k}{2}} \cdot \Pr[X = x]$. We get that the distribution of X conditioned upon $\mathcal{V} = v$ is a $\frac{1}{2}$-source. Since E is an extractor, we get that for a random variable Z which is uniformly distributed on $\{0,1\}^r$,

$$\sum_{y_j \in \{0,1\}^t} \sum_{z \in \{0,1\}^r} |\Pr[(Z = z) \wedge (Y_j = y_j)] - \Pr[(E(X, Y_j) = z) \wedge (Y_j = y_j) : \mathcal{V} = v]| \leq \epsilon$$

thus,

$$\begin{aligned}
&\mathbf{E}_{y_j \in Y_j}[\|\mathbf{E}_{x \in X}[v(x, y_1, \ldots, y_{j-1})(M_j - \tilde{M}_j(x, y_j))]\|] = \\
&= \mathbf{E}_{y_j \in Y_j}[\|\mathbf{E}_{x \in X}[\sum_{v \in [w]} \Pr[\mathcal{V} = v](M_j[v, \cdot] - \tilde{M}_j(x, y_j)[v, \cdot])]\|] \\
&\leq \sum_{v \in L} \Pr[\mathcal{V} = v] \\
&\quad \mathbf{E}_{y_j \in Y_j}[\sum_{v' \in [w]} |\Pr[Z \in S_j(v, v')] - \Pr[E(X, y_j) \in S_j(v, v') : \mathcal{V} = v]|] \\
&\quad + \sum_{v \notin L} \Pr[\mathcal{V} = v] \mathbf{E}_{y_j \in Y_j}[\|\mathbf{E}_{x \in X}[(M_j[v, \cdot] - \tilde{M}_j(x, y_j)[v, \cdot])]\|] \\
&\leq \sum_{v \in L} \Pr[\mathcal{V} = v] \cdot \epsilon + \sum_{v \notin L} \Pr[\mathcal{V} = v] \cdot 1 \leq \epsilon + \sum_{v \notin L} \frac{\epsilon}{w} \leq 2\epsilon \leq \frac{\gamma}{n}
\end{aligned}$$

□

Theorem 1. *There exist constants c_2, c_3 such that for every n, w, r and γ, if $k = c_2(r + \log w + e(\frac{n}{\gamma}))$ and $t = c_3 \log \frac{nr \log w}{\gamma}$ then $\hat{G} : \{0,1\}^{k+nt} \to \{0,1\}^{nr}$ is a γ-pseudo-random generator for (w, n, r)-BPs that runs in NC.*

Proof: Let $c_2 = 4$ and $c_3 = 2c_1$. Let Y be a random variable uniformly distributed on $\{0,1\}^{nt}$. By Lemma 4, for every (w, n, r)-BP P

$$\|D_P - D_{P \circ \hat{G}}\| = \|D_P - \mathbf{E}_{y \in Y}[D_{P \circ \hat{G}(y)}]\| \leq \mathbf{E}_{y \in Y}[\|D_P - D_{P \circ \hat{G}(y)}\|] \leq \gamma$$

Since each of the output bits of $\hat{G}$ is simply computed by applying an extractor to some predefined subset of the input bits, and since the extractor runs in NC, then each of the output bits of $\hat{G}$ is computed in NC as well. □

3.2 Composing Generators for Branching Programs

In this subsection, we will see that the generator $\hat{G}$ that was defined in subsection 3.1 can be composed on itself. To this end, we need the following proposition:

Proposition 1. *Let P be a (w,n,r)-BP and let $\hat{G}:\{0,1\}^{k+nt}\rightarrow\{0,1\}^{nr}$ be the pseudo-random generator for (w,n,r)-BPs of Theorem 1. Then, for every $x\in\{0,1\}^k$,*

$$P^{(x)}(y_1,\ldots,y_n)\stackrel{\text{def}}{=}P(\hat{G}(x,y_1,\ldots,y_n))$$

is a (w,n,t)-BP.

Proof: Note that $P^{(x)}$ has the same vertex set as P has, but its edge set is smaller. Every edge from layer $i-1$ to layer i of P with label from $E(x,y_i)$ for some value of $y_i\in\{0,1\}^t$ is labeled in $P^{(x)}$ by y_i while all other edges of P do not appear in $P^{(x)}$. Hence, $P^{(x)}$ is a (w,n,t)-BP. □

In order to be able to compose the generator, we would need to view the BP $P^{(x)}$ as a (w,n',r)-BP for some n'. In the following definition we formally show how we do so.

Definition 8. *Let Q be a (w,n,t)-BP. We define the operation* collapse-c*, the result of which is a $(w,\lceil\frac{n}{c}\rceil,tc)$-BP, $\tilde{Q}$, as follows. Denote $n'\stackrel{\text{def}}{=}\lceil\frac{n}{c}\rceil$, then $\tilde{Q}$ has $n'+1$ layers indexed by $0,\ldots,n'$, each of which contains w vertices.*
For every $i<n'$, for every $u,v\in[w]$, the edge between u at layer i and v at layer $i+1$ exists in $\tilde{Q}$ if and only if there exists a path in Q between vertex u at layer ic and vertex v at layer $(i+1)c$ (in the case that $(i+1)c>n$, we look for a path ending at vertex v at layer n, which is of course of length shorter than c). The label of the edge (u,v) in $\tilde{Q}$ is the concatenation of the labels on the edges in the corresponding path in Q. In the case that the length of the paths to the last layer in Q are of length c' shorter than c, the edges in $\tilde{Q}$ corresponding to these paths have labels of length only $c't$. In this case we lengthen these labels by concatenating to each of them, all the possible suffixes from $\{0,1\}^{(c-c')t}$ which results in replacing each edge by $2^{(c-c')t}$ parallel edges.

Assuming that $t<r$, we would like to perform the operation collapse-$\lceil\frac{r}{t}\rceil$ on $P^{(x)}$ and get a new BP, $\tilde{P}^{(x)}$, with labels of length at least r. This fact allows us to feed the output of $\hat{G}$ instead of a real random string to $\tilde{P}^{(x)}$. For the rest of this subsection, we will properly define the composition of $\hat{G}$ and prove that this composition really gives us a pseudo-random generator for BPs.

Definition 9. *Let n,w,r and α be given. Let $\gamma\stackrel{\text{def}}{=}\frac{\alpha}{\log n}$ and let $k=c_2(r+\log w+e(\frac{n}{\gamma}))$. Let $t=c_3\log\frac{2nk\log w}{\gamma}$, $r'=t\max\{2,\lceil\frac{2\log w+e(\frac{n}{\gamma})}{t}\rceil\}$ and $k'=2c_2r'$.*

Let $h\stackrel{\text{def}}{=}\lceil\frac{\log n}{\log\frac{r'}{t}}\rceil$, let n_0 be n and for $i>0$, let $n_i\stackrel{\text{def}}{=}\frac{n_{i-1}}{\frac{r'}{t}}$. For $i>0$, we define generators $G_i:\{0,1\}^k\times\{0,1\}^{(i-1)\cdot k'}\times\{0,1\}^{n_{i-1}\cdot t}\rightarrow\{0,1\}^{n\cdot r}$ recursively as follows:

$$G_1(x_1,y_1,\ldots,y_n)\stackrel{\text{def}}{=}\hat{G}(x_1,y_1,\ldots,y_n)$$

and for $i>1$

$$G_i(x_1,\ldots,x_i,y_1,\ldots,y_{n_{i-1}})\stackrel{\text{def}}{=}G_{i-1}(x_1,\ldots,x_{i-1},\hat{G}(x_i,y_1,\ldots,y_{n_{i-1}}))$$

where we think of each of the n_{i-1} output blocks of $\hat{G}$ of length r' as partitioned into $\frac{r'}{t}$ blocks of length t, which gives a total of n_{i-2} blocks from $\{0,1\}^t$ each. Now, for every $x_1 \in \{0,1\}^k$, every $x_2, \ldots, x_h \in \{0,1\}^{k'}$ and for every $y \in \{0,1\}^t$ we define

$$G(x_1, \ldots, x_h, y) \stackrel{\text{def}}{=} G_h(x_1, \ldots, x_h, y)$$

Theorem 2. *G is an α-pseudo-random generator for (w,n,r)-BPs that expands a seed of length $l = O(r + \log n \frac{\log rw + e(\frac{n}{\alpha})}{\max\{1, \log\log w - \log\log \frac{nr}{\alpha}\}})$ and it runs in space $O(l)$.*

Proof: The length of the seed that G uses is $k + k'(h-1) + t = O(r + k'h)$.

$$k'h = O(k' \frac{\log n}{\log \frac{r'}{t}}) = O(\frac{r' \log n}{\max\{1, \log\log w - \log t\}})$$
$$= O(\log n \frac{\log rw + 4^{\log^* \frac{n}{\alpha}} \log \frac{n}{\alpha}}{\max\{1, \log\log w - \log\log \frac{nr}{\alpha}\}})$$

thus, the total length of the seed is $O(r + \log n \frac{\log rw + e(\frac{n}{\alpha})}{\max\{1, \log\log w - \log\log \frac{nr}{\alpha}\}})$. Since each of the output bits of $\hat{G}$ is computed in NC, and G is a composition of $\hat{G}$ at most h times, then G does not use more than $k'h = O(l)$ space.

Let P be a (w,n,r)-BP with n and r that satisfy the conditions in Definition 9. We prove by induction on i that $\|D_P - D_{P \circ G_i}\| \leq i\gamma$.
For $i = 1$ the claim follows immediately from the fact that by the choice of k and t and by Theorem 1, $\hat{G}$ is a γ-pseudo-random generator for (w,n,r)-BPs. Assume correctness for $i-1$ and prove for $i > 1$.

$$\|D_P - D_{P \circ G_i}\| \leq \|D_P - D_{P \circ G_{i-1}}\| + \|D_{P \circ G_{i-1}} - D_{P \circ G_i}\|$$

By the induction hypothesis, the first summand is bounded by $(i-1)\gamma$ so we are left to bound the second. Let $x_1 \in \{0,1\}^k$, $x_2, \ldots, x_{i-1} \in \{0,1\}^{k'}$ and $y_1, \ldots, y_{n_{i-1}} \in \{0,1\}^t$ we define

$$P^{(x_1, \ldots, x_{i-1})}(y_1, \ldots, y_{n_{i-1}}) \stackrel{\text{def}}{=} P(G_{i-1}(x_1, \ldots, x_{i-1}, y_1, \ldots, y_{n_{i-1}}))$$

Let X_1 be a random variable uniformly distributed over $\{0,1\}^k$ and let $X_2, \ldots, X_{i-1}$ be random variables uniformly distributed over $\{0,1\}^{k'}$. Then,

$$D_{P \circ G_{i-1}} = \mathbf{E}_{x_1 \in X_1}[\ldots \mathbf{E}_{x_{i-1} \in X_{i-1}}[D_{P^{(x_1, \ldots, x_{i-1})}}] \ldots]$$
$$D_{P \circ G_i} = \mathbf{E}_{x_1 \in X_1}[\ldots \mathbf{E}_{x_{i-1} \in X_{i-1}}[D_{P^{(x_1, \ldots, x_{i-1}) \circ \hat{G}}}] \ldots]$$

we get that

$$\|D_{P \circ G_{i-1}} - D_{P \circ G_i}\| \leq$$
$$\leq \mathbf{E}_{x_1 \in X_1}[\ldots \mathbf{E}_{x_{i-1} \in X_{i-1}}[\|D_{P^{(x_1, \ldots, x_{i-1})}} - D_{P^{(x_1, \ldots, x_{i-1}) \circ \hat{G}}}\|] \ldots] \leq \gamma$$

where the last inequality is correct since $P^{(x_1, \ldots, x_{i-1})}$ is a (w, n_{i-1}, t)-BP and by performing $\frac{r'}{t}$-collapse we get a (w, n_i, r')-BP and by the choice of k' and t and by Theorem 1, $\hat{G}$ is a γ-pseudo-random generator for (w, n_i, r')-BPs. We conclude that $\|D_P - D_{P \circ G_h}\| \leq h\gamma \leq \alpha$ □

Corollary 1. *There is an α-pseudo-random generator $G : \{0,1\}^l \to \{0,1\}^R$ for space-S machines that use R random bits, where G uses a seed of length $l = O(\frac{(S+e(\frac{R}{\alpha}))\log R}{\max\{1,\log S - \log\log\frac{R}{\alpha}\}})$.*

Proof: Use Theorem 2 and Lemma 3. □

4 Derandomization of Space Bounded Computations

In this section we show how to derandomize space bounded computations using our generator. We will use the method of [6], but instead of employing the generator of [4], we are taking the generator we constructed in the previous section. Unfortunately, this is not a straight forward task, since a close investigation of [6] reveals the fact that they use a very unique property of Nisan's generator that our generator doesn't have. This property allows one to fix most of the bits in the seed of the generator and yet, by exhausting all possibilities of the remaining bits, to get a good estimation on the probability of moving from one configuration to another in the machine (or in the branching program). Our first task here is to show that every generator can be augmented with this property and only then, we proceed to use our generator in [6]. To this end, we will use samplers, such as the oblivious sampler of [7].

A second use for the sampler is to decrease the error. A key idea in this paper is to have the error of the generator be 1 over polynomial in the length of the BP. However, when we carry out the perturbation and truncation technique of [6], we introduce an error which is factored by a polynomial in the width of the BP. This factor cannot be tolerated, so we need to somehow decrease the error. Again, we use the (very same) sampler in a similar way to [1].

Lemma 5. *Let $b < 1$. Let $G : \{0,1\}^l \to (\{0,1\}^r)^n$ be an α-pseudo-random generator for (w,n,r)-BPs that runs in space-s. For every $\beta > w2^{-l^b}$, there exists an algorithm $\mathcal{G} : \{0,1\}^{3l} \times [k] \to (\{0,1\}^r)^n$ where $k = (\frac{wl}{\beta})^{O(1)}$, $\mathcal{G}$ runs in space $s + poly\log l$ and if for every $y \in \{0,1\}^{3l}$ we define the function $\mathcal{G}_y : [k] \to (\{0,1\}^r)^n$ by $\mathcal{G}_y(i) = \mathcal{G}(y,i)$, then for every (w,n,r) BP P,*

$$\Pr_{y\in\{0,1\}^{3l}}[\|D_{P\circ\mathcal{G}_y} - D_{P\circ G}\| > \beta] < w2^{-l}$$

Proof: Let $A : \{0,1\}^{3l} \times [k] \to \{0,1\}^l$ be the $(\frac{\beta}{w}, 2^{-l})$ oblivious sampler of Lemma 2. Define $\mathcal{G} = G \circ A$. Let P be a (w,n,r) BP. For every $j \in [w]$, define the function $f_j : \{0,1\}^l \to \{0,1\}$ as $f_j(x) = 1$ if and only if $P(G(x)) = j$. We get that for every $y \in \{0,1\}^l$

$$\begin{aligned}\|D_{P\circ\mathcal{G}_y} - D_{P\circ G}\| &= \sum_{j\in[w]} |\Pr_{i\in[k]}[P(G(A(y,i))) = j] - \Pr_{x\in\{0,1\}^l}[P(G(x)) = j]| \\ &= \sum_{j\in[w]} |\mathbf{E}_{i\in[k]}[f_j(A(y,i))] - \mathbf{E}_{x\in\{0,1\}^l}[f_j(x)]|\end{aligned}$$

and by the fact that A is a $(\frac{\beta}{w}, 2^{-l})$ sampler we conclude that

$$\begin{aligned}
&\Pr_{y\in\{0,1\}^{3l}}[\|D_{P\circ\mathcal{G}_y} - D_{P\circ G}\| > \beta] \leq \\
&\leq \Pr_{y\in\{0,1\}^{3l}}[\sum_{j\in[w]} |\mathbf{E}_{i\in[k]}[f_j(A(y,i))] - \mathbf{E}_{x\in\{0,1\}^l}[f_j(x)]| > \beta] \\
&\leq \sum_{j\in[w]} \Pr_{y\in\{0,1\}^{3l}}[|\mathbf{E}_{i\in[k]}[f_j(A(y,i))] - \mathbf{E}_{x\in\{0,1\}^l}[f_j(x)]| > \frac{\beta}{w}] \leq w2^{-l}
\end{aligned}$$

□

Corollary 2. *Let $b < 1$. There exists an algorithm $\mathcal{G}$ such that for every w, n, r, α and $\beta > 2^{-l^b}$ where $l = O(r + \log n \frac{\log rw + e(\frac{n}{\alpha})}{\max\{1, \log\log w - \log\log\frac{nr}{\alpha}\}})$, $\mathcal{G}$ takes an input y of length $|y| = l$ and an input i of length $O(\log\frac{wl}{\beta})$, $\mathcal{G}$ runs in space $O(l)$ and outputs n blocks of length r such that with probability at least $1 - w2^{-\frac{l}{3}}$ over $y \in \{0,1\}^l$, $\mathcal{G}_y(\cdot) = \mathcal{G}(y,\cdot)$ is an $\alpha+\beta$ pseudo-random generator for (w,n,r) BPs.*

Proof: Combine Theorem 2 and Lemma 5 using for every $y \in \{0,1\}^l$

$$\|D_{P\circ\mathcal{G}_y} - D_P\| \leq \|D_{P\circ\mathcal{G}_y} - D_{P\circ G}\| + \|D_{P\circ G} - D_P\|$$

□

Now we can start following the proof of [6], while replacing Nisan's generator by ours. For the rest of this paper, we assume that the reader is familiar with the details of [6]. The first step will be to redefine their PRS algorithm, which will now use our generator $\mathcal{G}$. The following Lemma replaces [6, Lemma 4.1].

Lemma 6. *Let $b < 1$. Given as input a $w \times w$ sub-stochastic matrix M, an $\alpha > 0$ and integers t and K, set $r = t - \log\alpha$ and $l = O(t\frac{\log w + e(r)}{\max\{1, \log\log w - \log r\}})$, then if $K < l^b$, then the algorithm $PRS(M,t,r;y)$ runs in space $O(r)$, takes a random string $y \in \{0,1\}^l$ and computes a sub-stochastic matrix of dimension w which approximates $\Lambda^{(t)}(M)$ with accuracy $K - \log\alpha$ and error probability $w2^{-\frac{l}{3}}$.*

Proof: We basically follow the proof of [6, Lemma 4.1], employing corollary 2 and noticing that the PRS algorithm does not use more than $O(r)$ space other than the space needed to compute $\mathcal{G}$. However, although PRS will be part of a recursive algorithm, the $\mathcal{G}$ algorithm does not participate in the recursion, thus, it uses the same $O(l)$ extra space each time it is invoked (even in different levels of the recursion). Moreover, while using corollary 2 with $\beta = 2^{-K}$ and $n = 2^t$, the error probability over the $y \in \{0,1\}^l$ of using $\mathcal{G}$ is $w2^{-\frac{l}{3}}$. □

Now, we let the algorithm MAIN of [6] work normally, only using our new version of the PRS algorithm which uses shorter random input. To fully utilize this extra power, we fix the parameters somewhat differently. Given an $\alpha > 0$ and integers w and t we compute the following parameters: $r = t - \log\alpha$,

$K = 13(r + \log w)$, $D = 7(r + \log w)$, $d = K - D$ as in [6], and we set $t_1 = \lceil \sqrt{t \max\{1, \log\log w - \log r\}} \rceil$, $t_2 = \lceil \frac{t}{t_1} \rceil$ (for simplicity assume in the sequel that $t = t_1 t_2$) and $l = O(t_1 \frac{\log w + e(r)}{\max\{1, \log\log w - \log r\}})$. The algorithm MAIN gets as input a sub-stochastic $w \times w$ matrix M and random inputs $y \in \{0,1\}^l$ and $q_1, \ldots, q_{t_2} \in \{0,1\}^D$ and computes a sequence of matrices, where for every $i \in [t_2]$,

$$M_i = \lfloor \Sigma_{\delta_i}(PRS(M_{i-1}, t_1, r; y)) \rfloor_d$$

$M_0 = M$ and $\delta_i = q_i 2^{-K}$.

Theorem 3. *The algorithm MAIN approximates $\Lambda^{(t)}(M)$ with L_1 distance $2^{2t}(\alpha + 2w2^{-d})$ and error probability $t_2(w2^{-\frac{l}{3}} + 2w^2 2^{-D})$ and it does so in space $O(l + Kt_2)$.*

Proof: For a $w \times w$ matrix N, an algorithm $g : \{0,1\}^l \to (\{0,1\}^r)^{2^{t_1}}$ and a $y \in \{0,1\}^l$, define the operation $S_g(N; y)$ as the $w \times w$ matrix that in its i, j entry contains 1 if we can move from i to j in 2^{t_1} steps in the $(w, 2^{t_1}, r)$ machine associated with N, where the "instructions" are the 2^{t_1} blocks of length r of $g(y)$. Note that $PRS(N, t_1, r; y) = \mathbf{E}_{j \in [k]}[S_{\mathcal{G}_y}(N; j)]$. Define the following sequences of matrices for every $i \in [t_2]$:

$$N_i = \lfloor \Sigma_{\delta_i}(\mathbf{E}_y[S_G(N_{i-1}; y)]) \rfloor_d$$
$$M_i(y) = \lfloor \Sigma_{\delta_i}(\mathbf{E}_{j \in [k]}[S_{\mathcal{G}_y}(M_{i-1}(y); j)]) \rfloor_d$$

where $N_0 = M_0(y) = M$. Now, since G a an α-pseudo-random generator for $(w, 2^{t_1}, r)$-BPs and since the operations $\lfloor \cdot \rfloor_d$ and $\Sigma(\cdot)$ introduce an error of at most $w2^{-d}$ each, we get that

$$\begin{aligned} \|N_i - \Lambda^{(it_1)}(M)\| &\leq \|\lfloor \Sigma_{\delta_i}(\mathbf{E}_y[S_G(N_{i-1}; y)]) \rfloor_d - (N_{i-1})^{2^{t_1}}\| \\ &\quad + \|(N_{i-1})^{2^{t_1}} - (\Lambda^{((i-1)t_1)}(M))^{2^{t_1}}\| \\ &\leq \alpha + 2 \cdot w2^{-d} + 2^{t_1} 2^{2(i-1)t_1}(\alpha + 2 \cdot w2^{-d}) \\ &\leq 2^{2it_1}(\alpha + 2 \cdot w2^{-d}) \end{aligned}$$

We will now prove by induction on i that with high probability over the random choices of y and $\delta_1, \ldots, \delta_i$, the matrices $M_i(y) = N_i$. More precisely, we will prove by induction on i that

$$\Pr_{y \in \{0,1\}^l, q_1, \ldots, q_i \in \{0,1\}^D}[\wedge_{j=1}^{i} M_j(y) = N_j] \geq 1 - i(w2^{-\frac{l}{3}} + 2w^2 2^{-D})$$

For $i = 0$, the statement is trivial.
Denote $M_P(y) = \mathbf{E}_{j \in [k]}[S_{\mathcal{G}_y}(N_{i-1}; j)]$ and $N_P = \mathbf{E}_y[S_G(N_{i-1}; y)]$.

$$\begin{aligned} &\Pr_{y \in \{0,1\}^l, q_i \in \{0,1\}^D}[M_i(y) \neq N_i : M_{i-1}(y) = N_{i-1}] \leq \\ &\leq \Pr_{y \in \{0,1\}^l}[\|M_P(y) - N_P\| > 2^{-K}] \\ &\quad + \Pr_{y \in \{0,1\}^l, q_i \in \{0,1\}^D}[\lfloor \Sigma_{\delta_i}(M_P(y)) \rfloor_d \neq \lfloor \Sigma_{\delta_i}(N_P) \rfloor_d : \|M_P(y) - N_P\| \leq 2^{-K}] \end{aligned}$$

Remembering that $\mathcal{G} = G \circ A$ and using Lemma 5 we bound the first summand by $w2^{-\frac{l}{3}}$. The second summand is bounded in [6, Lemma 5.5] by $2w^2 2^{-D}$, so we get that

$$\begin{aligned}
&\Pr_{y\in\{0,1\}^l, q_1,\ldots,q_i\in\{0,1\}^D}[\wedge_{j=1}^{i} M_j(y) = N_j] = \\
&= \Pr_{y\in\{0,1\}^l, q_1,\ldots,q_i\in\{0,1\}^D}[\wedge_{j=1}^{i-1} M_j(y) = N_j] \cdot \\
&\quad \Pr_{y\in\{0,1\}^l, q_i\in\{0,1\}^D}[M_i(y) = N_i : \wedge_{j=1}^{i-1} M_j(y) = N_j] \\
&\geq (1-(i-1)(w2^{-\frac{l}{3}} + 2w^2 2^{-D}))(1 - w2^{-\frac{l}{3}} - 2w^2 2^{-D}) \\
&\geq 1 - i(w2^{-\frac{l}{3}} + 2w^2 2^{-D})
\end{aligned}$$

Now we can conclude that

$$\begin{aligned}
\Pr_{y\in\{0,1\}^l, q_1,\ldots,q_i\in\{0,1\}^D}[\|A^{(t_1 t_2)}(M) - M_{t_2}\| > 2^{2t_1 t_2}(\alpha + 2w2^{-d})] < \\
< t_2(w2^{-\frac{l}{3}} + 2w^2 2^{-D})
\end{aligned}$$

The space required by algorithm MAIN is $O(l)$ to store the y argument, $O(t_2 D)$ to store $q_1, \ldots, q_{t_2}$, the $O(l)$ space for computing the function $\mathcal{G}$, and then, for each of the t_2 levels of the recursion, an $O(K)$ space is needed (for the PRS algorithm by Lemma 6 and for computing Σ_{δ_i} and $\lfloor\cdot\rfloor_d$). This sums up to $O(l + t_2 K)$ space. □

Corollary 3. *Every language which is decidable randomly in space S using $R \leq 2^S$ random bits with two sided error, can be decided deterministically in space $O(\frac{(S+e(R))\sqrt{\log R}}{\sqrt{\max\{1, \log S - \log\log R\}}})$.*

Proof: Let M be the stochastic matrix associated with the Turing machine that decides the language in space S using R random bits, i.e., the (i,j) entry of M contains the probability of moving from configuration i to j in the Turing machine on a single random bit. Set $w = 2^S$ (M is a $w \times w$ matrix), $t = \lceil \log R \rceil$ and $\alpha = R^{-3}$. We now exhaust all the possible random inputs $y \in \{0,1\}^l$ and $q_1, \ldots, q_{t_2} \in \{0,1\}^D$ to the algorithm MAIN and run it on every input to get an estimation to M^R by taking average of every entry.

By Theorem 3 we approximates the matrix M^R with error smaller than $\frac{1}{10}$ (by the choice of parameters) and in space $O(\frac{(S+e(R))\sqrt{\log R}}{\sqrt{\max\{1, \log S - \log\log R\}}})$. Taking majority vote on the accepting configuration, we decide whether the given input is in the language. □

Acknowledgment

I would like to thank Avi Wigderson for many discussions and helpful comments on this paper. I would also like to thank Amnon Ta-Shma, Oded Goldreich, Noam Nisan and Shiyu Zhou for reading earlier versions of this paper and for their constructive comments.

References

1. Roy Armoni, Amnon Ta-Shma, Avi Wigderson, and Shiyu Zhou. $SL \subseteq L^{4/3}$. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 230–239, 1997.
2. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3(4):319–354, 1993.
3. Russel Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 356–364, Montréal, Québec, Canada, May 1994.
4. Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, June 1992.
5. Noam Nisan and David Zuckerman. More deterministic simulation in logspace. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 235–244, San Diego, CA, May 1993.
6. Michael Saks and Shiyu Zhou. $RSPACE(S) \subset DSPACE(S^{\frac{3}{2}})$. In *36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 344–353, Milwaukee, Wisconsin, October 1995. To appear in JCSS.
7. David Zuckerman. Randomness-optimal sampling, extractors, and constructive leader election. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 286–295, Philadelphia, Pennsylvania, May 1996.

Talagrand's Inequality and Locality in Distributed Computing

Devdatt P. Dubhashi *

Department of Computer Science and Engg.
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi 110016
INDIA
dubhashi@cse.iitd.ernet.in,
WWW home page:http://bhim.cse.iitd.ernet.in/dubhashi

Abstract.

1 Introduction

The aim of this paper is to advocate the use of Talagrand's isoperimetric inequality [10] and an extension of it due to Marton [5, 6] as a tool for the analysis of distributed randomized algorithms that work in the *locality* paradigm. Two features of the inequality are crucially used in the analysis: first, very refined control on the influence of the underlying variables can be exercised to get significantly stronger bounds by exploiting the non–uniform and asymmetric conditions required by the inequality (in contrast to previous methods) and second, the method, using an extension of the basic inequality to dependent variables due to Marton [6] succeeds in spite of lack of full independence amongst the underlying variables. This last feature especially makes it a particularly valuable tool in Computer Science contexts where lack of independence is omnipresent. Our contribution is to highlight the special relevance of the method for Computer Science applications by demonstrating its use in the context of a class of distributed computations in the locality paradigm.

We give a high probability analysis of a distributed algorithm for edge–colouring a graph [8]. Apart from its intrinsic interest as a classical combinatorial problem, and as a paradigm example for locality in distributed computing, edge colouring is also useful from a practical standpoint because of its connection to scheduling. In distributed networks or architectures an edge colouring corresponds to a set of data transfers that can be executed in parallel. So, a partition of the edges into a small number of colour classes – i.e. a "good" edge colouring– gives an efficient schedule to perform data transfers (for more details, see [8, 2]). The analysis of edge colouring algorithms published in the literature is extremely

* Work partly done while at BRICS, Department of Computer Science, University of Aarhus, Denmark. Partially supported by the ESPRIT Long Term Research program of the EU under contract No. 20244 (ALCOM–IT)

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 60-70, 1998.

long and difficult and that in [8] is moreover, based an a certain *ad hoc* extension of the Chernoff-Hoeffding bounds. In contrast, our analysis is a very simple, short and streamlined application of Talagrand's inequalty, only two pages long.

In § 5, we outline how other edge and vertex colouring algorithms can also be tackled by the same methods in a general framework. These examples are intended moreover, as a dramatic illustration of the versatility and power of the method for the analysis of locality in distributed computing in general.

2 Distributed Edge Colouring

Vizing's Theorem shows that every graph G can be edge coloured sequentially in polynomial time with Δ or $\Delta + 1$ colours, where Δ is the maximum degree of the input graph (see, for instance, [1]).

It is a challenging open problem whether colourings as good as these can be computed fast in a distributed model. In the absence of such a result one might aim at the more modest goal of computing reasonably good colourings, instead of optimal ones. By a trivial modification of a well-known *vertex* colouring algorithm of Luby it is possible to edge colour a graph using $2\Delta - 2$ colours in $O(\log n)$ rounds (where n is the number of processors) [4].

We shall present and analyze a simple localised distributed algorithm that compute near optimal edge colourings. The algorithm proceeds in a sequence of rounds. In each round, a simple randomised heuristic is invoked to colour a significant fraction of the edges successfully. The remaining edges are passed over to succeeding rounds. This continues until the number of edges is small enough to employ a brute–force method at the final step. For example, the algorithm of Luby mentioned above can be invoked when the degree of the graph becomes small i.e. when the condition $\Delta \gg \log n$ is no longer satisfied.

First the algorithm invokes a reduction to bipartite graphs by a standard procedure, see [8]. We describe the action carried out by the algorithm in a single round starting with a bipartite graph. At the beginning of each round, there is a palette of fresh new available colours, $[\Delta]$, where Δ is the maximum degree of the graph at the current stage.

Algorithm P[1]: There is a two step protocol:

- Each bottom vertex, in parallel, makes a *proposal* independently of other bottom vertices by assigning a random *permutation* of the colours to their incident edges.
- Each top vertex, in parallel, then picks a *winner* out of every set of incident edges that have the same colour. Tentative colours of winner edges become final.
- The *losers*– edges who are not winners– are decoloured and passed to the next round.

[1] for *permutation*.

It is apparent that the algorithm is truly distributed. That is to say, each vertex need only exchange information with the neighbours to execute the algorithm. This and its simplicity makes the algorithms amenable for implementations in a distributed environment. Algorithm P is exactly the algorithm used in [8].

We focus all our attention in the analysis of one round of the algorithm. Let Δ denote the maximum degree of the graph at the beginning of the round and Δ' denote the maximum degree of the leftover graph. One can easily show that $\mathtt{E}[\Delta' \mid \Delta] \leq \beta\Delta$, for some constant $0 < \beta < 1$. The goal is to show that this holds with high probability. This is done in § 4 after the relevant tools – the concentration of measure inequalities – are introduced in the next section.

3 Concentration of Measure Inequalities

Talagrand's isoperimetric inequality for the concentration of measure involves a product space $\Omega = \prod_{i\in[n]} \Omega_i$, equppied with the product measure $P = \prod_{i\in[n]} P_i$ (where the P_is are arbitrary measures on the individual spaces) and, crucially. a certain notion of "convex distance" between a point $x \in \omega$ and a subset $A \subseteq \Omega$:

$$d_T(x, A) := \sup_{\sum \alpha_i^2 = 1} \min_{y\in A} \sum_{x_i \neq y_i} \alpha_i. \tag{1}$$

(The sup is over all reals $\alpha_1, \ldots, \alpha_n$ satisfying the stated condition.) For sets $A, B \subseteq \Omega$, set $d_T(A, B) := \min_{x\in A} d_T(x, B)$.

Talagrand's inequality [10] is:

$$P(A)P(B) \leq \exp\left(-d_T^2(A, B)/4\right), \quad A, B \subseteq \Omega. \tag{2}$$

In an alternative elegant approach, Marton [5, 6] shows that this inequality is in turn a consequence of certain information–theoretic inequalities. This requires an analgoue to (1) for distributions. Note that for a one point set $A = \{y\}$, the distance (1) is

$$d_T(x, y) = \sup_{\sum \alpha_i^2 = 1} \sum_{x_i \neq y_i} \alpha_i.$$

Now, for distributions P and Q on Ω, we define $d_T(P, Q)$ as the minimum of $\mathtt{E}[d_T(X, Y)]$ over all joint distributions of X and Y with marginals P and Q respectively.

We also require the notion of *conditional* or *relative entropy* $\mathtt{H}(P \mid Q)$ of distribution P with respect to Q:

$$\mathtt{H}(P \mid Q) := \sum_{\omega\in\Omega} P(\omega) \log \frac{P(\omega)}{Q(\omega)}. \tag{3}$$

(This is also sometimes called the *informational divergence* and denoted $D(P||Q)$.)

The information theoretic inequality is: if P is a product measure, then for any other measure Q,

$$d_T(P,Q) \leq \sqrt{2H(P \mid Q)}. \tag{4}$$

One can show that the concentration of measure inequality (2) is a consequence of (4).

Marton generalises (4) to the case when the underlying measure P is not the product measure (i.e. the corresponding variables are not independent). The setting is very similar to that in martingale inequalities where the underlying variables are "exposed" one by one. Suppose $k \leq n$ and $x_1^k, \hat{x}_1^k$ differ only in the last co–ordinate:

$$x_1^{k-1} = \hat{x}_1^{k-1}, x_k \neq \hat{x}_k.$$

We want to consider the corresponding conditional distributions on the remaining variables, once the first k variables are fixed to the values x_1^k and $\hat{x}_1^k$ respectively. Let us consider a *coupling* of the two conditional distributions $P(\cdot \mid X_1^k = x_1^k)$ and $P(\cdot \mid X_1^k = \hat{x}_1^k)$ and denote this by $\pi_k^n(\cdot \mid x_1^k, \hat{x}_1^k)$. This can be thought of as the joint distribution of a pair of random variable sequences, $(Y_k^n, \hat{Y}_k^n)$.

Now for $k < i \leq n$, define the "influence" coefficients $v_{i,k}(x_1^k, \hat{x}_1^k)$ by:

$$v_{i,k}(x_1^k, \hat{x}_1^k) := \left(\sum_{y_k^n} \left(\pi_k^n[\hat{Y}_i \neq y_i \mid Y_k^n = y_k^n] \right)^2 P[Y_k^n = y_k^n \mid Y_1^k = x_1^k] \right)^{1/4} \tag{5}$$

Put $v_{k,k}(x_1^k, \hat{x}_1^k) := 1$. Set

$$v_{i,k} := \max_{x_1^k, \hat{x}_1^k} v_{i,k}(x_1^k, \hat{x}_1^k).$$

Now define

$$U := \max_i \sum_{1 \leq \ell \leq i} v_{i,\ell}^2, \tag{6}$$

and

$$V := \max_k \sum_{k \leq i \leq n} v_{i,k}^2. \tag{7}$$

Then the generalisation of (4) is [6, Theorem3.1]:

$$d_T(P,Q) \leq \sqrt{UV}\sqrt{2H(P \mid Q)}. \tag{8}$$

As an example of this inequality, consider the space of the m–fold product of the symmetric group S_n. The measure on each component space is the uniform measure and on the product space we take the product measure. To compute the coefficients (5), let us expose the variables component by component. Having exposed all variables in the first $t < m$ components, suppose we have further

exposed $k < n$ variables in the $t+1$st component. We take the natural coupling that simply swaps the permutation in this component and is the identity everywhere else. For this coupling, an easy calculation shows that

$$v_{i,k}(x_1^k, \hat{x}_1^k) = 1/\sqrt{n-k},$$

for k within this component and zero everywhere else. Thus

$$\begin{aligned} U &= \max_i \sum_{\ell<i} v_{i,\ell}^2 \\ &= \max_i \sum_{1\leq\ell<i} \frac{1}{n-\ell} \\ &= H_n, \end{aligned}$$

where $H_n \approx \log n$ is the nth Harmonic number and

$$\begin{aligned} V &= \max_k \sum_{i>k} v_{i,\ell}^2 \\ &= \max_k \sum_{i>k} \frac{1}{n-k} \\ &= 1. \end{aligned}$$

Hence on the product space S_n^m of the m–fold product of the symmetric group, we get the inequality:

$$d_T(P,Q) \leq \sqrt{\log n}\sqrt{2H(P \mid Q)} \tag{9}$$

Actually Talagrand is able to prove a stronger result on the symmetric group. For the m–fold product of the symmetric group, he proves:

$$P(A)P(B) \leq \exp\left(-d_T^2(A,B)/16\right) \tag{10}$$

There is a simple packaged form in which it is often convenient to apply these inequalities. In this form, the two components of the analysis are separated from each other:

- The *probabilistic* component which gives a concentration of measure inequality.
- The *smoothness* of the function of interest.

In applications, the concentration of measure inequality is taken ready–made without extra effort and one has only to verify the smoothness conditions on the function, which is often very easy.

Theorem 1. *Let f be a real–valued function on a product space $\Omega = \prod_{i\in[n]} \Omega_i$ with a measure P (which is not necessarily with product measure). Suppose for*

each $x \in \Omega$, there are reals $\alpha_i(x), i \in [n]$ such that any **one** *of the following conditions holds:*

$$f(x) \leq f(y) + \sum_{x_i \neq y_i} \alpha_i(x), \quad \textit{for all } y \in \Omega, \tag{11}$$

or,

$$f(x) \geq f(y) - \sum_{x_i \neq y_i} \alpha_i(x), \quad \textit{for all } y \in \Omega. \tag{12}$$

– *If a measure concentration inequality holds in the form*

$$P(A)P(B) \leq \exp\left(-d_T^2(A,B)/\alpha\right), \tag{13}$$

for some $\alpha > 0$, and uniformly for all $x \in \Omega$,

$$\sum_i \alpha_i^2(x) \leq c, \tag{14}$$

then we have the following concentration result for f around its median value $\mathtt{M}[f]$:

$$P[|f - \mathtt{M}[f]| > t] \leq 2\exp\left(-\frac{t^2}{\alpha c}\right). \tag{15}$$

– *If a measure concentration inequality holds in the form*

$$d_T(P,Q) \leq \alpha\sqrt{2H(P \mid Q)} \tag{16}$$

for some $\alpha > 0$ and also the coefficients $\alpha_i(x)$ satisfy

$$\mathtt{E}[\sum_i \alpha_i^2(X)] \leq c, \tag{17}$$

then, we have the following concentration result for f around its mean:

$$P[|f - \mathtt{E}[f]| > t] \leq 2\exp\left(-\frac{t^2}{\alpha c}\right). \tag{18}$$

Note two features of the condition (11)or (12): first the asymmetry: we only need one of these one–sided versions to hold. Second its non–uniformity: the coefficients α_i are allowed to depend on x. Both these features contribute to the power of the inequalities in applications.

4 High Probability Analyses

4.1 Top vertices

The analysis is particularly easy when v is a top vertex in Algorithm P. For, in this case, the incident edges all receive colours independently of each other. This is exactly the situation of the classical balls and bins experiment: the incident

edges are the "balls" that are falling at random independently into the colours that represent the "bins". Let $T_e, e \in E$, be the random variables taking values in $[\Delta]$ that represent the tentative colours of the edges. Then the number of edges unsuccessfully coloured around v (and hence the new degree) is a function $f(T_e, e \in N^1(v))$, where $N^1(v)$ denotes the set of edges incident on v. It is easily seen that this function has the *Lipschitz* property with constant 1: changing only one argument while leaving the others fixed only changes th evalue of f by at most 1. Thus:

- We can take all coeffcients $\alpha_i = 1$ in (11).
- Since the variables $T_e, e \in N^1(v)$ are independent, we can apply the Talagrand inequality (2)for the product spaces $[\Delta]^{N^1(v)}$ when v is a "top" vertex.

Hence, applying the first part of Theorem 1, we get the following sharp concentration result easily:

Theorem 2. *Let v be a top vertex in algorithm P and let f be the number of edges around v that are successfully coloured in one round of the algorithm. Then,*

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq 2 \exp\left(\frac{-t^2}{4\Delta}\right),$$

For $t := \epsilon\Delta$ $(0 < \epsilon < 1)$, this gives an exponentially decreasing probability for deviations around the mean. If $\Delta \gg \log n$ then the probability that the new degree of any vertex deviates far from its expected value is inverse polynomial, i.e. the new max degree is sharply concentrated around its mean.

4.2 Bottom Vertices

The analysis for the "bottom" vertices in Algorithm P is more complicated in several respects. It is useful to see why so that one can appreciate the need for a more sophisticated analysis.

To start with, one could introduce an indicator random variable X_e for each edge e incident upon a bottom vertex v. These random variable are not independent however. Consider a four cycle with vertices v, a, w, b, where v and w are bottom vertices and a and b are top vertices. Let's refer to the process of selecting the winner (step 2 of the algorithm P) as "the lottery". Suppose that we are given the information that edge va got tentative colour red and lost the lottery— i.e. $X_{va} = 0$— and that edge vb got tentative colour green. We'll argue intuitively that given this, it is more likely that $X_{vb} = 0$. Since edge va lost the lottery, the probability that edge wa gets tentative colour red increases. In turn, this increases the probability that edge wb gets tentative colour green, which implies that edge vb is more likely to lose the lottery. So, not only are the X_e's not independent, but the dependency among them is particularly malicious.

One could hope to bound this effect by using Talagrand's inequality in it simplest form. This is also ruled out however, for two reasons. The first is that the tentative colour choices of the edges around a vertex are not independent.

This is because the edges incident ona vertex are assigned a permutation of the colours. The second reason applies even if we pretend that all edges act independently. The new degree of v, a bottom vertex in algorithm P is a function $f = f(T_e, e \in N(v))$, where $N(v)$ is the set of edges at distance at most 2 from v. Thus f depends on as many as $\Delta(\Delta-1) = \Theta(\Delta^2)$ edges. Even if f is Lipshitz with constants $d_i = 1$, this is not enough to get a strong enough bound because $d = \sum_i d_i^2 = \Theta(\Delta^2)$. Applying Theorem 1 as above would give the bound

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq 2\exp\left(-\frac{t^2}{\Theta(\Delta^2)}\right).$$

This bound however is useless for $t = \epsilon\mathtt{E}[f]$ since $\mathtt{E}[f] \approx \Delta/e$.

We will use Marton's extension of Talagrand's inequality to handle the dependence and we shall use the asymmetric and non–uniform nature of the condition (11) to control the effects of the individual random choices much more effectively.

Let $N^1(v)$ denote the set of "direct" edges– i.e. the edges incident on v– and let $N^2(v)$ denote the set of "indirect edges" that is, the edges incident on a neighbour of v. Let $N(v) := N^1(v) \bigcup N^2(v)$. The number of edges unsuccessfully coloured at vertex v is a function $f(T_e, e \in N(v))$.

For a tentative colouring $T_e = c_e$, choose the coefficients $\alpha_i(c)$ as follows. Recall that edges compete in a "lottery" at the top vertices: for each colour, one "winner" is picked by a top vertex out of all the edges incident on it that receive the same tentative colour. Choose α_i to be 1 for all unsuccessfully coloured edges around v and for all "winners" (w, z) such that the edge (v, w) took part in the lottery that (w, z) won (hence (w, z) was responsible for (v, w) being unsuccessful and serves as a *witness* for this fact). All other α_i are 0. Thus at most 2Δ edges have non–zero α_i values, and hence $\sum_i \alpha_i^2 \leq 2\Delta$. To see that (11) holds, let us look at an unsuccessfully coloured edge e in the colouring x. If it is also unsuccessful in the colouring y, it is counted in the term $f(y)$. Otherwise, at least one of e or its "witness" e' must be coloured differently in y and this will be counted in the second term in the right–hand side.

The underlying space is $S_\Delta^{\bar{N}^2(v)}$ where $\bar{N}^2(v) := \{u \mid d(u, v) = 2\} \bigcup \{v\}$. Now applying the measure concentration result for the m–fold product of the symmetric group S_Δ from Marton's inequality, namely (9) and using the second part of Theorem 1, we arrive at the following sharp concentration result:

Theorem 3. *Let v be a bottom vertex in algorithm P and let Let f be the number of edges successfully coloured around v in one stage of either algorithm. Then,*

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq 2\exp\left(-\frac{t^2}{2\Delta\sqrt{\log\Delta}}\right).$$

For $t = \epsilon\Delta$, this gives a probability that decreases almost exponentially in Δ. As remarked earlier, if $\Delta \gg \log n$, this implies that the new max degree is sharply concentrated around the mean (with failure probability roughly inverse polynomial in n).

One can improve this by using the stronger inequality (10) for the symmetric group. This gives the bound:

$$\mathtt{Pr}[|f - \mathtt{M}[f]| > t] \leq 2\exp\left(-\frac{t^2}{32\Delta}\right).$$

We first gave the result with Marton's inequality because although it gives a somewhat weaker bound in this example, it actually illustrates a general method with wider applicability.

5 A General Framework

The method used in the last section is actually applicable in a much more general way to the analysis of a wide range of randomised algorithms in a distributed setting. In this section, we outline a fairly general framework in which such concentration results can be directly inferred. Let f be a function to be computed by a randomised local algorithm in a distributed environment respresented by a graph $G = (V, E)$. We shall lay down conditions on f and on algorithms computing f locally that will enable the methods in the previous section to be extended to to derive a sharp concentration result on f. In particular, we indicate how the edge colouring algorithm above [8] as well as the edge and vertex colouring algorithms from [2, 3] follow directly as well as the analysis of a vertex colouring process in [7].

Suppose that f is a function determined by each vertex v of the graph assigning labels $\ell(v)$ to itself and labels $\ell(v, e), e \in N(v)$ to its incident edges by some randomised process. In the edge colouring problem, the labels on the edges are their colours (we may assume for instance that the lower numbered vertex assigns the colour to an incident edge) and the vertex labels are empty; in the vertex colouring problems, the vertex labels are the colours and the edge labels are empty.

The two necessary and sufficient conditions for sharp concentration of f are as follows:

L: Locality of the function: The function f is *Lipschitz*: changing any one label changes the value of f by at most 1. Furthermore, following Spencer [9], f is *h–certifiable* for some function $h : R \to R$ i.e. for each x, there is a subset $I = I(x)$ (the "certificate") of the labels of size at most $h(f(x))$ such that for any other y agreeing with x on I, $f(y) \geq f(x)$. In edge colouring, the function f is the number of edges unsuccessfully coloured around a vertex. It is clearly Lipschitz. Also it is h certifiable for $h(x) = 2x$ since each unsuccessfully coloured vertex can be attributed to one other adjacent edge of the same colour.

M: Concentration of Measure in the Probability Space: There is concentration of measure in the underlying space in one of the two forms (13) or (16) with some positive coefficient α. Marton's inequality gives a method for determining this coefficient as $\alpha = \sqrt{UV}$ for U and V determined by (5)

through (7). Two particularly important cases in which this obtains are the following. First we assume that the vertex label assigned by a vertex is independent of the edge labels it assigns and that each vertex acts independently of the others.

I: The labels $\ell(v,e)$ assigned by a vertex to its incident edges are independent of each other. In this case, the algorithm defines a fully independent probability space and Talagrand's inequality (2) applies.

S: The labels $\ell(v,e)$ assigned by a vertex to its incident edges are given by a permutation distribution. In this case, the algorithm is *symmetric* with respect to the labels. Marton's theorem yields measure concentration in the form (16) with $\alpha = \sqrt{\log n}$ and Talagrand's inequality for the symmetric group gives measure concentration in the form (13) with $\alpha = 16$.

The condition **I** obtains in the algorithms in [3, 7] while the condition **S** obtains in the algorithm of [8] discussed in detail above.

Theorem 4. *Let f be a Lipschitz function which is h–certifiable computed by an algorithm satisfying the measure concentration property* **M** *for some $\alpha > 0$. Then Then*

$$\mathtt{Pr}[|f - \mathtt{M}[f]| > t] \leq 2\exp\left(-\frac{t^2}{\alpha h(\mathtt{M}[f])}\right).$$

In particular if the algorithm satisfies either full independence **I** *or is symmetric, satisfying* **S**, *then we have the concentration result:*

$$\mathtt{Pr}[|f - \mathtt{M}[f]| > t] \leq 2\exp\left(-\frac{t^2}{ch(\mathtt{M}[f])}\right),$$

where the constant c is 4 for the independent case and 16 for the symmetric case.

6 Acknowledgement

I would like to thank Kati Marton for her invaluable help in understanding how to apply her result. I would like to thank Alessandro Panconesi for his help in drafting this paper, for invaluable discussions and for resolutely prodding me to pursue the "demise of the permanent" from [8]. Now, finally, R.I.P.

References

1. Bollabas, B. : Graph Theory: An Introductory Course. Springer–Verlag 1980.
2. Dubhashi, D., Grable, D.A., and Panconesi, A.: Near optimal distributed edge colouring via the nibble method. Theoretical Computer Science 203 (1998), 225–251, a special issue for ESA 95, the 3rd European Symposium on Algorithms.
3. Grable, D., and Panconesi, A: Near optimal distributed edge colouring in $O(\log\log n)$ rounds. Random Structures and Algorithms **10**, Nr. 3 (1997) 385–405
4. Luby, M.: Removing randomness in parallel without a processor penalty. J. Computer and Systems Sciences **47**:2 (1993) 250–286.

5. Marton, K.: Bounding $\overline{d}$ distance by informational divergence: A method to prove measure concentration. Annals of Probability **24** (1996) 857–866.
6. Marton, K.: On the measure concentration inequality of Talagrand for dependent random variables. submitted for publication. (1998).
7. Molloy, M. and Reed, B.: A bound on the strong chromatic index of a graph. J. Comb. Theory (B) **69** (1997) 103–109.
8. Panconesi, A. and Srinivasan, A.: Randomized distributed edge coloring via an extension of the Chernoff–Hoeffding bounds. SIAM J. Computing **26**:2 (1997) 350–368.
9. Spencer, J.: Probabilistic methods in combinatorics. In proceedings of the International Congress of Mathematicians, Zurich, Birkhauser (1995).
10. Talagrand, M.: Concentration of measure and isoperimetric inequalities in product spaces. Publ. math. IHES, **81**:2 (1995) 73–205..

On-line Bin-Stretching

Yossi Azar[1] and Oded Regev[2]

[1] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.
azar@math.tau.ac.il ***
[2] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.
odedr@math.tau.ac.il

Abstract. We are given a sequence of items that can be packed into m unit size bins. In the classical bin packing problem we fix the size of the bins and try to pack the items in the minimum number of such bins. In contrast, in the bin-stretching problem we fix the number of bins and try to pack the items while stretching the size of the bins as least as possible. We present two on-line algorithms for the bin-stretching problem that guarantee a stretching factor of $5/3$ for any number m of bins. We then combine the two algorithms and design an algorithm whose stretching factor is 1.625 for any m. The analysis for the performance of this algorithm is tight. The best lower bound for any algorithm is $4/3$ for any $m \geq 2$. We note that the bin-stretching problem is also equivalent to the classical scheduling (load balancing) problem in which the value of the makespan (maximum load) is known in advance.
Keywords. On-line algorithms, approximation algorithms, bin-stretching, load balancing, scheduling, bin-packing.

1 Introduction

The on-line bin-stretching problem is defined as follows. We are given a sequence of items that can be packed into m bins of unit size. We are asked to pack them in an on-line fashion minimizing the stretching factor of the bins. In other words, our goal is to stretch the sizes of the bins as least as possible to fit the sequence of items. Bin-stretching is somewhat related to the bin-packing problem [10, 13, 18]. In both cases all the items are to be packed in bins of a certain size. However, in bin-packing the goal is to minimize the number of bins while in bin-stretching the number of bins is fixed and the goal is to minimize the stretching factor of the bins. Hence, results for bin packing do not seems to imply results for the bin-stretching problem.

A bin-stretching algorithm is defined to have a stretching factor β if for every sequence of items that can be assigned to m bins of a unit size, the the algorithm assigns the items to m bins of size of at most β.

The motivation for our problem comes from the following file allocation problem. Consider a case in which a set of files are stored on a system of m servers,

*** Research supported in part by the Israel Science Foundation and by the United States-Israel Binational Science Foundation (BSF).

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 71-81, 1998.

each of some unit capacity. The files are sent one by one to a remote system of m servers in some order. The only information the remote system has on the files is that they were originally stored on m servers of unit capacity. Our goal is to design an algorithm that can assign the arriving sequence of files on the remote system with the minimum capacity required. An algorithm for our problem whose stretching factor is β can assign the sequence of jobs to servers of capacity β.

It is also natural to view the bin-stretching problem as scheduling (load balancing) problem. In the classical on-line scheduling (load balancing) problem there are m identical machines and n jobs arriving one by one. Each job has some weight and should be assigned to a machine upon its arrival. The makespan (load) of a machine is the sum of the weights of the jobs assigned to it. The objective of an assignment algorithm is to minimize the makespan (maximum load) over all machines. In the bin-stretching problem we have the additional information that the optimal load is some known value and the goal is to minimize the maximum load given this information.

It is clear that an upper bound for the classical scheduling (load balancing) problem is also an upper bound for the bin-stretching problem since we may ignore the knowledge of the optimal makespan (load). The classical scheduling problem was first introduced by Graham [14, 15] who showed that the greedy algorithm has a performance ratio of exactly $2 - \frac{1}{m}$ where m is the number of machines. Better algorithms and lower bounds are shown in [7, 8, 9, 11, 12, 19, 21]. Recently, Albers [1] designed an algorithm whose performance ratio is 1.923 and improved the lower bound to 1.852.

The only previous result on bin-stretching is for two machines (bins). Kellerer et al. [20] showed that the performance ratio ratio is exactly 4/3 for two machines. For $m > 2$ there were no algorithms for bin-stretching that achieve a better performance than those for scheduling. In this paper we provide for the first time algorithms for bin-stretching on arbitrary number of machines (bins) that achieve better bounds than the scheduling/load-balancing results. Specifically, we show the following results:

- Two algorithms for the bin-stretching problem whose stretching factor is 5/3 for any number m of machines (bins).
- An improved algorithm which combines the above two algorithms whose stretching factor is 1.625 for any number m of machines (bins). Our analysis for the stretching factor of this algorithm is tight (for large m).
- For a fixed number $m \geq 3$ we get an upper bound $\frac{5m-1}{3m+1}$ which is better than 1.625 for $m \leq 20$.
- Also, we easily extend the lower bound of 4/3 on the stretching factor of any deterministic algorithm for $m = 2$ for any number $m \geq 2$.

Observe that the additional information that bin-stretching has over the scheduling problem really helps in improving the performance of the algorithms. Moreover, our upper bounds for the bin-stretching problem are lower than the lower bounds for the classical load balancing problem for all $m \geq 2$ and this fact separates the two problems.

Note that the notion of stretching factor has been already used for various problems and, in particular, for scheduling. A paradigm that is used for attacking many of the off-line and on-line problems is to design algorithms that know an upper bound on the value of the optimal algorithm. Binary search for the optimal value is used in the off-line setting. In fact, this is the way that scheduling is reduced to bin-stretching by the polynomial approximation scheme of [17]. This paradigm is also used for the related machines model [16] which corresponds to bins of different sizes. In the on-line case the paradigm of stretching factor is used with a doubling technique. Reducing the case of unknown optimal value to known optimal value results in loosing a factor of 4 [2]. The notion of stretching factor has also been used in the temporary jobs model where jobs arrive and depart at arbitrary times [3, 4, 5, 6].

2 Notation

Let M be a set of machines (bins) and J a sequence of jobs (items) that have to be assigned to the machines (bins). Each job j has an associated weight, $w_j \geq 0$. As job j arrives it must be permanently assigned to one of the machines. An assignment algorithm selects a machine i for each arriving job j. Whenever we speak about time j we mean the state of the system after the jth job is assigned. Let $l_i(j)$ denote the load on machine i at time j, i.e., the sum of the weights of all the jobs on machine i at time j. The cost of an assignment algorithm A on a sequence of n jobs J is defined as the maximum load over all machines, or, $C_A(J) = \max_{i \in M} l_i(n)$.

The objective of an on-line bin-stretching algorithm is to minimize the stretching factor β; i.e., the cost of a sequence of jobs given that the optimal off-line assignment algorithm (that knows the sequence of jobs in advance) assigns them at a unit cost. This is unlike the classical on-line scheduling (load balancing) problems where the optimal cost is not known in advance and the performance is measured by the regular competitive ratio which is defined as the supremum of the ratio between the cost of the on-line assignment and the cost of the optimal off-line assignment.

We say that a sequence of jobs can be assigned to m machines by an optimal off-line algorithm if it can be assigned with a unit cost. We note some simple properties of such sequences of jobs. First, the weight of all jobs must be at most 1 since a job that is larger than 1 cannot be assigned by any algorithm without creating a load larger than 1. Second, the sum of weights of all jobs in a sequence of jobs is at most m, the number of machines. That follows from the fact that the optimal off-line algorithm can assign jobs with total weight of at most 1 to each machine.

3 Two algorithms with 5/3 stretching factor

In this section we present two algorithms with a stretching factor of 5/3 for the on-line bin-stretching problem. These are actually two families of algorithms. For each family we prove the same 5/3 upper bound.

We start with a simple algorithm with a stretching factor of 2: put each arriving job on an arbitrary machine such that the resulting load on that machine will not exceed 2. Obviously, if the algorithm does not fail to find such machine it has a stretching factor of 2 by definition. In order to show that such a machine is always available we notice that there must be a machine whose load is at most 1. Otherwise, all the machines have loads larger than 1 which contradicts the fact that the optimal solution has maximal load 1. Since the weight of each job is at most 1, each arriving job can be assigned to some machine which implies that the algorithm never fails.

Our algorithms use a threshold α to classify machines according to their loads. An appropriate choice of α will lead as described later to an algorithm whose stretching factor is $1+\alpha$.

Definition 1. *A machine is said to be short if its load is at most α. Otherwise, it is tall.*

At the arrival time of job j, we define three disjoint sets of machines based on the current load and the job's weight.

Definition 2. *When job j arrives, $1 \leq j \leq n$, define the following three disjoint sets:*

- $S_1^\alpha(j) = \{i \in M \mid l_i(j-1) + w_j \leq \alpha\}$
- $S_2^\alpha(j) = \{i \in M \mid l_i(j-1) \leq \alpha,\ \alpha < l_i(j-1) + w_j \leq 1+\alpha\}$
- $S_3^\alpha(j) = \{i \in M \mid l_i(j-1) > \alpha,\ l_i(j-1) + w_j \leq 1+\alpha\}$

The set S_1 is of machines that are short and remain short if the current job is placed on them. The second set S_2 is of machines that are short but become tall if the job is placed on them. The last set S_3 is of machines that are tall but remain below $1+\alpha$ if the job is placed on them. Note that there may be machines which are not in any of the sets. We omit the indices j and α when they are clear from the context.

Using this definition we can now describe the two algorithms:

$ALG1_\alpha$**:** When job j arrives:

- Put the job on any machine from the set S_3 or S_1 but not on an empty machine from S_1 if there is a non-empty machine from S_1.
- If $S_1 = S_3 = \phi$ then put the job on the least loaded machine from the set S_2.
- If $S_1 = S_2 = S_3 = \phi$ then report failure.

$ALG2_\alpha$**:** When job j arrives:

- Put the job on any machine from the set S_1.

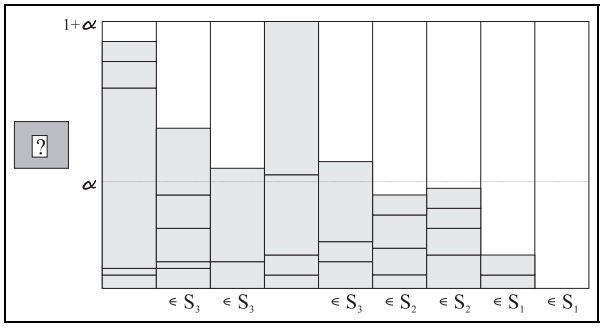


Fig. 1. S_1, S_2 and S_3

- If $S_1 = \phi$ then put the job on any machine from the set S_3.
- If $S_1 = S_3 = \phi$ then put the job on the least loaded machine from the set S_2.
- If $S_1 = S_2 = S_3 = \phi$ then report failure.

Notice that these two algorithms are actually families of algorithms. In the first algorithm we are free to choose how to select a machine from S_3 and whether we put a job on a machine from S_1 or from S_3. In the second algorithm we are free to choose how to select a machine from S_1 and from S_3.

Note that since the algorithms assign job j only to machines from the sets $S_1(j)$, $S_2(j)$ and $S_3(j)$, their stretching factor is at most $1 + \alpha$ as long as they do not fail. For $1 \leq i \leq 3$ let J_i be the set of jobs j assigned to a machine in $S_i(j)$ at their arrival time by the algorithm.

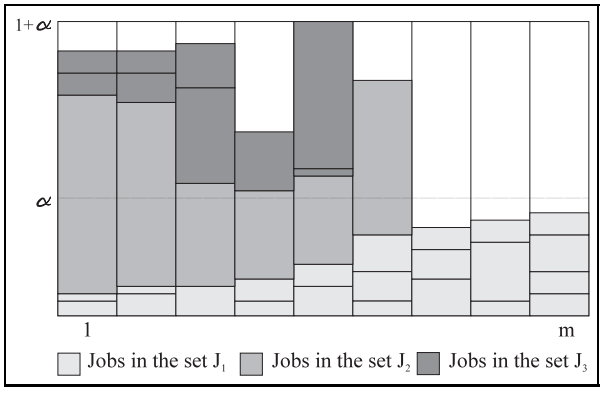


Fig. 2. J_1, J_2 and J_3

Theorem 1. *$ALG1_\alpha$ above never fails for $\alpha \geq 2/3$. Therefore, for $\alpha = 2/3$ it has a stretching factor of $5/3$.*

Theorem 2. *$ALG2_\alpha$ above never fails for $\alpha \geq 2/3$. Therefore, for $\alpha = 2/3$ it has a stretching factor of $5/3$.*

In order to prove the above theorems we assume by contradiction that $ALG1_\alpha$ or $ALG2_\alpha$ fail on the last job of some sequence of $n+1$ jobs and that this sequence can be assigned by an optimal algorithm. We start with the following simple lemmas:

Lemma 1. *At time n all the machines are tall and there are at least two machines whose load is less than 1..*

Proof. At time n, when the last job arrives, the three sets, S_1, S_2 and S_3 are empty. Hence, $l_i(n)+w_{n+1} > 1+\alpha$ for all $1 \leq i \leq m$. Since the weight of each job is at most 1, $l_i(n) > 1+\alpha-w_{n+1} \geq \alpha$ for all $1 \leq i \leq m$. Thus, all the machines are tall. Assume by contradiction that except a machine i, all the machines have loads of 1 or more. When the last job comes, $l_i(n) + w_{n+1} > 1 + \alpha > 1$ and since all other machines also have loads of 1 or more it implies that the sum of all loads is above m which contradicts the fact that the sequence of jobs can be assigned by an optimal algorithm.

Corollary 1. *The last job is larger than α.*

Proof. At time n, when the last job arrives, there is a machine i whose load is less than 1 by lemma 1. Since the algorithm fails to assign the last job, $1 + w_{n+1} > 1 + \alpha$ or $w_{n+1} > \alpha$.

To utilize some of our lemmas for the improved algorithm we use a more general formulation. Consider a subset $M' \subseteq M$ of machines. We define the notion of composed algorithm $D(ALG, M')$ where ALG is $ALG1_\alpha$ or $ALG2_\alpha$ on a sequence of jobs I and a set of machines M as follows: The algorithm decides on an arbitrary set $I' \subseteq I$ and assigns it to a machine in M' and it assigns the rest of the jobs to a machine in $M - M'$. The assignment of jobs I' is done by running algorithm ALG on the set of machines M'. However, the jobs in $I - I'$ are assigned to a machine in $M - M'$ in any arbitrary way. Moreover, we make no assumption on the sequence I, for example, the optimal algorithm may not be able to assign them in M without exceeding a load of 1 (in particular, jobs of weight larger than 1 may exist).

Note that $D(ALG, M')$ is the same as ALG for $M' = M$. We already proved that if $ALG1_\alpha$ or $ALG2_\alpha$ fail on the $n+1$ job of sequence J of jobs then at time n all the machines are tall and there are two machines whose load is less than 1. Meanwhile, for the composed algorithms we assume that after a sequence of n jobs I was assigned by $D(ALG, M')$ all the machines from the set M' are tall and two of them have loads below 1. This assumption is used until (including) lemma 6. Also, we assume that $0 \leq \alpha \leq 1$ unless otherwise specified.

Define the *raising job* k_i of machine $i \in M'$ as the job that raises machine i from being short to being tall. More formally, $l_i(k_i) > \alpha$ and $l_i(k_i - 1) \leq \alpha$. The raising jobs are well defined since we assumed that all machines from M' are tall. Rename the indices of the machines in M' to $1, \ldots, m'$ such that $k_1 < k_2 < \ldots <$

$k_{m'}$ i.e., the order of the machines in M' is according to the time the machines crossed α. From now on, all the indices are according the the new order. Note that the set of the raising job is J_2. Denote by s_1, s_2 the two machines in M' $(s_1 < s_2)$ whose load is less than 1 at time n.

Lemma 2. *If at time n, the load of some machine $i \in M'$ is at most l then $w_{k_{i'}} > 1 + \alpha - l$ for $i' > i, i' \in M'$.*

Proof. Both $ALG1_\alpha$ and $ALG2_\alpha$ assign jobs to machines from S_2 only if the two other sets are empty. By definition of $k_{i'}$, at time $k_{i'} - 1$, job $k_{i'}$ arrived and was assigned to machine i'. By the definitions of S_2 and $k_{i'}$, machine i' was in the set $S_2(k_{i'})$ and therefore the sets $S_1(k_{i'})$ and $S_3(k_{i'})$ were empty. Machine i was already tall at that time since $i < i'$. This implies that at time $k_{i'} - 1$ machine i was not in $S_2(k_{i'})$. Hence $l_i(k_{i'} - 1) + w_i > 1 + \alpha$ or $w_i > 1 + \alpha - l_i(k_{i'} - 1) \geq 1 + \alpha - l_i(n) \geq 1 + \alpha - l$.

Since we assumed the load of machine s_1 is at most 1 at time n, the lemma above implies:

Corollary 2. *Jobs k_i for $s_1 < i \leq m'$ are larger than α.*

Let $f_i = l_i(k_i - 1)$ for $1 \leq i \leq m'$. This is the load of each machine just before it was raised by the raising job.

Fig. 3. The series f_i. Only machines from M' are shown for clarity.

Lemma 3. *For $i' > i$, both in M', $f_i \leq l_{i'}(k_i - 1) \leq f_{i'}$.*

Proof. At time $k_i - 1$ the load of machine i is f_i by definition. At this time, by definition of k_i, machine i is in the set S_2 which means that S_1 and S_3 are empty. Thus, at the same time, each machine $i' > i$ must be in S_2 or not in any of the sets. Note that if the load of machine i' is below f_i at time $k_i - 1$ then

it is in $S_2(k_i)$ since machine i, whose load is higher, is in $S_2(k_i)$. Therefore, the load of machine i' is at least f_i since both algorithms choose the least loaded machine from S_2. Machine i' is still short so its load is at most $f_{i'}$.

Corollary 3. *The series f_i , $1 \leq i \leq m'$, is non-decreasing.*

Lemma 4. *For $i \leq s_2$, $f_i < 1 - \alpha$.*

Proof. According to corollary 2, $w_{k_{s_2}} > \alpha$. Since the load of machine s_2 is below 1 at time n,

$$1 > l_{s_2}(n) \geq l_{s_2}(k_{s_2}) = l_{s_2}(k_{s_2} - 1) + w_{k_{s_2}} = f_{s_2} + w_{k_{s_2}}.$$

Therefore,

$$f_{s_2} < 1 - w_{k_{s_2}} < 1 - \alpha.$$

By corollary 3, $f_i < 1 - \alpha$ for $i \leq s_2$.

Note that up to now our proof was not specific to one of the algorithms. Now we focus our attention on the first algorithm. Recall that we still assume that the set of jobs I is assigned by algorithm $D(ALG1_\alpha, M')$ or $D(ALG2_\alpha, M')$ to the set of machines M.

Lemma 5. *At any time of the activity of $D(ALG1_\alpha, M')$, there is at most one non empty machine in M' whose load is at most $\frac{\alpha}{2}$.*

Proof. Assume by contradiction that at a certain time there are two such machines. Let j be the first job that its assignment created two such machines. Thus, job j arrived and was placed on an empty machine i_2 while another non empty machine i_1 had a load of at most $\frac{\alpha}{2}$. Clearly $w_j \leq \frac{\alpha}{2}$ and $l_{i_1}(j-1) + w_j \leq \alpha$. Therefore $i_1 \in S_1(j)$ and job j should have been assigned to i_1.

Lemma 6. *Assume $\alpha \geq 2/3$ and $D(ALG1_\alpha, M')$ assigns a set of n jobs I to a set of machines M. Then the weight of each job k_i, $1 \leq i \leq m'$, is more than α.*

Proof. We have already seen in corollary 2 that jobs k_i for $s_1 < i \leq m'$ are larger than α. Now we show that jobs k_i for $i \leq s_1$ are also larger than α.

By lemma 4, f_{s_1} and f_{s_2} are both below $1 - \alpha$. According to lemma 3, $f_{s_1} \leq l_{s_2}(k_{s_1} - 1) \leq f_{s_2}$. Recall that the load of machine s_1 at time $k_{s_1} - 1$ is f_{s_1}. At that time, the loads of machines s_1 and s_2 are below $1 - \alpha \leq \frac{\alpha}{2}$. Thus, by lemma 5, the less loaded machine, s_1, is empty, or $l_{s_1}(k_{s_1} - 1) = f_{s_1} = 0$. By corollary 3, $f_i = 0$ for all machines $i \leq s_1$. A small f_i implies that machine i has a large raising job. More formally, for $i \leq s_1$:

$$w_{k_i} = l_i(k_i) - l_i(k_i - 1) = l_i(k_i) - 0 > \alpha.$$

Now we are ready to complete the proof of theorem 1. Assume that $ALG1_\alpha$ fails on the $n+1$ job of a sequence J of jobs. After the n jobs have been assigned, all the machines are tall and there are two machines whose load is less than 1

by lemma 1. We take $M' = M$ and therefore $I' = I$ where I is the set of jobs J without the last job. The previously defined series k_i is now defined over all machines since we took $M' = M$. By lemma 6, for $\alpha \geq 2/3$, this implies that there are m jobs larger than α. Corollary 1 shows that the last job is also larger than α. We showed there are $m+1$ jobs larger than α. This contradicts the fact that the number of jobs of weight larger than $1/2$ is at most m since the optimal algorithm can assign at most one such job to each machine. This completes the proof of theorem 1.

The proof of theorem 2, i.e. $ALG2_\alpha$ has the same stretching factor, is omitted.

4 Improved Algorithm

In this section we present an improved algorithm whose stretching factor is 1.625. The improved algorithm combines both of the previous algorithms into a single algorithm.

At the arrival time of job j we define five disjoint sets of machines based on the current load and the job's weight.

Definition 3. *When job j arrives, $1 \leq j \leq n$, define the following five sets:*

- $S_{11}^\alpha(j) = \{i \in M \mid l_i(j-1) + w_j \leq \alpha,\ l_i(j-1) + w_j \leq 2\alpha - 1\}$
- $S_{12}^\alpha(j) = \{i \in M \mid l_i(j-1) + w_j \leq \alpha,\ l_i(j-1) \leq 2\alpha - 1,\ l_i(j-1) + w_j > 2\alpha - 1\}$
- $S_{13}^\alpha(j) = \{i \in M \mid l_i(j-1) + w_j \leq \alpha,\ l_i(j-1) > 2\alpha - 1\}$
- $S_2^\alpha(j) = \{i \in M \mid l_i(j-1) \leq \alpha,\ \alpha < l_i(j-1) + w_j \leq 1 + \alpha\}$
- $S_3^\alpha(j) = \{i \in M \mid l_i(j-1) > \alpha,\ l_i(j-1) + w_j \leq 1 + \alpha\}$

Note that the previously defined S_1 is split into three sets according to a low threshold of $2\alpha - 1$. We still use the notation S_1 for the union of these three sets. We omit the indices j and α when they are clear from the context. The sets J_1, J_2 and J_3 are defined as in the previous section.

Improved Algorithm: When job j arrives:

- Put the job on a machine from the set S_1 according to:
  - Put the job on any machine from the set S_{13} or S_{11} but not on an empty machine from the set S_{11} if there is a non-empty machine from the set S_{11}.
  - If $S_{11} = S_{13} = \phi$ then put the job on the least loaded machine from the set S_{12}.
- If $S_1 = \phi$ then put the job on the *earliest* machine from the set S_3, that is, the machine that was the first to cross the threshold α from all machines in S_3.
- If $S_1 = S_3 = \phi$ then put the job on the least loaded machine from the set S_2.
- If $S_1 = S_2 = S_3 = \phi$ then report failure.

This improved algorithm is contained in the family of $ALG2_\alpha$ presented in the last section. Our algorithm, however, defines the methods used in placing jobs on machines from the sets S_1 and S_3. The way we choose a machine from S_1 is by the method presented in $ALG1_\alpha$. In choosing a machine from S_3 we prefer the earliest machine according to the order of crossing the threshold. The proof of the theorem below is omitted.

Theorem 3. *The improved algorithm above never fails for* $5/8 \leq \alpha \leq 2/3$. *Thus, for* $\alpha = 5/8$ *it has a stretching factor of* $13/8$.

5 Lower Bounds

In this section we prove a general lower bound of $4/3$ on the stretching factor of deterministic algorithms for any number of machines. We show a lower bound of $5/3 - \epsilon$ for arbitrary small ϵ for the family of $ALG1_\alpha$ and a lower bound of $13/8 - \epsilon$ for arbitrary small ϵ on the stretching factor of our improved algorithm. Note that it is impossible to show a lower bound of $5/3 - \epsilon$ for $ALG2_\alpha$ since the improved algorithm is in that family. In these two cases we assume the number of machines is large enough. The details of all the lower bounds are omitted.

References

[1] S. Albers. Better bounds for on-line scheduling. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 130–139, 1997.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. 25th ACM Symposium on the Theory of Computing*, pages 623–631, 1993. Also in *Journal of the ACM* 44:3 (1997) pp. 486–504.

[3] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 321–327, 1994.

[4] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, pages 218–225, 1992. Also in *Theoretical Compute Science* 130 (1994) pp. 73-84.

[5] Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. In *5th Israeli Symp. on Theory of Computing and Systems*, pages 119–125, 1997.

[6] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Proc. Workshop on Algorithms and Data Structures*, pages 119–130, August 1993.

[7] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symposium on Theory of Algorithms*, pages 51–58, 1992. To appear in *Journal of Computer and System Sciences.*

[8] B. Chen, A. van Vliet, and G. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51:219–222, 1994.

[9] B. Chen, A. van Vliet, and G. J. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16:221–230, 1994.

[10] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation algorithms*. 1996.

[11] U. Faigle, W. Kern, and G. Turan. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[12] G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *SIAM J. Computing*, 22:349–355, 1993.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, 1979.

[14] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[15] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:263–269, 1969.

[16] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[17] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. of the ACM*, 34(1):144–162, January 1987.

[18] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[19] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 132–140, 1994.

[20] H. Kellerer, V. Kotov, M. G. Speranza, and Zs. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*. To appear.

[21] J. Sgall. On-line scheduling on parallel machines. Technical Report Technical Report CMU-CS-94-144, Carnegie-Mellon University, Pittsburgh, PA, USA, 1994.

Combinatorial Linear Programming: Geometry Can Help *

Bernd Gärtner

Institut für Theoretische Informatik, ETH Zürich, ETH-Zentrum, CH-8092 Zürich, Switzerland (gaertner@inf.ethz.ch)

Abstract. We consider a class $\mathcal{A}$ of generalized linear programs on the d-cube (due to Matoušek) and prove that Kalai's subexponential simplex algorithm RANDOM-FACET is polynomial on all actual linear programs in the class. In contrast, the subexponential analysis is known to be best possible for general instances in $\mathcal{A}$. Thus, we identify a "geometric" property of linear programming that goes beyond all abstract notions previously employed in generalized linear programming frameworks, and that can be exploited by the simplex method in a nontrivial setting.

1 Introduction

While Linear Programming (LP) is known to belong to the complexity class **P** [17], its *combinatorial* complexity in the unit cost (RAM) model is as yet unresolved. This means, it is not known whether there is a polynomial $p(n, d)$, such that every linear program with n constraints in d variables can be solved in time $p(n, d)$, if all arithmetic operations are assumed to incur unit cost. In other words, no *strongly* polynomial algorithm for LP is known. One motivation for getting down to this problem is the *simplex method*, the oldest and still most widely used algorithm to solve LP [6]. The simplex method naturally lends itself to unit cost analysis, where the basic complexity measure is the number of *pivot steps* which in turn depends on the *pivot rule* chosen. The hope is that eventually a pivot rule is discovered where this number can be bounded by a polynomial in n and d.

Previous results in this direction are rather discouraging. For the pivot rule originally proposed by Dantzig [6], Klee and Minty have constructed a class of LP (the so-called "Klee-Minty cubes") where this rule leads to an exponential number of steps [18]. In the sequel, such worst-case examples have been found by various researchers for almost all known deterministic pivot rules, see [12] for an overview and [2] for a new unified view of these examples.

A few years ago, progress has been made using *randomized* pivot rules. The subexponential bounds independently established by Kalai [15] as well as Matoušek, Sharir and Welzl [19] are far from being polynomial, but they still represent a breakthrough towards the goal of understanding the complexity of LP

* supported by the Swiss Science Foundation (SNF), project No. 21-50647.97.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 82-96, 1998.

in the RAM model. The currently best algorithm based on these results invokes algorithms by Clarkson and achieves (for $n \geq d$) expected runtime

$$O(d^2 n + \exp(O(\sqrt{d \log d}))),$$

see [11] for a survey. If $n = O(d)$, one can even prove a bound of $\exp(O(\sqrt{d}))$ [15].

A remarkable fact about the subexponential pivot rules is that they are *combinatorial* in the sense that actual coordinates of the LP are only used to decide whether progress is possible in a certain situation, but not to measure, let alone maximize that progress in any way. This is in sharp contrast to the pivot rules employed in practice, where strategies that adapt to the given instance as well as possible are typically most successful.

In the theoretical setting, this "ignorance" is rather a strong point, in particular when combined with randomization (which is just an elegant way to deal with ignorance). The worst-case constructions above all work by "penalizing" behavior that is either based on the coordinates of the problem (like in Dantzig's rule) or on deterministic choices foreseeable in advance (like in case of Bland's rule [4]). Randomized combinatorial rules are harder to fool by such examples, and the subexponential bounds established in 1992 are still the best known worst-case bounds for LP.

Another advantage gained by ignorance is generality; in fact, the algorithm by Matoušek et. al. requires only very basic properties of LP in order to work, and these properties are shared by many other problems, including nonlinear and even nonconvex optimization problems, for which in some cases the best known bounds are obtained using the uniform algorithm for the general problem class [19, 21]. The abstract class of problems amenable to this approach has been termed *LP-type problems*.

Similarly, Kalai's subexponential algorithm works in the more general setting of so-called *abstract objective functions* (AOF), which he uses as a tool to derive extremely simple and beautiful proofs for known and new facts in LP and polytope theory [16]. While his algorithm is a primal simplex algorithm under the pivot rule RANDOM-FACET (which we discuss below), the algorithm of Matoušek et. al. is a dual version of it.

As it turns out, the concepts of LP-type problems and abstract objective functions are basically equivalent, and similar to still other attempts to generalize linear programming in connection with the simplex method [1, 8]. In a sense, these frameworks represent all properties of LP that have been found to be useful in combinatorial algorithms so far. No property of LP not present in these frameworks is known that can provably speed up the existing subexponential algorithms, or help in devising new ones with better runtimes. It is even possible that the subexponential bounds that have been established are a gross overestimate, and that the behavior is in fact polynomial on actual linear programs.

At least in the abstract setting, however, the subexponential analysis is tight. This has been shown by Matoušek [20] who constructed a class of LP-type prob-

lems (containing LP and non-LP instances) with $n = 2d$ constraints, on most of which the algorithm in [19] is indeed subexponentially slow. He failed, however, in proving an actual LP in the class to have this property, and the question remains open whether linear programs have some distinguishing feature which allows them to be handled faster by the algorithms in [15] and [19] (or any other combinatorial algorithm).

In this paper we show that such a distinguishing feature exists at least among the problems in the abstract 'worst-case' class constructed by Matoušek. In particular, after reformulating these problems as abstract objective functions (where they appear as generalized linear programs on a d-cube), we prove that Kalai's RANDOM-FACET algorithm handles the LP instances among them in $O(d^2)$ time. It follows that the 'slow' examples in the class must be non-LP instances. The result is obtained by characterizing a certain necessary condition for being an LP instance in terms of a simple combinatorial property.

Although Matoušek's class is only a small subclass of all AOF on the d-cube, this result is interesting in two respects. On the one hand it shows that even a completely ignorant pivot rule like RANDOM-FACET can implicitly "recognize" LP instances, which raises hopes that it might make such a distinction also on the general class of all AOF, ultimately leading to a strongly polynomial algorithm for LP.

On the other hand, we derive the first combinatorial property of LP that goes beyond the ones present in the abstract frameworks considered so far, and that can algorithmically be exploited in a nontrivial setting.

Note that as an isolated fact, the polynomiality of RANDOM-FACET on the LP instances in Matoušek's class is not remarkable; there are other (even very trivial) algorithms that achieve this for all problems in the class, as will become clear below. The interesting statement is that RANDOM-FACET is fast on the LP instances, *although* it is slow on other instances. This means, the LP instances are *provably easier* is this context; a similar statement in the general situation would be a major step forward.

Figure 1 summarizes the situation. The main challenge remains to replace the question mark in that figure by a meaningful bound. One way of achieving this could be to extract more useful combinatorics from the rich geometry of LP. In a specific situation, our result presents a first approach.

The paper is organized as follows. In Section 2 we introduce the concept of abstract objective functions on a polytope.

Section 3 turns to the special case where the polytope is a d-dimensional cube, and describes the simplex algorithm with Kalai's pivot rule RANDOM-FACET, specialized to that situation. We also derive the subexponential upper bound for the case of a cube, where it is simpler to obtain than for general polytopes.

Section 4 introduces Matoušek's class of AOF on the d-cube, along with a review of his lower bound proof. Finally, Section 5 contains our new polynomial analysis of RANDOM-FACET for the LP instances in Matoušek's class. A conclusion appears in Section 6.

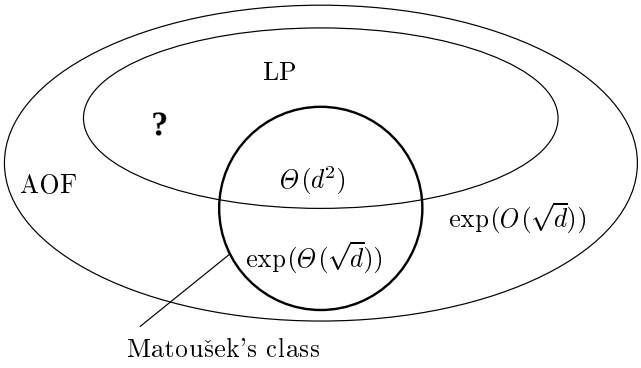


Fig. 1. Complexity of RANDOM-FACET, known bounds for $n = 2d$

2 Abstract Objective Functions

LP can be formulated as the problem of minimizing a linear objective function in d variables over a polyhedron, given as the intersection of n halfspaces in $\mathbb{R}^d$. Without loss of generality, one can assume that the polyhedron is simple and bounded, so that the *feasible region* of the LP is a simple polytope P, meaning that every vertex has exactly d incident edges. In this setting, the simplex method traverses the graph of vertices and edges of P – along a path of decreasing objective function value – until the traversal ends in a vertex of minimum value. A pivot rule decides at each intermediate vertex which decreasing edge will be followed, in case there is a choice. (See [5] for a thorough introduction to the simplex method.)

The fact that the traversal actually ends in a vertex of minimum value and not just in some local minimum is an obvious but important feature of LP. Moreover, this property holds for every face F of P because F is a polytope itself. This means, the simplex method can be used to find face minima, and this is a crucial substep of the RANDOM-FACET rule we introduce below.

If the pivot rule is *combinatorial*, it will classify edges incident to the current vertex only by the property of being increasing or decreasing with respect to the objective function. Geometric notions of "steepness" or "amount of progress" as frequently considered by pivot rules in practice are not taken into account.

In that situation, however, the geometric information that is actually used boils down to an acyclic orientation of the graph of P with the property that the subgraph induced by a face F has a unique sink, for every nonempty face F of P. If an orientation with these properties is obtained by assigning distinct 'abstract' objective function values $\phi(v)$ to the vertices v (with the meaning that edge $\{v, w\}$ is directed $v \to w$ if and only if $\phi(v) > \phi(w)$), we call ϕ an *abstract objective function* on P.

It is clear that every linear functional in general position w.r.t. P is an AOF, but there are more, as we will see soon. All simplex algorithms with combinatorial

pivot rules can be run on an AOF and will discover the vertex v of minimum value $\phi(v)$ in P.

It is worth noticing that AOF do not only arise in connection with the simplex algorithm; in fact, they have a surprizing interpretation as *shelling orders* of $P^{\triangle}$, the polytope dual to P [13].

3 The RANDOM-FACET simplex algorithm

An important special case (which we deal with exclusively in the sequel) arises when P is combinatorially equivalent to the d-dimensional cube C^d. In this case, the vertices can be identified with the set of 0/1-vectors $\mathcal{V} := \{0,1\}^d$, where it is convenient to view $\mathcal{V}$ as the vector space $GF(2)^d$ over the two-element field $GF(2)$. Two vertices v, v' are adjacent in the cube graph if and only if they differ in exactly one component, i.e. if $v - v' = \mathbf{e}_i$ for some unit vector $\mathbf{e}_i \in \mathcal{V}$.

The nonempty faces of C^d can be identified with pairs (v, S), $v \in \mathcal{V}$, $S \subseteq [d] := \{1, \ldots, d\}$, where

$$(v, S) \simeq \{v' \in \mathcal{V} \mid v_i = v'_i \ \forall i \notin S\}.$$

Thus, the face (v, S) consists of all vertices that agree with v outside of S. The dimension of (v, S) is $|S|$, and we have $(v, [d]) \simeq C^d$ and $(v, \emptyset) \simeq \{v\}$.

Now we are prepared to describe the algorithm RANDOM-FACET for an AOF ϕ on the d-cube. Its basic idea is as follows. Given some vertex v, choose a facet F containing v at random and recursively find the vertex v' of smallest value $\phi(v')$ in F. If $\phi(v') < \phi(v'')$, where v'' is the unique neighbor of v' not in F, then stop and return v'; otherwise, repeat from v''. In pseudocode, the algorithm can be described as follows (v is the current vertex and (v, S) denotes the face to be handled—initially, it is the whole cube).

Algorithm 1.

```
RANDOM-FACET(v, S):
  IF S = ∅ THEN
    RETURN v
  ELSE
    choose i ∈ S at random
    v' := RANDOM-FACET(v, S \ {i})
    IF φ(v') < φ(v' + e_i) THEN
      RETURN v'
    ELSE
      v'' := v' + e_i
      RETURN RANDOM-FACET(v'', S)
    END
  END
```

Because in the first recursive call the set S gets smaller, and in the second recursive call we have $\phi(v'') < \phi(v)$, the algorithm eventually terminates and returns the vector

$$\operatorname{opt}(v, S)$$

satisfying $\phi(\operatorname{opt}(v,S)) = \min(v,S)$, where we set $\min(v,S) := \min_{v' \in (v,S)} \phi(v')$. Here we need the AOF property that the local face minimum returned by the algorithm is actually a global face minimum.

The complexity of the algorithm can be estimated by counting the number of *pivot steps*, which is the number of times the operation $v'' := v' + \mathbf{e}_i$ is executed throughout the algorithm. It is easy to see that the overall number of operations is larger by a factor of at most $O(d)$. Here is a sketch of the subexponential bound on the expected number of pivot steps.

Consider a pair (v, S) and $i \in S$. We say that i is *fixed* w.r.t. (v, S) if $\phi(v) < \min(v + \mathbf{e}_i, S \setminus \{i\})$. This means, v is "better" than the best vertex in (v, S) that differs from v in the i-th position. Consequently, if we start the algorithm RANDOM-FACET on (v, S), the i-th position will never get flipped, because all vertices encountered throughout are at least as good as v itself.

The *hidden dimension* of (v, S) is then defined as

$$h(v,S) := |S| - |\{i \in S \mid i \text{ is fixed w.r.t. } (v,S)\}|.$$

The motivation for this definition is that although the face (v, S) has dimension $|S|$, the actual degree of freedom w.r.t. the algorithm RANDOM-FACET is only $h(v, S)$.

The following is the crucial fact: if $h(v, S) = k$ and the non-fixed indices in S are ordered such that

$$\min(v, S \setminus \{i_1\}) \geq \cdots \geq \min(v, S \setminus \{i_k\}),$$

then the hidden dimension of (v'', S) is at most $k - \ell$ if $i = i_\ell$ in the algorithm.

Now let $T(k)$ denote the maximum expected number of pivot steps that occur in a call to RANDOM-FACET(v, S), where (v, S) has hidden dimension k. From the preceding discussion one easily proves that $T(0) = 0$ and

$$T(k) \leq T(k-1) + \frac{1}{k} \sum_{\ell=1}^{k} (1 + T(k-\ell)),$$

for $k \geq 0$. From this, an upper bound of $T(d) \leq \exp(2\sqrt{d}) - 1$ for the expected number of pivot steps follows, see [7, Lemma 5.21] for a derivation of this bound.

4 Matoušek's Lower Bound Construction

Matoušek considers special AOF on the d-cube, defined by regular, lower-triangular matrices $A \in GF(2)^{d \times d}$. Such matrices have one-entries along the main diagonal and arbitrary entries below. For given A, an AOF ϕ is defined by

$$\phi(v) = Av \text{ for all } v \in \mathcal{V}, \tag{1}$$

where the values are compared by lexicographical order over $GF(2)^d$, i.e. $w < w'$ if $w_i = 0$ at the smallest index where $w_i \neq w'_i$. If we consider the values w as d-bit binary numbers $w_1 w_2 \ldots w_d$, we get an intuitive interpretation of this order as the usual order among natural numbers.

It follows that $v = 0$ is the optimal vertex of the whole cube C^d for every matrix A. To see that ϕ is indeed an AOF, we need the following

Lemma 1. *Let (v, S) be a face. We have $v = \mathrm{opt}(v, S)$ if and only if $(Av)_i = 0$ for all $i \in S$.*

Proof. Assume $v = \mathrm{opt}(v, S)$. Then v is in particular a local minimum in (v, S), i.e. $Av < A(v + \mathbf{e}_i) = Av + A_i$ for all $i \in S$, where A_i is the i-th column of A. Because A is lower-triangular with $a_{ii} = 1$, the first index where Av and $Av + A_i$ differ is i, and this implies $(Av)_i = 0$.

Now assume $(Av)_i = 0$ holds for all $i \in S$, and let v' be some other vertex in (v, S). Let $j \in S$ be the smallest index where v and v' differ. Again, the shape of A implies that j is then also the smallest index where Av and Av' differ. By assumption we must have $(Av')_j = 1$, so $\phi(v') > \phi(v)$. Because this holds for all v', $v = \mathrm{opt}(v, S)$ follows. □

The proof shows in particular that (v, S) contains exactly one local minimum, namely $\mathrm{opt}(v, S)$, and this proves the AOF property. The lemma also suggests the following alternative formulation of RANDOM-FACET, based on handling values $w = Av$ instead of vertices v.

Algorithm 2.

RF-FLIP(w, S):
```
IF S = ∅ THEN
    RETURN w
ELSE
    choose i ∈ S at random
    w' := RF-FLIP(w, S \ {i})
    IF w'_i = 0 THEN
        RETURN w'
    ELSE
        w'' := w' + A_i
        RETURN RF-FLIP(w'', S)
    END
END
```

It is obvious that RANDOM-FACET(v, S) is equivalent to RF-FLIP(Av, S). The latter, however, is more convenient in the following sketch of Matoušek's lower bound proof. An intuitive feature of RF-FLIP is that every "pivot step" $w'' := w' + A_i$ decreases the value currently maintained by the algorithm. Moreover, the actions of RF-FLIP(w, S) only depend on the $|S| \times |S|$-submatrix $A(S)$ of A consisting of the rows and columns with indices in S, and on the vector w_S, the restriction of w to S.

Let us first define a concept of *rank* for pairs (w, S), similar in spirit to the hidden dimension for pairs (v, S) in the previous section. If $S = \{i_1, \ldots, i_m\}$, $i_1 \leq \cdots \leq i_m$, and t is the smallest index such that $w_{i_t} = 1$, then we define

$$r(w, S) := |S| - t + 1.$$

If no such t exists, we set $r(w, S) = 0$. In other words, $r(w, S)$ is the number of significant bits in the binary number interpretation $w_{i_1} w_{i_2} \ldots w_{i_m}$ of w_S.

Let $T_S(k)$ denote the expected number of pivot steps in a call to RF-FLIP(w, S), where A is a random lower-triangular matrix with $a_{ii} = 1$ for all i, and w is a random vector with $r(w, S) \leq k$. This means, the expectation is over A, w and the choices of i in the algorithm. Because A and w are random, $T_S(k)$ only depends on the size of S. W.l.o.g. we can assume $|S| = k$, because insignificant bits in w_S remain insignificant throughout the algorithm and thus never lead to pivot steps. This means, we can write $T(k)$ instead of $T_S(k)$

The crucial observation is that if the second recursive call is executed at all (which happens with probability $1/2$, depending on the bit w_i of the start value w), then $r(w'', S) \leq k - \ell$ if $i = i_\ell$. This follows quite directly from Lemma 1. Moreover, w'' is of the form $w' + A_i$ and therefore random again, because A_i was random. It remains to observe that the actions of RF-FLIP(w'', S) do not depend on A_i anymore, so that w'' is independent of the (random) entries of the matrix that are still relevant. From this, one can prove that $T(0) = 0$ and

$$T(k) = T(k-1) + \frac{1}{2}\left(\sum_{\ell=1}^{k}(1 + T(k-\ell)\right),$$

for $k > 0$, see [20] for details. A subexponential lower bound of

$$T(d) = \Omega\left(\frac{1}{d}\exp(\sqrt{2d})\right) \tag{2}$$

for the expected number of pivot steps in the whole cube C^d follows, see [7, Result 6.10].

We note that this proof can be derandomized, i.e. we can construct a fixed matrix A and a fixed start value w such that RF-FLIP$(w, [d])$ is subexponentially slow (with worse constants than in (2), though). This construction will appear in the full paper.

5 A Polynomial Bound for the LP Instances

In this section we show that the subexponential lower bound developed in the previous section does not apply to the LP instances in Matoušek's class which we also call *realizable* instances below. We prove that RANDOM-FACET solves any realizable instance with an expected polynomial number of $O(d^2)$ pivot steps.

The proof consists of two stages. In the first stage we observe that if the AOF generated by a matrix A is realizable, then A does not contain certain

forbidden submatrices which come from 3-dimensional realizability restrictions. Subsequently we show that this imposes a rather strong condition on A^{-1} which already implies that most matrices A generate non-realizable instances.

In the second stage, we show how RANDOM-FACET exploits the structure of A^{-1} to arrive at a polynomial bound.

5.1 Three-dimensional Realizability Restrictions

An AOF ϕ on the d-cube C^d is realizable if and only if there exists a polytope P, combinatorially equivalent to the unit d-cube $[0,1]^d$ (we say that P is a *combinatorial d-cube* in this case), and a linear objective function $\alpha : \mathbb{R}^d \mapsto \mathbb{R}$, such that the orientation generated by α on the graph of P is isomorphic to the orientation generated by ϕ on C^d. Note that we cannot assume P to be *equal* to the unit cube in this definition. For example, any AOF ϕ on the 2-cube satisfying $\phi(0,0) < \phi(1,0) < \phi(1,1) < \phi(0,1)$ generates the orientation of Figure 2 (left), and it takes a "deformed" unit square to realize it, see Figure 2 (right).

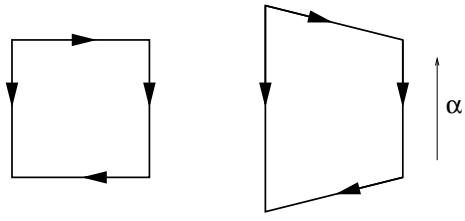


Fig. 2. Orientation not realized by unit cube

A necessary condition for realizability is of course that the directed subgraphs induced by ϕ on the 3-dimensional faces of C^d are realizable in this sense. The following lemma develops a condition for this in case of Matousek's AOF.

Lemma 2. *Assume the AOF generated by $A \in GF(2)^{d\times d}$ is realizable. Then for all $S \subseteq [d]$, $|S| = 3$, the submatrix $A(S)$ is not equal to*

$$A_1 := \begin{pmatrix} \mathbf{1} & 0 & 0 \\ \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \quad \textit{or} \quad A_2 := \begin{pmatrix} \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix},$$

.

Proof. It is easy to check from the definition (1) of Matoušek's AOF that A_1 and A_2 generate the C^3-orientations depicted in Figure 3.

In [10] it is shown that these orientations do not come from a linear objective function on a combinatorial 3-cube (they are actually the only ones on the 3-cube that don't, up to isomorphism). Another way to see this is due to Holt and

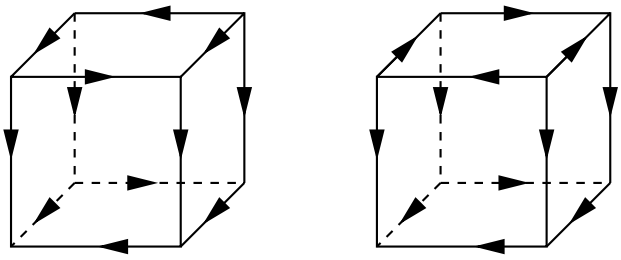


Fig. 3. Cube orientations generated by A_1 (left) and A_2 (right)

Klee who have shown—based on [3]—that a necessary condition for realizability is the existence of three vertex-disjoint directed paths between source and sink [14]. In either of the two orientations in Figure 3, only two such paths can be found.

From the interpretation of RANDOM-FACET as Algorithm 2 RF-FLIP it is clear that the orientation induced by A on a 3-face (v, S) is isomorphic to the orientation induced by $A(S)$ on C^3. Because the former is realizable by assumption, $A(S)$ must be distinct from A_1 and A_2. □

As it turns out, the forbidden submatrix conditions established by the Lemma are quite strong and manifest themselves directly in A^{-1} (which is again lower-triangular). The following is the crucial result.

Theorem 1. *If A does not contain submatrices $A(S) = A_1$ or $A(S) = A_2$, then A^{-1} has no more than two one-entries per row (including the diagonal one).*

This shows that among the $2^{\binom{d}{2}}$ matrices A, at most $d! \approx 2^{d \ln d - d}$ generate realizable instances.

Proof. Because A is invertible and lower-triangular, each column A_i can be written in the form

$$A_i = \mathbf{e}_i + \sum_{j \in J(i)} A_j, \tag{3}$$

where $J(i) \subseteq \{i+1, \ldots, d\}$ is a unique index set. We now prove two claims. For this let $A = (a_{ij}), 1 \leq i, j \leq d$, i.e. a_{ij} is the element in row i and column j.

Claim 1. For all i, the columns $A_j, j \in J(i)$ are disjoint in the sense that no two of them have a one-entry in the same row.

Proof (of Claim 1): assume on the contrary that $j_1 < j_2 \leq k$ exists such that $j_1, j_2 \in J(i)$ and $a_{k,j_1} = a_{k,j_2} = 1$. Suppose that (j_2, k) is lexicographically smallest with this property.

Case 1. $j_2 < k$. Then we have the situation of Figure 4 (left), and because (j_2, k) was lexicographically smallest with $a_{k,j_1} = a_{k,j_2} = 1$, we have $\star = a_{j_2,j_1} = 0$, which gives a forbidden submatrix.

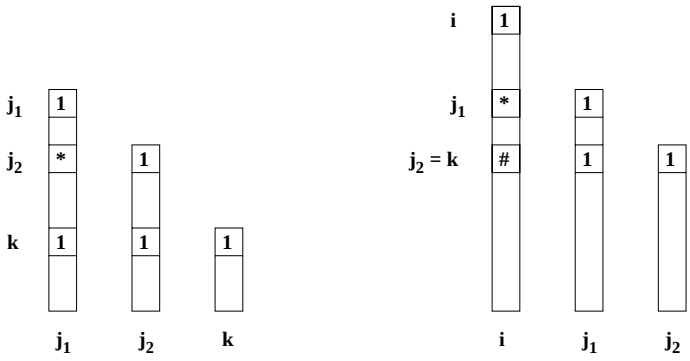


Fig. 4. Case $j_2 < k$ (left), $j_2 = k$ (right)

Case 2. $j_2 = k$. Then the situation is as in Figure 4 (right). Because in rows $j \in J(i), j < j_1$, position j_1 must be zero (otherwise we had a lexicographically smaller conflict again), it follows that $\star = a_{j_1,i} = 1$. For the same reason, in all columns $j \in J(i), j \neq j_1, j < j_2$, position k is zero. But this implies $\# = a_{k,i} = 0$, and we have a forbidden submatrix again

Claim 2. The index sets $J(i), i \in [d]$ are pairwise disjoint, i.e. in representing the columns according to (3), every column A_j is used for at most one other column A_i.

Proof (of Claim 2): assume a column A_j is used twice for distinct columns $A_{i_1}, A_{i_2}, i_1 < i_2 < j$. Because A_j is disjoint from all other columns used to represent A_{i_1} resp. A_{i_2}, we get $a_{j,i_1} = a_{j,i_2} = 1$, see Figure 5 (left). In order not to get a forbidden submatrix, we must have $\star = a_{i_2,i_1} = 1$.

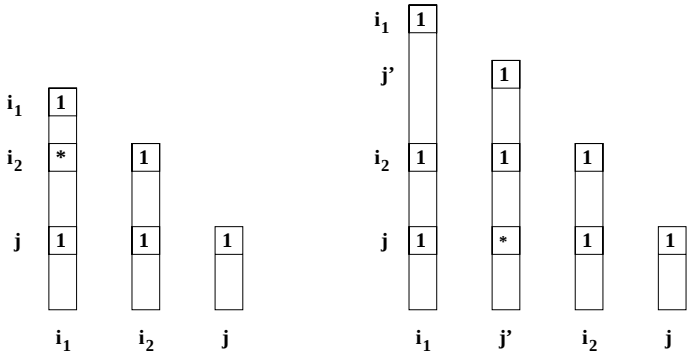


Fig. 5. A_j is used for A_{i_1} and A_{i_2} (left), $A_{j'}$ and A_j contribute to A_{i_1} (right)

Let $j' \in J(i_1)$ be the column 'responsible' for the one-entry a_{i_2,i_1}. We must have $j' \neq i_2$ because $A_{j'}$ is disjoint from A_j. Therefore, $j < i_2$, and the situation is that of Figure 5 (right).

Because both $A_{j'}$ and A_j contribute to A_{i_1}, they are disjoint, and $\star = a_{j,j'} = 0$ must hold. Then, however, we have a forbidden submatrix once more.

Now we are prepared to prove the statement of the Theorem, which follows if we can show that the columns of $M := E + A^{-1}$ are disjoint in the sense previously defined, where E is the unit matrix. The i-th column of M is given as

$$M_i = \mathbf{e}_i + A_i^{-1} = A^{-1}(A_i + \mathbf{e}_i) = A^{-1} \sum_{j \in J(i)} A_j = \sum_{j \in J(i)} \mathbf{e}_j.$$

The disjointness of the $J(i), i \in [d]$ now immediately implies the disjointness of the columns M_i. □

5.2 RANDOM-FACET under Realizability Restrictions

Theorem 1 implies that A^{-1} is extremely sparse in the realizable case, and this will entail that RANDOM-FACET is fast on the AOF induced by A. Before we go into the formal analysis, here is the intuitive argument why this is the case, and how the inverse matrix A^{-1} comes in. Consider starting the algorithm on the pair $(v, [d])$. With probability $1/d$, the first recursive call will compute $v' = \mathrm{opt}(v, [d] \setminus \{i\})$, for some $i \in [d]$. We know from Lemma 1 that $Av' = 0$ or $Av' = \mathbf{e}_i$, and only in the latter case, the algorithm performs a second recursive call, starting with the vertex $v'' = v' + \mathbf{e}_i$. It follows that

$$v'' = A^{-1}A(v' + \mathbf{e}_i) = A^{-1}(\mathbf{e}_i + A_i) = A_i^{-1} + \mathbf{e}_i.$$

This means, the possible v'' coming up in the second recursive call are nothing else than the pairwise disjoint columns of the matrix M considered in the proof of Theorem 1. Now, if that call fixes some position $j \in [d]$ in its first recursive step, it fixes a zero position with high probability, because $v''_j \neq 0$ for at most one i. This means that already after the first recursive call in RANDOM-FACET$(v'', [d])$, the problem has been optimally solved in $d - 1$ out of d cases, because fixing a zero means that the optimal vertex 0 lies in the facet that is being considered.

In the following formal derivation of the $O(d^2)$ bound, it will be more convenient to argue about the algorithm RF-FLIP instead. Let $T(w, S)$ be the expected number of pivot steps in a call to RF-FLIP(w, S), and define

$$w^{(i)} := \mathbf{e}_i + A_i, \text{ for all } i \in [d].$$

The following equation is the basis of the analysis.

Lemma 3. *Let $|S| = m > 0$ and define $\mathrm{opt}(w, S)$ to be the value returned by RF-FLIP(w, S). With $w^{(i,j)} := \mathrm{opt}(w^{(i)}, S \setminus \{j\})$ we have*

$$T(w^{(i)}, S) = \frac{1}{m} \sum_{j \in S} \Big(T(w^{(i)}, S \setminus \{j\}) + (1 + T(w^{(j)}, S))[w_j^{(i,j)} = 1] \Big). \qquad (4)$$

Here, $[\cdot]$ is the indicator variable for the event in brackets.

The proof follows immediately from the description of RF-FLIP in Algorithm 2, together with the following little observation: the value $w' = w^{(i,j)}$ computed from the first recursive call satisfies $w'_k = 0$ for all $k \in S \setminus \{j\}$ by Lemma 1. Exactly if $w'_j = 1$, we have a second recursive call with value $w'' = w' + A_j$. We do not necessarily have $w'' = w^{(j)}$ as suggested by the equation, but when we restrict both values to S, they agree. As previously observed, we then have $T(w'', S) = T(w^{(j)}, S)$.

Define

$$\overline{T}(S) = \sum_{i \in S} T(w^{(i)}, S).$$

Using (4) and the fact that $T(w^{(j)}, S \setminus \{j\}) = T(w^{(j)}, S)$ (the bit $w_j^{(j)}$ is insignificant and does not contribute any pivot steps in RF-FLIP$(w^{(i)}, S)$)), we obtain

$$\overline{T}(S) = \frac{1}{m-1} \sum_{j \in S} \left(\overline{T}(S \setminus \{j\}) + (1 + T(w^{(j)}, S)) \sum_{i \in S} [w_j^{(i,j)} = 1] \right), \quad m > 1. \tag{5}$$

The claim now is that $w_j^{(i,j)} = 1$ for at most one i. We have $w_j^{(j,j)} = 0$ ($w_j^{(j)}$ is an insignificant bit which therefore never gets flipped in a pivot step), and for $i \neq j$ we can argue as follows. By definition of $w^{(i,j)}$ we know that

(i) $w_k^{(i,j)} = 0$ for all $k \in S \setminus \{j\}$, and
(ii) $(A^{-1}(w^{(i)} - w^{(i,j)}))_k = 0$, for all $k \notin S \setminus \{j\}$.

The second condition holds because the vectors corresponding to the two values are in the same facet $(v, S \setminus \{j\})$. Condition (ii) implies $(A(S)^{-1}(w_S^{(i)} - w_S^{(i,j)}))_j = 0$. Using the definition of $w^{(i)}$ and property (i), one deduces that $w_j^{(i,j)}$ is equal to entry i in row j of $A(S)^{-1}$. By Theorem 1 (which also applies to $A(S)$), at most one of these entries is nonzero, and this proves the claim. Then, (5) implies

$$\overline{T}(S) \leq \frac{1}{m-2} \sum_{j \in S} \overline{T}(S \setminus \{j\}) + \frac{m}{m-2}, \quad m > 2.$$

If we let $\overline{T}(m) := \max_{|S|=m} \overline{T}(S)$ we get $\overline{T}(m) \leq (m/(m-2))(\overline{T}(m-1) + 1)$, for $m > 2$ and $\overline{T}(2) \leq 1$ (by directly inspecting the possible cases), from which we obtain

$$\overline{T}(m) \leq 3\binom{m}{2} - m, \quad m > 2. \tag{6}$$

To conclude the analysis we observe that

$$T(w, S) \leq \frac{1}{m} \sum_{j \in S} \left(T(w, S \setminus \{j\}) + 1 + T(w^{(j)}, S) \right),$$

for all start values w. With (6) and $T(m) := \max_{w, |S|=m} T(w, S)$ we get $T(0) = 0, T(1) = 1$ and

$$T(m) \leq T(m-1) + 1 + \frac{1}{m} \left(3\binom{m}{2} - m \right), \quad m > 2$$

from which

$$T(m) \leq \frac{3}{2}\binom{m}{2} + 2$$

follows. This gives the main result of the paper.

Theorem 2. *If the AOF defined by $A \in GF(2)^{d\times d}$ via (1) arises from a linear program on a combinatorial d-cube, then the expected number of pivot steps in Algorithm 1* RANDOM-FACET *is bounded by*

$$\frac{3}{2}\binom{d}{2} + 2 \approx \frac{3}{4}d^2,$$

for any start vertex v.

6 Conclusion

We have shown that the simplex algorithm RANDOM-FACET takes an expected $O(d^2)$ number of pivot steps on a subclass of Matoušek's AOF, characterized by a sparsity condition according to Theorem 1. This subclass contains at least all realizable instances, but we do not know whether it contains *only* realizable instances. On the other hand, there are (non-realizable) instances in the class where the expected number of steps is of the order $\Omega(\exp(\sqrt{2d})/d)$. This means, we have presented the first scenario in which the known combinatorial LP frameworks are provably weaker than LP itself.

Our $O(d^2)$ upper bound on the number of pivot steps is tight. Matoušek's class also contains an instance equivalent to the d-dimensional Klee-Minty cube (when all entries of A below the diagonal are one), and the behavior of RANDOM-FACET on this polytope can completely be analyzed: for some start vertices, the expected number of pivot steps is $\Theta(d^2)$ [20, 9].

The main open problem is to extend our result to AOF beyond Matoušek's class. For example, is RANDOM-FACET in fact polynomial on *all* realizable AOF on the d-cube? Even for this special polytope, nothing is known that goes beyond the subexponential bound of Section 3. It would also be interesting to find a realizable AOF that requires asymptotically more than $O(d^2)$ pivot steps. Namely, although there is no reason to believe that $O(d^2)$ is an upper bound in the general case, no ideas are currently known that may possibly lead to worse bounds. In this respect, the situation is quite similar to that of the algorithm RANDOM-EDGE; here, the best lower bound that is known for AOF on the d-cube is $\Omega(d^2/\log d)$, even when nonrealizable AOF are admitted [9]. (Incidentally, this bound is obtained for the d-dimensional Klee-Minty cube again). Breaking the d^2-barrier from below would therefore require substantially new classes of examples, for either of the two algorithms.

Acknowledgment

I would like to thank Jirka Matoušek, Falk Tschirschnitz, Emo Welzl and Günter Ziegler for many remarks that helped to improve the presentation.

References

1. I. Adler and R. Saigal. Long monotone paths in abstract polytopes. *Math. Operations Research*, 1(1):89–95, 1976.
2. N. Amenta and G. M. Ziegler. Deformed products and maximal shadows. In J. Chazelle, J.B. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational geometry*, Contemporary Mathematics. Amer. Math. Soc., 1998.
3. D. Barnette. A short proof of the d-connectedness of d-polytopes. *Discrete Math.*, 137:351–352, 1995.
4. R. G. Bland. New finite pivoting rules for the simplex method. *Math. Operations Research*, 2:103–107, 1977.
5. V. Chvátal. *Linear Programming*. W. H. Freeman, New York, NY, 1983.
6. G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.
7. B. Gärtner. *Randomized Optimization by Simplex-Type Methods*. PhD thesis, Freie Universität Berlin, 1995.
8. B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM J. Comput.*, 24:1018–1035, 1995.
9. B. Gärtner, M. Henk, and G. M. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica* (to appear).
10. B. Gärtner and V. Kaibel. Abstract objective function graphs on the 3-cube – a characterization by realizability. Technical Report TR 296, Dept. of Computer Science, ETH Zürich, 1998.
11. B. Gärtner and E. Welzl. Linear programming – randomization and abstract frameworks. In *Proc. 13th annu. Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 1046 of *Lecture Notes in Computer Science*, pages 669–687. Springer-Verlag, 1996.
12. D. Goldfarb. On the complexity of the simplex algorithm. In S. Gomez, editor, *IV. Workshop on Numerical Analysis and Optimization*, Oaxaca, Mexico, 1993.
13. K. Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discr. Applied Math.*, 20:69–81, 1988.
14. F. Holt and V. Klee. A proof of the strict monotone 4-step conjecture. In J. Chazelle, J.B. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational geometry*, Contemporary Mathematics. Amer. Math. Soc., 1998.
15. G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th annu. ACM Symp. on Theory of Computing.*, pages 475–482, 1992.
16. G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Math. Programming*, 79:217–233, 1997.
17. L. G. Khachiyan. Polynomial algorithms in linear programming. *U.S.S.R. Comput. Math. and Math. Phys*, 20:53–72, 1980.
18. V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.
19. J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.
20. J. Matoušek. Lower bounds for a subexponential optimization algorithm. *Random Structures & Algorithms*, 5(4):591–607, 1994.
21. M. Sharir and E. Welzl. Rectilinear and polygonal p-piercing and p-center problems. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 122–132, 1996.

A Note on Bounding the Mixing Time by Linear Programming

Abraham Sharell

Université Paris Sud, Laboratoire de Recherche en Informatique
F-91405 Orsay, France
sharell@lri.fr http://www.lri.fr/~sharell

Abstract. We relate the mixing time of Markov chains on a partial order with unique minimal and maximal elements to the solution of associated linear programs. The linear minimization program we construct has one variable per state and (the square of) its solution is an upper bound on the mixing time. The proof of this theorem uses the coupling technique and a generalization of the distance function commonly used in this context. Explicit solutions are obtained for the simple Markov chains on the hypercube and on the independent sets of a complete bipartite graph.
As an application we define new a Markov chain on the down-sets (ideals) of a partial order for which our technique yields a simple proof of rapid mixing, provided that in the Hasse-graph of the partial order the number of elements at distance at most 2 from any given element is bounded by 4. This chain is a variation of the Luby-Vigoda chain on independent sets, which can also be used directly to sample down-sets, but our result applies to a larger class of partial orders.

1 Introduction

When confronted with a large set of configurations (a *state space*), S, about which little can be deduced analytically, one might attempt to estimate some of its parameters by random sampling, i.e. by Monte Carlo methods. It is well known [5] that the ability to sample states at random from a distribution close to uniform (or some other weighting of the states w) is usually sufficient to estimate, the size of the state space (or its weighted analogue, the partition function, $Z := \sum_{x \in S} w(x)$).

1.1 The Markov chain Monte Carlo method

However even random sampling from S might be difficult. A general approach to overcome this difficulty, is to start from some initial state $X_0 = x \in S$ and then apply some *local* random transformation f for $N > 0$ steps, computing $X_{t+1} := f(X_t)$, for $0 \leq t < N$, to obtain a random state $X_N \in S$. The process $\{X_t\}$ is a discrete time Markov chain $\mathsf{MC} = \langle S, P \rangle$ on S with transition probabilities P satisfying $P(x, y) = \mathbb{Pr}[f(x) = y]$ for all $x, y \in S$. If this Markov chain is ergodic

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 97-115, 1998.

then the distribution of X_N, $P^N(x,\cdot)$, will converge to the chain's stationary distribution, say π, as N grows large, thus providing a sample X_N from π with small bias for sufficiently large N. This method is often called *Markov chain Monte Carlo* (MCMC), and its main difficulty is to define the random function f, or alternatively the chain MC, such that it is possible to derive an upper bound on the *mixing rate* of MC (the smallest t so that $P^t(x,\cdot)$ is close to π). Such a bound can the be used to fix an integer N that guarantees small bias. While considerable progress has been made to bound the mixing rate it remains a difficult task [2, 11, 7].

1.2 Overview of technique and results

Consider a Markov chain MC on a partially ordered set $\langle S, P, s_{\min}, s_{\max}\rangle$ that has unique minimal and maximal elements $s_{\min}$ and $s_{\max}$ respectively. Let us say that MC is *monotone* if there exists a monotone transition function f for it, (see Definitions 2 and 3). A *rank function* on the poset $\langle S, \sqsubseteq\rangle$ is a strictly monotone function $h : S \to \mathsf{N}$. In section 3 we define, for a given monotone Markov chain MC, a set of linear constrains on a rank function h on S, such that the mixing time of the chain is bounded essentially by $h(s_{\max})^2$ for any h satisfying the constrains (Theorem 2). This means that we can obtain a bound on the mixing rate as the solution to the linear minimization program:

$$\text{minimize } h(s_{\max}) \qquad \text{subject to } h \in \mathcal{H}_0 \ ,$$

where $\mathcal{H}_0$ is the polyhedra defined by the linear constrains on h. Basically we use a simple coupling for the Markov chain and a special kind of distance function (defined by a rank function) to bound the coupling time and therefore also the mixing rate (see Remark 1).

The large number of linear constrains makes it usually infeasible to solve the program automatically. For the simple case of Markov Chains on the n-dimensional hypercube and the independent sets of a complete bipartite graph, we find an explicit solution (see Example 1 and Section 4.5 respectively).

In sections 4 and 5 we apply our technique to obtain bounds on the mixing rate of Markov chains on the *down-sets* (or *ideals*) of a partial order. In each cases we prove that the rank function $h(x) = |x|$ (cardinality of x) satisfies the constrains for the respective Markov chain if the degree of the poset is appropriately bounded. The first chain is just the random walk on the lattice of the down-sets. We show that this chain mixes rapidly for degree-2 posets (see Definition 5). The second Markov chain is a variation of the chain in [8] on independent sets. For an element u in the poset let $d_2(u)$ be the number of elements at distance at most 2 from u in the Hasse graph [1] of the poset. Our chain mixes rapidly if $d_2(u) \leq 4$ for every element u (Theorem 6). As explained in Remark 5 Luby and Vigoda's chain can also be used directly, but our one can be shown to mix rapidly for a larger class of posets.

[1] The Hasse graph of the poset $\langle S, \sqsubseteq\rangle$ is the directed graph on S with an edge from x to y whenever $x \sqsubset y$ and there is no element z satisfying $x \sqsubset z \sqsubset y$.

Acknowledgements

The work presented in this paper is part of the author's Ph.d. thesis [10] prepared at the L.R.I. and I wish to thank Michael de Rougemont, my advisor, as well as Miklos Santha, Stephane Boucheron, Wenceslas Fernando de la Vega, Jean-Pierre Tilich, Claire Kenyon and Michael Luby for advice, support and time spent in listening and correcting me. Special thanks to Jean-Pierre for proof-reading this paper and pointing out shortcomings, some of which I tried to amend.

2 Preliminaries

Let $\mathsf{MC} = \langle S, P \rangle$ be an ergodic Markov chain on a finite set of states S. Ergodicity implies the existence of a *stationary* distribution π on S that satisfies $\lim_{t\to\infty} P^t(x,y) = \pi(y)$ for all $x, y \in S$. The *(total) variation distance* is usually used as metric on distributions p and q on S:

$$\|p - q\| := \frac{1}{2} \sum_{x \in S} |p(x) - q(x)| = \max\{|p(A) - q(A)| : A \subseteq S\} \ .$$

The time an ergodic Markov chains needs to be close to its stationary distribution π is called the *mixing rate*. For initial state $x \in S$ it is defined by

$$\tau_x(\epsilon) := \min\{t \geq 0 : \forall t' \geq t \ \|P^{t'}(x, \cdot) - \pi\| \leq \epsilon\}$$

and in general by

$$\tau(\epsilon) := \max\{\tau_x(\epsilon) : x \in S\} \ .$$

We can sample elements of S with a distribution ϵ-close in variation distance to π by starting at some arbitrary state of S and simulating transitions of MC for $t = \tau(\epsilon)$ steps. To bound the mixing rate one can sometimes use a *coupling*: the construction of a random process $\{(X_t, Y_t)\}$ on pair of states, such that $\{X_t\}$ and $\{Y_t\}$ seen by themselves, obey the same law as the original random process.

Definition 1 (Coupling). *A* coupling *for a Markov chain* $\mathsf{MC} = \langle S, P \rangle$ *is a Markov chain* $\{(X_t, Y_t)\}$ *on* $S \times S$*, such that the transition matrix* Q *of the coupling satisfies for all* $x, y \in S$*,*

$$\forall\, x' \in S : \ \sum_{y' \in S} Q(x, y, x', y') = P(x, x') \quad \textit{and}$$
$$\forall\, y' \in S : \ \sum_{x' \in S} Q(x, y, x', y') = P(y, y') \ .$$

Any coupling $\{(X_t, Y_t)\}$ for a finite and ergodic Markov chain will (with probability 1) eventually hit the diagonal $X_t = Y_t$ [3, p.274]. The time at which this happens is called the *coupling time* and it is a random variable that depends on the initial distribution of the process. Denote by $T^{x,y}$ the coupling time of

a coupling Q, when started from initial state $(X_0, Y_0) = (x, y)$ and define the *expected coupling time of* Q by

$$\hat{T} = \hat{T}_Q := \max \{ \mathbb{E} [T^{x,y}] : x, y \in S \} \ .$$

An upper bound on the mixing rate is given, for example, in [1]

$$r(\epsilon) \leq 2e\hat{T}_Q \, (1 - \ln \epsilon) \ , \tag{1}$$

valid for all $0 < \epsilon < 1$ (where ln is the natural logarithm, i.e. to base e).

2.1 Bounding the coupling time

To bound $\hat{T}$ [1, 7] use the following idea. Assume we have defined a real-valued and bounded distance function Φ on S. The random process $\Phi(X_t, Y_t)$ is real-valued and bounded. If in addition one can show that it decreases, in some probabilistic sense, then this should give us a bound on T^{X_0, Y_0}.

Remark 1. For a Markov chain on a partial order the derivation of bounds on the mixing rate can be simplified, as we explain now.

1. The coupling on MC can be constructed from *any* monotone transition function for MC, as explained below (this part is based mainly on [9]).
2. Instead of a distance function Φ, we use a rank function h; this induces a distance function on *comparable* states x, y by $\Phi(x, y) = \max\{h(x), h(y)\} - \min\{h(x), h(y)\}$.
3. Finally the decrease of the expected distance after one transition step is expressed by linear constrains on h.

Definition 2 (Transition function). *We say that f is a* transition function *for the Markov chain $\mathsf{MC} = \langle S, P \rangle$ if there is a finite set Ω so that $f : S \times \Omega \to S$ and for each $x, y \in S$*

$$\mathbb{P}\mathrm{r} [f(x, \omega) = y] = P(x, y) \ , \tag{2}$$

where ω is drawn from Ω uniformly at random.

Definition 3 (Monotone Markov chain). *A transition function f for MC is* monotone *if for all $x, y \in S$ and for all $\omega \in \Omega$: $x \sqsubseteq y \quad \Rightarrow \quad f(x, \omega) \sqsubseteq f(y, \omega)$. A Markov chain MC is called monotone if there exists a monotone transition function f for it.*

Sometimes we view f as a random function on S with $\mathbb{P}\mathrm{r} [f(x) = y] := |\{\omega \in \Omega : f(x, \omega) = y\}| / |\Omega|$.

Remark 2. Since for a given Markov chain, there might monotone and non-monotone transition functions, we assume, to avoid ambiguities, that any monotone Markov chain is specified by a monotone transition function.

A theorem in [9] states that for a monotone Markov chain MC, defined on a partially ordered set with unique minimal and maximal states, it suffices to consider couplings defined by a (monotone) transition function. Let Q^* be the coupling defined by

$$\begin{aligned}(X_0, Y_0) &:= (s_{\min}, s_{\max}) \\ (X_{t+1}, Y_{t+1}) &:= (f(X_t, \omega), f(Y_t, \omega)) \quad \text{for } t \geq 0\ ,\end{aligned} \tag{3}$$

where ω is taken with uniform probability from Ω and f is a monotone transition function for MC.

Theorem 1 (adapted from [9]). *Let T^* be the coupling-time of the coupling Q^* for* MC *defined above and let $\tau(\epsilon)$ be the mixing rate of* MC. *Let $n \in \mathsf{N}$ be the length of the longest chain in $\langle S, \sqsubseteq \rangle$. Then for $0 \leq \epsilon \leq 1/e$*

$$\frac{1}{8(1 + \log_{1/\epsilon} n)} \mathbb{E}[T^*] \leq r(\epsilon)$$

Together with (1) this implies that for this type of Markov chains a coupling can be constructed in a generic manner, if one is willing to settle for bounds that are sufficient to prove polynomial time bounds, but maybe not optimal. We will refer to the coupling defined in (3) as the *standard coupling* for MC (or more accurately, for f).

3 Main theorem

For what follows let S be a finite set partially ordered by $\sqsubseteq$ with unique minimal and maximal elements $s_{\min}$ and $s_{\max}$ and fix some ergodic Markov chain $\mathsf{MC} = \langle S, P \rangle$ on S. For a function $h : S \to [0, \infty)$ define the functions $\hat{h}$, $\Delta[h] : S \to \mathsf{R}$ by

$$\hat{h}(x) := \sum_{y \in S} P(x, y) h(y) \tag{4}$$

$$\Delta[h](x) := \hat{h}(x) - h(x) = \sum_{y \in S} P(x, y)(h(y) - h(x)) \tag{5}$$

for all $x \in S$. The functions $\hat{h}$ and $\Delta[h]$ denote the expected value and difference of h after one transition respectively and they can be written, in vector notation, as $\hat{h} = Ph$, and $\Delta[h] = -(I - P)h$. [2]

We express now by linear constrains the conditions on h to be a rank function with the additional property that the expected difference in rank between any two states decreases in expectation after one transition. Let $\mathcal{H}$ be the set of all functions $h : S \to [0, \infty)$ that satisfy for all $x \sqsubseteq y \in S$

1. $x \neq y \Rightarrow h(y) > h(x)$ and

[2] The expression $I - P$ is also known as the *Laplacian* of P

2. $\hat{h}(y) - \hat{h}(x) \geq h(y) - h(x)$ or equivalently $\Delta[h](y) \leq \Delta[h](x)$.

Lemma 1. *$\mathcal{H}$ satisfies the following closure properties. For any $h, h' \in \mathcal{H}$,*

1. *$h + h' \in \mathcal{H}$,*
2. *$\alpha h \in \mathcal{H}$ for any $\alpha > 0$,*
3. *$h + c\bar{1} \in \mathcal{H}$ for any real constant c,*

where $\bar{1}$ is the constant function: $\bar{1}(x) = 1$ for all $x \in S$.

Proof. Straightforward and omitted. □

The above lemma allows us to normalize $\mathcal{H}$. Denote $x \sqsubset\!\!\cdot\, y$ if $x \sqsubset y$ and for no $z \in S$ $x \sqsubset z \sqsubset y$ (that is $x \sqsubset\!\!\cdot\, y$ are the edges in the Hasse graph of the poset $\langle S, \sqsubseteq \rangle$). By transitivity 2 inequalities for every pair $x \sqsubset\!\!\cdot\, y$ are sufficient to express the constrains on h. Let $\mathcal{H}_0$ be the set of functions $h : S \to \mathsf{R}$ such that

H1 $h(s_{\min}) = 0$,
H2 for every pair $x \sqsubset\!\!\cdot\, y$ $\quad h(y) - h(x) \geq 1$,
H3 for every pair $x \sqsubset\!\!\cdot\, y$ $\quad \Delta[h](y) \leq \Delta[h](x)$.

Clearly $\mathcal{H}_0 \subseteq \mathcal{H}$ and the closure properties in the last lemma imply that if $\mathcal{H}$ is not empty than neither is $\mathcal{H}_0$.

Theorem 2 (Main theorem). *Let $\mathsf{MC} = \langle S, P \rangle$ be a monotone Markov chain on a poset with unique minimal and maximal element $\langle S, \sqsubseteq, s_{min}, s_{max} \rangle$ and monotone transition function f. For $h \in \mathcal{H}_0$ and $x \in S$ define the random variable $d(x) := h(f(x)) - h(y)$ and set*

$$V(h) := min\{\mathbb{E}\left[(d(y) - d(x))^2 | x, y\right] : x \sqsubseteq y \in S\} .$$

Then the coupling time T^* of the standard coupling Q^* (defined in 3) satisfies

$$\mathbb{E}[T^*] \leq h(s_{max})^2 / V(h) .$$

In particular for the solution h^* to the linear minimization program

$$\min h(s_{max}) \quad \textit{subject to} \quad h \in \mathcal{H}_0 ,$$

we have $\mathbb{E}[T^*] \leq h^{*2}/V(h)$.

Remark 3. Note that because of condition **H2** the convex space $\mathcal{H}_0$ is bounded from below. Conditions **H1**, **H2** together guarantee that any $h \in \mathcal{H}_0$ satisfies $h(x) \geq |x|$ = height of x in $\langle S, \sqsubseteq \rangle$.

For the proof we adapt the following theorem from [7].

Lemma 2 ([7]). *Let MC be a monotone Markov chain on a poset $\langle S, \sqsubseteq, s_{min}, s_{max} \rangle$ as above and let $\{(X_t, Y_t)\}$ be a coupling for MC with coupling time T. Let $B > 0$ and let $\Phi : S \times S \to [0, B]$ be a function on S satisfying $\Phi(X_t, Y_t) = 0$ iff $t = T$. Set $\Phi(t) := \Phi(Y_t, X_t)$, $\Delta\Phi(t) := \Phi(t+1) - \Phi(t)$. If for some $V > 0$ and all values of $t \geq 0$*

1. $\mathbb{E}[\Delta\Phi(t)|X_t, Y_t] \leq 0$ *and*
2. $\Phi(t) > 0 \Rightarrow \mathbb{E}\left[(\Delta\Phi(t))^2|X_t, Y_t\right] \geq V$

then the expected coupling time satisfies

$$\mathbb{E}[T] \leq \frac{\Phi(0)(2B - \Phi(0))}{V} \ .$$

Proof. Outline of proof of theorem 2. Let (X_t, Y_t) be the standard coupling. Since it is defined by a monotone transition function we have, $X_t \sqsubset Y_t$ for all $0 \leq t < T^*$. Any $h \in \mathcal{H}_0$ is strictly increasing, therefore $h(X_t) < h(Y_t)$ and equality holds only at $t = T^*$. This means that we can use $\Phi(x, y) = h(y) - h(x)$ for $x \sqsubseteq y$ in lemma 2.

A straightforward calculation, using the fact that Q^* is a coupling, shows that for any $x, y \in S$

$$\mathbb{E}[\Phi(f(x), f(y))|x, y] = \hat{h}(y) - \hat{h}(x) \ , \tag{6}$$

and since h satisfies condition **H2**, the first hypothesis of the lemma is satisfied. The definition of $V(h)$ guarantees the second hypothesis immediately. Finally, by **H1**, $\Phi(0) = B = h(s_{\max})$ and the upper bound in the lemma simplifies to $h(s_{\max})^2/V(h)$. □

Remark 4. In practice the quantity $V(h)$ can be lower-bounded without difficulty. For example we have (because Φ is integer valued)
$V \geq \min_{x,y} \Pr[\Phi(f(x), f(y)) \neq \Phi(x, y)|x, y]$.

A common situation for which we can derive an explicit bound for $V(h)$ is when the transitions of the Markov chain satisfy the following condition. Let $f : S \times \Omega \to S$ be the transition function for MC

Condition 4. Assume that for all $x, y \in S$ so that $x \sqsubset y$ there are $\omega_1, \omega_2 \in \Omega$ so that

$$\begin{aligned} x \sqsubset f(x, \omega_1) \quad &\text{and} \quad f(y, \omega_1) = y \\ x = f(x, \omega_2) \quad &\text{and} \quad f(y, \omega_2) \sqsubset y \end{aligned}$$

Theorem 3. *Let* MC*, f and T^* be as in theorem 2. If the transition function f of* MC *satisfies condition 4 then the coupling time T^* of the standard coupling for* MC *satisfies*

$$\mathbb{E}[T^*] \leq (h^*)^2 \cdot \frac{|\Omega|}{2} \ ,$$

where h^* is the value obtained from the linear minimization program defined by

$$\min h(s_{max}) \quad \textit{subject to} \quad h \in \mathcal{H}_0 \ .$$

Proof. (Outline) By theorem 2 all that is left to do is to bound $V(h)$. And the above condition implies that $V(h) \geq 2/|\Omega|$ for any $h \in \mathcal{H}_0$. □

In all the examples that follow the state space S is a distributive sub-lattice of 2^U under set-inclusion, for some finite set $U = [n] = \{1, \dots n\}$. Therefore we use the simpler notation with set operations, i.e. $\subseteq$, U, $\emptyset$ for $\sqsubseteq$, $s_{\max}$ and $s_{\min}$ respectively.

Example 1 (The hypercube). Maybe the easiest examples for the concepts defined in the previous sections is provided by the hypercube. Let $U = [n]$ and consider the Markov chain on the n-dimensional hypercube 2^U defined by

$$\Omega := \{(u, +), (u, -) : u \in U\}$$

and the transition function

$$\tau(X, (u, \sigma)) := \begin{cases} X \cup \{u\} : \text{if } \sigma = + \\ X \setminus \{u\} \ : \text{if } \sigma = - \ . \end{cases}$$

It is easy to check that this chain is ergodic with uniform stationary distribution. Let $h(X) := |X|$ and $\Phi(X, Y) := h(Y) - h(X)$ for $X, Y \in 2^U$ such that $X \subseteq Y$. Φ is just the Hamming-distance restricted to pairs $X \subseteq Y$. The function h satisfies clearly conditions **H1** and **H2**. To verify condition **H3** note that for $X \in 2^U$

$$\Delta[h](X) = \frac{1}{2n}(n - 2|X|) \ ,$$

which is a decreasing function on $\langle 2^U, \subseteq \rangle$.

Condition 4 is easily verified and we can apply theorem 3 to obtain a bound on the expected coupling time of the standard coupling

$$\mathbb{E}\,[T^*] \leq h(U)^2 \frac{2n}{2} = n^3 \ .$$

It can be shown that T^* is concentrated around $\theta(n \log n)$ ([1]) so our bound is quite bad. However in less symmetrical cases the theorem provides bounds that are comparable (and sometimes better) to those provided by other methods (if other methods work at all).

4 Down-sets of a partial order

In this and the following section we study Markov chains on the set $\mathcal{I}$ of down-sets (or *ideals*) of a poset $\langle U, \preceq \rangle$. [3] $\mathcal{I}$ is a distributive lattice and by a theorem of Birkhoff, we can identify it with a distributive sub-lattice of $\langle 2^U, \subseteq \rangle$. Note that the Hasse-graph of $\mathcal{I}$ (seen as a poset) is a subgraph of the $|U|$-dimensional hypercube and neighbouring down-sets are those which differ in exactly one element. All of our results involve (various forms of) the degree of the poset.

[3] The symbol $\preceq$ is used here to avoid confusion with the partial order $\sqsubset$ or $\subset$ on the states of the Markov chains.

Definition 5 (Degree(s) of a poset). *Let $\langle U, \preceq \rangle$ be a poset and denote by $\Gamma^+(u) = \{v \in U : u \prec\!\cdot\, v\}$ the elements covering u and by $\Gamma^-(u) = \{v \in U : v \prec\!\cdot\, u\}$ the ones covered by u. The* degree *of an element $u \in U$ is defined by*

$$d(u) := |\Gamma^+(u)| + |\Gamma^-(u)|$$

By the degree of a poset we mean the maximal degree of its elements: $max\{d(u) : u \in U\}$. More generally we let $d_i(u)$ denote the number of elements v such that there is a directed path from u to v with i edges in the Hasse-graph. So $d_1(u) = d(u)$ and $d_*(u)$ denotes the number of elements comparable to u. We shall only use d, d_2 and d_*.

4.1 The simple Markov chain on down-sets

For a down-set $X \in \mathcal{I}$ let X^- be the set of maximal elements in X and X^+ the set of minimal elements in $U - X$. It can be seen that

- if $u \notin X$ then $X \uplus \{u\}$ is a down-set iff $u \in X^+$ and
- if $u \in X$ then $X - \{u\}$ is a down-set iff $u \in X^-$.

The graph on $\mathcal{I}$ in which a down-set X is connected with the down-sets $X \oplus \{u\}, u \in X^- \cup X^+$, is, as can easily be verified, connected (it is the Hasse graph of the poset $\langle \mathcal{I}, \subseteq \rangle$).

The simplest Markov chain on the down-sets $\mathcal{I}$ of a poset $\langle U, \preceq \rangle$ is just the random walk on this Hasse graph (with added self-loops to guarantee aperiodicity). While this chain mixes rapidly only for degree-2 posets, it serves us to illustrate the methods developed. We define a Markov chain $\mathsf{MC}_0 = \langle \mathcal{I}, P \rangle$ by the transition function $f : \mathcal{I} \times \Omega \to \mathcal{I}$ where $\Omega := U \times \{+, -\}$ and for $X \in \mathcal{I}$, $(u, \sigma) \in \Omega$

$$f(X, (u, \sigma)) := X \oplus \{u\} \text{ if } u \in X^\sigma \text{ else } X \ . \tag{7}$$

This implies the transition probabilities

$$P(X, Y) = \begin{cases} 1/2n & : |X \oplus Y| = 1 \\ 0 & : |X \oplus Y| > 1 \\ 1 - \frac{|X^+| + |Y^-|}{2n} & : X = Y \end{cases} .$$

It is not difficult to check that the chain is ergodic with uniform stationary distribution.

4.2 The *'Barrier'* functions - a tool

Before we continue let us define the following set-valued functions. For a down-set $X \in \mathcal{I}$ and $i \in \mathsf{N}$ define

$$b_i^-(X) := \{u \in X : |\{v \in X : u \prec v\}| = i\}$$

and dually

$$b_i^+(X) := \{u \in U - X : |\{v \in U - X : v \prec u\}| = i\} .$$

Note that $X^+ = b_0^+(X)$ and $X^- = b_0^-(X)$. If $u \in b_i^+(X)$ then there are i elements that have to be added to X before we can add u. This i elements constitute a kind of *barrier* for the movement of the Markov chain in the direction of the coordinate u. Similarly if $u \in b_i^-(X)$ then there are i elements to be removed from X before we are allowed to remove u. For another interpretation define for $X \in \mathcal{I}$ and $k \in \mathsf{N}$

$$\begin{aligned} \mathcal{B}^+(X,0) &= X \\ \mathcal{B}^+(X,k) &:= X \uplus b_0^+(X) \uplus \cdots \uplus b_{k-1}^+(X) \end{aligned}$$

and dually

$$\begin{aligned} \mathcal{B}^-(X,0) &:= U - X \\ \mathcal{B}^-(X,k) &:= (U - X) \uplus b_0^-(X) \uplus \cdots\cdots \uplus b_{k-1}^-(X) . \end{aligned}$$

Lemma 3. *Let $\langle U, \preceq \rangle$ be a poset with down-sets $\mathcal{I}$. For $i \in \mathsf{N}$ let $\mathcal{B}_i^+, \mathcal{B}_i^-$ be as defined above. Let $X, Y \in \mathcal{I}$ so that $X \subseteq Y$. Then for all $k \in \mathsf{N}$*

$$\mathcal{B}^+(X,k) \subseteq \mathcal{B}^+(Y,k) \quad \textit{and} \quad \mathcal{B}^-(Y,k) \subseteq \mathcal{B}^-(X,k) .$$

Proof. Observe that for any down-set $I \in \mathcal{I}$ and $k \in \mathsf{N}^+$ the set $\mathcal{B}^+(I,k)$ includes all elements u that are in I or that can be added to I by adding before u at most $k-1$ other elements to I. This implies that $\mathcal{B}^+(I,k)$ is monotone growing in I and $\mathcal{B}^+(X,k) \subseteq \mathcal{B}^+(Y,k)$.

The inclusion for $\mathcal{B}^-$ is just the dual of the one for $\mathcal{B}^+$ and can be obtained from it by 'inverting' $\mathcal{I}$. □

4.3 Coupling time of $\mathsf{MC_0}$

Before we can apply theorem 3 we have to check that f is monotone.

Lemma 4. *Let $X, Y \in \mathcal{I}$ and Ω as defined above. For every $\omega \in \Omega$, if $X \subseteq Y$ then $f(X,\omega) \subseteq f(Y,\omega)$. Therefore f is a monotone transition function and MC_0 a monotone Markov chain.*

Proof. Let $X, Y \in \mathcal{I}$ such that $X \subseteq Y$. Consider first an 'adding' move $\omega = (u, +)$ for some $u \in U$. This move will change X and add u to it iff $u \in X^+$, i.e., if all $v \prec u$ are included in X, and therefore also in Y since $X \subseteq Y$. If $u \in X^+$ then $u \in Y \uplus Y^+$ (by lemma 3 with $k = 1$). So if we can add u to X then either $u \in Y$ or we can add u also to Y.

The analogous argument works for a 'subtracting' move since $Y^- \subseteq (U - X) \uplus X^-$. □

4.4 Rapid mixing for degree-2 posets

Using the rank function $h : X \to |X|$ we show now that for degree-2 posets the Markov chain MC_0 defined above mixes rapidly.

Lemma 5. *Let $h : \mathcal{I} \to \mathsf{R}$ be defined by $h(X) := |X|$ for all $X \in \mathcal{I}$. Let $X, Y \in \mathcal{I}$ with $X \subseteq Y$, and Ω as defined above. Then*

$$\Delta[h](X) = \frac{1}{|\Omega|}(|X^+| - |X^-|) \ .$$

Proof. For this specific function h we can easily evaluate $\Delta[h]$:

$$\begin{aligned} |\Omega| \cdot \Delta[h](X) &= \sum_{\omega \in \Omega} (h(f(X,\omega)) - h(X)) \\ &= \sum_{u \in X^+} (|X \uplus \{u\}| - |X|) + \sum_{u \in X^-} (|X - \{u\}| - |X|) \\ &= |X^+| - |X^-| \ . \end{aligned}$$

□

Lemma 6. *Let $h(X) := |X|$ for all $X \in \mathcal{I}$ and let $X, Y \in \mathcal{I}$ with $Y = X \uplus \{w\}$ for some $w \in X^+$. Then*

$$\Delta[h](Y) - \Delta[h](X) \leq \frac{d(w) - 2}{|\Omega|} \ ,$$

where $d(w)$ is the degree of w in the poset $\langle U, \preceq \rangle$.

Proof. Let us look at the difference between X^+ and Y^+. On the one hand $w \in X^+ \setminus Y^+$ and this is the only element in $X^+ \setminus Y^+$. On the other hand $v \in Y^+ \setminus X^+$ iff $v \in Y^+ = (X \uplus \{w\})^+$ and w is the only element outside X so that $w \prec\!\cdot\, v$, i.e., iff

$$v \in \Gamma^+(w) \cap b_1^+(X) \ .$$

This gives

$$|Y^+| - |X^+| = -1 + |\Gamma^+(w) \cap b_1^+(X)| \ . \tag{8}$$

The difference between X^- and Y^- can be expressed similarly. The only element in $Y^- \setminus X^-$ is w. An element $v \in X^- \setminus Y^-$ iff $v \in X^-$ and $v \prec w$, therefore

$$|X^-| - |Y^-| = -1 + |\Gamma^-(w) \cap b_0^-(X)| \ . \tag{9}$$

Apply lemma 5 and add equation 8 with equation 9 to derive:

$$\begin{aligned} &|\Omega| \cdot (\Delta[h](Y) - \Delta[h](X)) = \\ &= (|Y^+| - |Y^-|) - (|X^+| - |X^-|) \\ &= |Y^+| - |X^+| + |X^-| - |Y^-| \\ &= -2 + |\Gamma^+(w) \cap b_1^+(X)| + |\Gamma^-(w) \cap b_0^-(X)| \\ &\leq -2 + |\Gamma^+(w)| + |\Gamma^-(w)| \\ &= -2 + d(w) \ . \end{aligned}$$

□

If the degree of the poset is bounded by 2 then the above lemma implies that $\Delta[h]$ is a decreasing function on the poset $\langle \mathcal{I}, \subseteq \rangle$. This is equivalent to the condition **H3** for MC_0. The result of this section is summarized in the following theorem.

Theorem 4. *If the degree of $\langle U, \preceq \rangle$ is bounded by 2 then the coupling time T^* of the standard coupling of MC_0 satisfies $\mathbb{E}[T^*] \leq n^3$, where $n = |U|$.*

Proof. The Markov chain MC_0 is monotone (lemma 4) and satisfies condition 4 (we have omitted the proof of this). Therefore we can apply theorem 3 if h is in the set $\mathcal{H}_0$ for MC. It is easy to check that the function $h : X \to |X|$ satisfies the conditions **H1** and **H2** for MC_0. The last condition **H3** is equivalent to the function $\Delta[h]$ being a decreasing function on $\mathcal{I}$ which is verified by lemma 6 for the case of a degree-2 poset. By theorem 3

$$\mathbb{E}[T^*] \leq h(U)^2 \frac{|\Omega|}{2} = n^2 \frac{2n}{2} = n^3 \ .$$

□

4.5 $K^{m,m}$ as partial order

The complete bipartite graph on $2m$ elements $\langle U = V \uplus W, V \times W \rangle$ can be seen as a partial order with V the minimal elements and W the maximal elements. As we shall see, the solution for the conditions $\mathcal{H}_0$ is very far from the simple function h used in the previous section. The Hasse graph G of the down-sets of this partial order consists of the hypercube 2^V and the 'translated' hypercube $\{C \uplus V : C \in 2^W\}$, connected at the down-set V. This graph has an extreme bottleneck at V: take $S' := 2^V - \{V\}$. There are only m edges out of S' and therefore G's expansion is (at most) $m/|S'| = m/(2^m - 1)$.

Theorem 5. *There is a function $h \in \mathcal{H}_0$ for the Markov chain MC_0 on the down-sets of the partial order, defined by the complete bipartite graph on $2m$ elements, that satisfies*

$$h(U) = m + 2 \sum_{i=0}^{m-1-k} \frac{k \cdots (k+i)}{(m-k) \cdots (m-k-i)} = \mathcal{O}(2^m)$$

where $k = \lceil m/2 \rceil$.

This gives a bound on the coupling time of $\mathcal{O}(2^m)$, which is of the correct magnitude.

Proof. Since $\mathcal{I}$ is completely symmetric we look for a solution $h(X)$ that depends only on $|X|$. For $i \in \{0, \ldots 2m\}$ write h_i and $\hat{h}_i$ respectively for the value of h

and $\hat{h}$ on a down-set X with i elements. Then, using the fact that $\mathcal{I}$ consists of 2 hypercubes joined at level $|V| = m$ we derive for $0 \le i \le 2m$

$$|\Omega|(\hat{h}_i - h_i) = \begin{cases} (m-i)(h_{i+1} - h_i) - i(h_i - h_{i-1}) \\ m(h_{i+1} - h_i) - m(h_i - h_{i-1}) \\ (2m-i)(h_{i+1} - h_i) - (i-m)(h_i - h_{i-1}) \end{cases}$$
$$= \begin{cases} (m-i)\delta_i - i\delta_{i-1} & : i \le m-1 \\ m\delta_m - m\delta_{m-1} & : i = m \\ (2m-i)\delta_i - (i-m)\delta_{i-1} & : m+1 \le i \end{cases}$$

where $\delta_i := h_{i+1} - h_i$ for $i \in \{0, \ldots, 2m-1\}$. Set $k := \lceil m/2 \rceil$. The solution we give satisfies

$$|\Omega|(\hat{h}_i - h_i) = \begin{cases} m - 2i & : i \le k-1 \\ 0 & : k \le i \le 2m-k \\ 3m - 2i & : 2m-k < i \ . \end{cases}$$

For $X \in \mathcal{I}$ with $|X| = i$ we have $\Delta[h](X) = \hat{h}(X) - h(X) = \hat{h}_i - h_i$ which decreases with i. Therefore $\Delta[h]$ is a decreasing function on the lattice and satisfies condition **H3**.

Solving 'backwards' we find that $\delta_i = 1$ for $i \in \{0, \ldots, k-1\} \cup \{2m-k, \ldots, 2m-1\}$. For the range $i \in \{k, \ldots, m-1\}$ we have the recursion $\delta_i = \frac{i}{m-i}\delta_{i-1}$. The next $m-k$ values, for $i \in \{m, \ldots 2m-k-1\}$, are the same, but in reverse order, until δ has decreased back to 1. Note that for all $0 \le i \le 2m-1$ we have $\delta_i \ge 1$ so that the solution satisfies condition **H2**. Finally define $h_0 := 0$ to satisfy condition **H1**.

For the $\mathcal{O}(2^m)$ bound it suffices to consider the last term in the sum, since it is the largest

$$\frac{k \cdots (m-1)}{(m-k) \cdots 1} = \frac{(m-1)!}{(m-k)!(k-1)!} = \binom{m-1}{k-1} = \mathcal{O}\big(m^{-1/2}\, 2^m\big) \ .$$

□

5 A variation on Luby and Vigoda's Markov chain

Let G be an undirected graph with maximal degree Δ. In [8] Luby and Vigoda define a Markov chain on the independent sets of G. This chain generates an independent set A with probability proportional to $\lambda^{|A|}$ for $\lambda > 0$ and mixes rapidly if $\lambda \le 1/(\Delta - 3)$. In the uniform case ($\lambda = 1$) this chain mixes rapidly when the degree of G is bounded by 4.

Remark 5. For a poset $\langle U, \preceq \rangle$ consider its *comparability* graph, which has vertex-set U and an edge for any pair u, v such that either $u \prec v$ or $v \prec u$. There is an easy to see bijection between the down-sets of $\preceq$ and the independent sets of the comparability graph: just observe that the set of maximal elements of X, X^-, defines X unambiguously and that X^- is an antichain for $\preceq$ and therefore

an independent set in the comparability graph. This bijection allows us to apply Luby and Vigoda's chain to sample down-sets efficiently, provided the degree (d_*) of the comparability graph is bounded by 4.

We can slightly improve on this by using a chain with *less* transitions: the transitions of the Luby-Vigoda chain are based on choosing a pair of comparable elements $u \prec v$, whereas our chain chooses only pairs of elements at distant at most 2 from each other; as we shall see the resulting chain mixes rapidly when $d_2(u) \leq 4$ for all $u \in U$. For posets with 4 or more levels, d_2 might still be bounded by 4, whereas the degree of the comparability graph is necessarily greater than 4 (see figure 1).

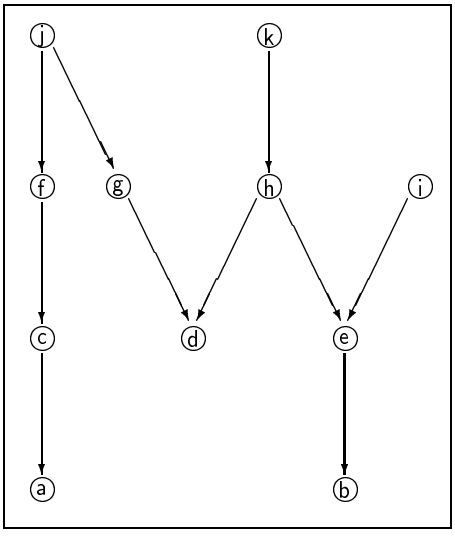


Fig. 1. A poset demonstrating the difference between d_* and d_2. In this poset $d_*(j) = 5$, but for the more local degree d_2 we have $d_2(j) = 4$

5.1 Definition of the Markov chain

Let $\langle U, \preceq \rangle$ be a poset with down-sets $\mathcal{I}$ and define the following binary relation on U:

$$u \prec_2 v \quad \text{iff} \quad u \prec\cdot v \text{ or } \exists w \in U : u \prec\cdot w \prec\cdot v$$

The random choice is to select a pair $u \prec_2 v$ and then to add or remove one or both of u, v. To be precise define

$$\Omega := \{(u, v), (u, \bar{v}), (\bar{u}, \bar{v}) : u \prec_2 v\}$$

and the auxiliary function $f' : \mathcal{I} \times \Omega \to 2^U$

$$\begin{aligned} f'(X,(u,v)) &:= X \cup \{u,v\} \\ f'(X,(u,\bar{v})) &:= X \cup \{u\} \setminus \{v\} \\ f'(X,(\bar{u},\bar{v})) &:= X \setminus \{u,v\} \ . \end{aligned}$$

The possibility to remove u and add v is omitted because it can never result in a down-set. The transition function and probabilities of the Markov chain $\mathsf{MC}_1 := \langle \mathcal{I}, P \rangle$ are defined by

$$f(X,\omega) := \begin{cases} f'(X,\omega) & : \text{if } f'(X,\omega) \in \mathcal{I} \\ X & : \text{otherwise} \ . \end{cases} \tag{10}$$

and

$$P(X,Y) := \mathbb{P}\mathsf{r}_\Omega \left[f(X,\omega) = Y \right] \ .$$

This definition is only interesting if the poset is not empty, i.e., there are $u, v \in U$ such that $u \prec_2 v$. However we can assume w.l.o.g. that the Hasse graph $\langle U, \prec \cdot \rangle$ is (weakly) connected; otherwise we can treat each component by itself. It is not difficult to check that this chain is ergodic with uniform stationary distribution. We omit this.

5.2 Bounding the coupling time

Define now the function $h : \mathcal{I} \to \mathsf{R}$ by

$$h(X) := |X| \tag{11}$$

for all $X \in \mathcal{I}$. We use theorem 3 to bound the coupling time of MC_1. To be specific there are three tasks: (a) check that the transition function f of MC_1 is monotone; (b) verify that MC_1 satisfies condition 4 and (c) show that $\Delta[h]$ is a decreasing function on $\langle \mathcal{I}, \subseteq \rangle$ (condition **H3**). This suffices since h satisfies conditions **H1** and **H2** independently of the Markov chain. Only the last of this three calculations is of interest here and we leave the other two for the reader. First we derive an expression for $\Delta[h]$.

Lemma 7. *Let $\langle U, \preceq \rangle$ be a poset and let h, and Ω be as defined above. For every down-set X of $\preceq$*

$$|\Omega|\Delta[h](X) = \sum_{u \in X^+} d_2(u) \ + 2|b_1^+(X)| - \sum_{u \in X^-} d_2(u) \ - 2|b_1^-(X)| \tag{12}$$

Proof. Fix $u \in X^+$. For every $v \prec_2 u$ the move (v,u) contributes $1/|\Omega|$ since necessarily v is already in X. Similarly for every w so that $u \prec_2 w$ the move $(u,\bar{w})$ contributes $1/|\Omega|$, since $w \notin X$. It should be clear that these are the only moves that contribute $1/|\Omega|$ and therefore each $u \in X^+$ contributes $d_2(u)/|\Omega|$.

The only possibility for a move (u, v) to contribute $2/|\Omega|$ is when $u \in X^+$ and $v \in (X \uplus \{u\})^+$. Note that

$$\left\{ \begin{array}{c} u \in X^+ \\ v \in (X \uplus \{u\})^+ \end{array} \right\} \Longleftrightarrow v \in b_1^+(X) \ ,$$

because $v \in b_1^+(X)$ means that there is an unique element, i.e., u, outside X so that $u \prec v$. Moreover this element u is in X^+, else there would be another element $w \notin X$ so that $w \prec\cdot\, u \prec\cdot\, v$, which contradicts $v \in b_1^+(X)$. Therefore each $v \in b_1^+(X)$ contributes exactly $2/|\Omega|$. The negative contributions can be similarly justified. □

Lemma 8. *Let $\langle U, \mathcal{I} \rangle$ be a poset and let Ω be as defined above. Let h be defined by equation 11. Let $X, Y \in \mathcal{I}(\preceq)$ such that $Y = X \uplus \{w\}$ for some $w \in X^+$. If $d_2(u) \leq 4$ for all $u \in U$ then*

$$\Delta[h](Y) \leq \Delta[h](X) \ .$$

Proof. By (12) the only elements that contribute to $\Delta[h](X)$ and $\Delta[h](Y)$ are those in $b_i^+(X)$, $b_i^-(X)$, $b_i^+(Y)$, and $b_i^-(Y)$ for $i = 0, 1$ (remember that $b_0^+(X) = X^+ and b_0^-(X) = X^-$). Therefore the only elements that may contribute a non-zero quantity to the difference $\Delta[h](Y) - \Delta[h](X)$ are those that are not in the same b_i^+ or b_i^- – set for Y and for X. Since the only difference between Y and X is the element w all those elements are comparable to w — as we shall see they have to be at distance at most 2 from w. Let

$$K := |\Omega| \cdot (\Delta[h](Y) - \Delta[h](X)) \ .$$

For $i > 0$ an element $u \in b_i^+(X)$ $i > 0$ moves to $b_{i-1}^+(Y)$ iff $w \prec u$. To calculate the contributions define for $i > 0$

$$D_i^+ := b_i^+(X) \cap \{u \in U : w \prec u\} \ .$$

Case $\mathbf{i = 2}$: An element u in D_2^+ moves from $b_2^+(X)$ to $b_1^+(Y)$ and therefore, by 12, contributes 2 to K.
Case $\mathbf{i = 1}$: $u \in D_1^+$ moves from $b_1^+(X)$ to $b_0^+(Y)$ and contributes $d_2(u) - 2$ to K (2 to the X part and $d_2(u)$ to the Y part).
Case $\mathbf{i = 0}$: observe that the only element that moves from X^+ to Y^- is w itself. Inspecting expression 12 we see that this contributes $-2d_2(w)$ to K.

Now set, for $i \geq 0$,

$$D_i^- := b_i^-(X) \cap \{u \in U : u \prec w\} \ .$$

and proceed similarly: an element $u \in D_0^-$ moves from $X^- = b_0^-(X)$ to $b_1^-(Y)$ and contributes accordingly $d_2(u) - 2$ to K. An element $u \in D_1^-$ moves from b_1^- to $b_2^-(Y)$ and contributes 2 to K. Putting all this together gives

$$K = 2|D_2^+| + \sum_{u \in D_1^+} (d_2(u) - 2) - 2d_2(w) + \sum_{u \in D_0^-} (d_2(u) - 2) + 2|D_1^-| \ .$$

By assumption $d_2(u) \leq 4$ for any $u \in U$ therefore we can bound all $(d_2(u) - 2)$ terms above by 2. This gives

$$K \leq 2\left(|D_2^+| + |D_1^+| + |D_0^-| + |D_1^-| - d_2(w)\right) \ . \tag{13}$$

The barrier-sets $b_i^+(X)$ and b_i^- partition U and are all disjoint. Therefore the sets D_i^+, D_i^- are also disjoint. Moreover it is not difficult to see that all elements in D_1^+ and in D_0^- are exactly at distant 1 from w. Finally observe that the elements in D_2^+ and D_1^- are at most at distant 2 from w. To see this assume by contradictions that $u \in D_2^+$ (for example) is at distance > 2 from w. This means that there are v, v' so that $w \prec v \prec v' \prec u$. But w, v and v' are all outside X, which contradicts $u \in b_2^+(X)$.

The four sets contributing positively in 13 are disjoint subsets of the elements counted in $d_2(w)$ and therefore $K \leq 0$ as desired. □

It remains only to collect everything into

Theorem 6. *Let $\langle U, \preceq \rangle$ be a poset on $n := |U|$ elements. If for all $u \in U$ we have $d_2(u) \leq 4$ then the coupling time T^* of the standard coupling for MC_1 satisfies*

$$\mathbb{E}[T^*] \leq 6n^3 \ ,$$

and therefore MC_1 is rapidly mixing.

Proof. (Outline) It can be verified that the transition function f is indeed monotone and that it satisfies condition 4. The function $h : X \mapsto |X|$ satisfies always the conditions **H1** and **H2** and by lemma 8, given the hypothesis of the theorem on d_2, also the condition **H3** for MC_1. Theorem 3 gives the bound

$$\mathbb{E}[T^*] \leq h(U)^2 \frac{|\Omega|}{2} = n^2 \frac{|\Omega|}{2} \leq n^2 \frac{12n}{2} = 6n^3 \ ,$$

since $|\Omega| \leq 3 \cdot 4n$, because $d_2(u) \leq 4$ for all $u \in U$. □

6 Conclusions and further research

We have shown how to construct a simple but large linear program which solution provides an upper bound on the mixing rate of a Markov chain. While the size of the program makes direct solution infeasible, we hope that further research might yield insights into the associated polyhedra and therefore also the behavior of the Markov chain. Since the underlying graph of most interesting Markov chains is small, so are the number of variables occurring in each constraint and therefore our system is sparse in this sense.

Another reason why a brute force attack on the linear program we define is not interesting, is given by the fact that for the Markov chains we study a Monte Carlo experiment allows one to estimate the coupling time T^*: simply simulate a coupling from the minimal and maximal state until they meet (see [9]). Repeating this one can obtain an estimate of the expected coupling time with

high confidence (say 99.999%). The time needed is polynomial in the expected coupling time itself. Therefore if one has reason to hope that the chain mixes (more or less) rapidly, then this approach is clearly better than to solving an exponentially large system of inequalities. However, if the chain does not mix rapidly enough, this approach becomes much less promising.

As we have seen solutions to the linear constrains can be derived directly for some simple examples and this solutions seem to have the correct order of magnitude. For the chains on down-sets one can not expect much better bounds and for the chain on the independent sets (or equivalently down-sets) of $K^{m,m}$ the author has verified by numerical methods that the solution given is optimal (at least for m up to 16). The recent paper [4] gives some variants of Markov chains on independent sets which are analyzed using path-coupling. Their results improve on those in [8] mentioned in the preceding sections, mainly for non-uniform sampling of independent sets (in the uniform case the degree needs still to be bounded by 4). Interestingly they show that the *insert-delete* chain on independent sets mixes rapidly not only when the degree is bounded by 2 (similar to our result on MC_0 in Theorem 5) but even when the degree is bounded by 4 (for the exact statement see [4, Thm. 5.5]). The proof is based on comparing the transition probabilities of the insert-delete chain with those of a more complicated chain on independent sets that mixes rapidly for degree-4 graphs. It seems possible that this method can be applied to show that the chain MC_0 mixes rapidly for the same posets as the chain MC_1. The author is currently investigating this conjecture.

The most interesting questions, however, are open:

Is the set of linear constrains $\mathcal{H}_0$ always feasible ? The only contribution to this we can offer at the moment, is that if we restrict ourselves to solutions h that depend only on $|X|$, as in most of the examples studied, then the answer is negative. There are bipartite graphs for which there is no such solution for the simple chain MC_0 on the independent sets of the graph.

Even more important is the question if rapid mixing of a chain is sufficient to guarantee a 'small' solution h. A positive answer would make the connection between the mixing rate and the polyhedra $\mathcal{H}_0$ much stronger. It might also open an approach to prove that a chain is *not* rapidly mixing.

We remark that while many interesting and important examples for which our technique applies, are actually distributive lattices, we do not use this additional structure. At the moment, all that is used, is the partial order relation and the uniqueness of the minimum and maximum. It would be very interesting if one could use the full structure of distributive lattices to obtain stronger results.

Finally we mention the paper by [6], which shows that the spectral gap of a Markov chain can be approximated (to a factor polynomial in the dimension) by semi-definite programming. Clearly this is a much stronger result, since it applies to general Markov chains and does not provide only an upper bound, but an approximation. However our approach is much simpler and there is reasonably

more hope to understand the polyhedra $\mathcal{H}_0$, then the structure of semi-definite programs.

References

1. D. Aldous. Random walks on finite groups and rapidly mixing Markov chains. In *Seminaire de Probabilites XVII*, volume 986 of *Lecture Notes in Math.*, pages 243–297. Springer Verlag, 1983.
2. L. Babai. Local expansion of symmetrical graphs. *Combinatorics, Probability and Computing*, 1:1–11, 1992.
3. Richard Durett. *Probability: Theory and Examples.* Duxbury Press, Belmont, CA, 1991.
4. M. Dyer and C. Greenhill. On Markov chains for independent sets. Preprint available at `http://www.scs.leeds.ac.uk/rand/acg.html`, 1997.
5. M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform generation. *Theoretical Computer Science*, 43:169–188, 1986.
6. N. Kahale. A semidefinite bound for mixing rates of Markov chains. Technical Report 95-41, DIMACS, 1995.
7. M. Luby, D. Randall, and A. Sinclair. Markov chain algorithms for planar lattice structures. In *Proc. of the 36th Symposium on Foundations of Computer Science*, pages 150–159. IEEE, 1995. Extended abstract.
8. M. Luby and E. Vigoda. Approximately counting up to four (extended abstract). In *29th STOC*. ACM, feb 1997.
9. J.G. Propp and D.B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1 & 2), August/September 1996.
10. A. Sharell. *Complexité Descriptive et l'Approximation des fonctions de dénombrement.* PhD thesis, L.R.I., Université Paris Sud, February 1998. Available at `http://www.lri.fr/~sharell`.
11. A. Sinclair. *Algorithms for Random Generation & Counting.* Birkhäuser, 1992.

Robotic Exploration, Brownian Motion and Electrical Resistance

Israel A. Wagner[1,2], Michael Lindenbaum[2] and Alfred M. Bruckstein[2]

[1] IBM Haifa Research Lab, Matam, Haifa 31905, Israel
israelw@vnet.ibm.com
[2] Department of Computer Science, Technion City, Haifa 32000, Israel
{*mic*|*freddy*}*@cs.technion.ac.il*

Abstract. A random method for exploring a continuous unknown planar domain with almost no sensors is described. The expected cover time is shown to be proportional to the electrical resistance of the domain, thus extending an existing result for graphs [11]. An upper bound on the variance is also shown, and some open questions are suggested for further research.

keywords: robotic exploration, cover time, Brownian motion, sheet resistance

1 Introduction

Exploring unknown terrain is an important issue in robotics. The problem has been intensively investigated, and several deterministic methods have been suggested and implemented. Most of those methods, however, rely on sophisticated, expensive and fragile systems of sensors (e.g. odometers, infra-red sensors, ultrasound radar or GPS), and/or sophisticated mapping algorithms. In this paper we suggest a minimalist approach in order to achieve the goal of covering with a minimum of sensing and computing, even if some performance reduction is implied. We show that on the average, a random walk is not too bad compared to deterministic algorithms that use much more sensing and computing to calculate their steps.

Formally, the on-line covering problem is to find a local rule of motion that will cause the robot to follow a *space-covering curve*, such that every point of the given region is in some prespecified r-neighborhood of the robot's trail, r being the covering radius of the robot. Such a rule, if obeyed for a sufficient number of steps, should lead the robot to follow a *covering path* which is a polygonal curve defined by the points $z_1, z_2, \ldots, z_T$, that *covers* a region R, i.e.,

$$R = \bigcup_{t=0}^{T} B_r(z_t), \tag{1}$$

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 116-130, 1998.

where $B_r(z)$ is a disk of radius r around z, and for all i, $|z_{i+1} - z_i| \leq r$ [3]. Note that the shape of R is not known in advance.

Existing methods for graph search (e.g. BFS, DFS) cannot be used for our purpose since no vertices or edges exist in our setting; a robot can move to arbitrary points on the continuum, while the BFS and DFS algorithms assume a discrete and finite set of possible locations. Also, those algorithms need a memory the size of which is, in general, proportional to the area to be explored. Yet another drawback of fully deterministic algorithms is their inability to provide a complete answer for realistic robotic problems, since both sensors and effectors are extremely vulnerable to noise and failures. As opposed to some purely computational problems, in robotics the environment of the robot is not known in advance and even if it is - it may change during operation. One way to tackle these problems is to make the robot itself non-deterministic by introducing randomness into its behaviour. This motivates our algorithm for the covering problem. We call this method PC - Probabilistic Covering. The basic rule of behavior here is to make a short step and then a random turn. Somewhat surprisingly, the expected performance of the PC approach is not so bad; for example, it covers a gridded rectilinear polygon in average time $O(n\rho \log n)$, where n is the area of the polygon and ρ - its "electrical resistance," to be defined and explained below.

Some **related work** has already been done in various areas:

- **Robotic covering:** In previous work ([14],[15]) a discrete problem of graph-exploration was solved using markers. More recently, the problem of covering a tiled floor was addressed in two different ways: In [29] the dirt on the floor served as memory to help the robot's navigation, while in [31] and [30] a vanishing trace was used for that purpose. In [6] the issue of inter-robot communication is addressed in the context of various missions, among them *grazing* - i.e. visiting every point of a region for purposes of object-fetching. There, a reactive model of behavior is presented, and simulation shows that detailed communication does not contribute too much to the performance. In [5] many experimental works are presented for planetary exploration by autonmous robots. Heuristic navigation methods are given in [17] for path planning of an autonomous mobile cleaning robot, and in [20] for a robot exploration and mapping strategy. However no rigorous analysis is given in the above references. In [18] an algorithm is presented for exploration of an undersea terrain, using exact location sensors and internal mapping. Practical implementations of covering algorithms have been demonstrated in [32] and [27]. In [32] a set of robots is described that help clean a railway station, using magnetic lines on the floor as guidelines. This method seems to work well, but is limited to pre-mapped regions. In [27] a cooperation of a team of robots is created by an explicit level of inter-robot communication. Each robot can choose one of multiple possible behaviors, according to its

[3] Note that if $r \to 0$, a covering path tends to be a **space** *filling* **curve** [28], which is a continuous 1-dimensional curve that fills a 2-dimensional domain.

specific conditions. In one of these behaviors the robot plays the role of a janitorial service man, by cleaning the dust around it.

- **Randomization and uncertainty in robotic tasks:** Uncertainty is an inherent factor in any real-life action, in particular one that relies on the information gained from sensors and manipulations performed by actuators. One way to cope with uncertainty is *randomization* - introducing a random selection into the robot's control. In [16] and [22], randomization is used to (partially) overcome uncertainty in various robotic tasks. In a sense, our PC algorithm is an extreme case of randomization, whereas almost no sensors are used.
- **Random Walk and Covering:** The analogy between random walks on graphs and the resistance of electrical networks was presented in [25], and later in [13], where it was used for investigating the recuerrence properties of random walks on 1, 2 and 3 dimensional grids. The rate of coverage of graphs by a random walk has been studied intensively. Two representative results in this context are the upper bounds of $O(mn)$ on the cover time of a graph with m edges and n vertices [2], and $O(m\rho \log n)$ where ρ is the resistance of the graph, assuming all edges to be 1-Ohm resistors [11]. In [10] it was shown that several random-walkers, if properly distributed in the graph, can bring a significant speed-up to the process of covering. Coverage of *continuous* domains by a Brownian motion process was less investigated. A significant contribution was made in [24], where a simple relation was derived between the cover time and the hitting time in a strong Markov process. The current paper aims to make a further progress in this direction, by relating the cover time of a Markov process (with discrete time and continuous location) to the electrical resistance of the explored region.
- **Off-line covering:** The off-line version of the problem (i.e. finding the shortest covering path for a given polygon) is NP-hard. The proof, as well as approximation algorithms for it are presented in [1]. The related (NP-hard) problem of optimal watchman route is to find the shortest path in a polygon such that every point of the polygon is *visible* from a point of the path. This problem is investigated in [12]. The goal there is to design a minimum-length path that will see each and every point in a given (i.e. known in advance) polygon.

The rest of the paper is organized as follows. In Section 2 we show a lower bound on the length of any covering path. Then in Section 3 we describe the PC process and show that the expected cover time and its variance can be expressed in terms of the electrical resistance of the shape to be covered. In section 4 we apply our results to prove the existence of a universal traversal sequence of angles, and then conclude with a discussion and some open questions.

2 A Universal Lower Bound on the Cover Time

We shall now show a lower bound on the length of any covering path, independent of the algorithm used to generate it.

Lemma 1. *The number of points in a covering sequence of r-circles, say* $Z = z_1, z_2, \ldots, z_{T_c}$, *such that* $|z_{i+1} - z_i| \leq r$, *is bounded from below*

$$T_c \geq \left\lceil \frac{6\pi}{4\pi + 3\sqrt{3}}(A/a) - 1 \right\rceil, \tag{2}$$

where A *is the region's area and* $a = \pi r^2$ *- the area covered by the robot in a single step.*

Proof: In each step (except, perhaps, the first one) the robot jumps at most a distance of r, and hence (due to overlapping) adds at most $(\sqrt{3}/2 + 2\pi/3)r^2$ to the covered area. Thus, after T points, the covered area is at most $S_T = (T-1)(\sqrt{3}/2 + 2\pi/3)r^2 + \pi r^2$. By equating S_{T_c} to A the lemma is implied. □

Remark: It is intuitively reasonable to assume that as r decreases, the "quality of covering" improves, i.e. the amount of overlap reduces. This intuition is made clear by the following result from [21]. Define $N(r)$ as the minimum number of r-circles needed to cover a region of area A. Then

$$\lim_{r \to 0} N(r) = (2\pi/\sqrt{27})(A/a), \tag{3}$$

and the minimum is attained in the "honeycomb" (hexagonal) arrangement of the circles, obtained by tiling the plane with congruent regular hexagons and circumscribing each hexagon with a circle. Note that the above result from [21] implies that, asymptotically, the cover time T_c cannot go below $1.209\ldots\times(A/a)$, while Lemma 1 implies that for *any* value of r, $T_c \geq 1.06\ldots\times(A/a)$.

In the rest of the paper we shall confine ourself to the problem of covering a unit-grid polygon of size n, i.e. a polygon made of a connected set of n unit squares on the grid. Two squares are considered connected if they have a common edge. We shall also assume that the covering radius of the robot is $\sqrt{2}$; thus we have that $A = n$ and $a = 2\pi$ and it follows from Lemma 1 that

Corollary 2. *If* R *is a unit-grid polygon of size* n, *then at least* $\left\lceil \frac{3n}{4\pi+3\sqrt{3}} \right\rceil$ *steps of a* $\sqrt{2}$*-radius robot are necessary to cover it.*

The *off-line* version of the covering path problem (i.e. when the shape of R is given in advance) is known to be NP-hard, and there are various heuristics to solve it [1]. However in many practical situations, the *on-line* problem is more relevant, since an efficient on-line solution enables an autonomous robot to cover a region without the need to be pre-programmed with a detailed map, thus being able to serve different shapes with the same hardware. Other advantages of the on-line approach are the ability to tolerate changes in the geometry and topolgy of the environment, and the flexible mode of cooperation that can only be achieved via on-line approach, while the pre-programming one is severly limited in this respect. This, in addition to the high cost of implementing a reliable system of sensors (which is needed for deterministic covering algorithms) motivates our probabilistic approach to the covering problem.

3 PC (Probabilistic Covering) - A Randomized Approach to the Covering Problem

In this section we consider a robot that acts with (almost) no sensory inputs; it makes a step, chooses a random new direction, and then makes another step. Clearly, the average performance of this method is not the optimal one, but it has the advantage of being almost sensorless, thus it is cheap and tolerant. In fact the only sensing is required for knowing how far are we from the boundary.

In the sequel we shall refer to the r-disk around z by $B_r(z)$, and to the r-circle around z by $C_r(z)$. Formally, the rule of motion is defined as follows:

```
/* PC - Probabilistic Covering with an r-disk */
Rule PC(z: current location)
A) cover B_r(z);
B) set μ(z) = min{r, max_{(B_{2r'}(z) ⊂ R)} {r'}} ;
    /* μ(z) is half the maximum radius */
    /* (not exceeding r) */
    /* of a circle around z within R */
C) choose a random neighbor w from C_{μ(z)}(z);
D) go to w;
end PC.
```

See Figures 1, 2 and 3 for examples of the process [4]. Note that if $C_r(z)$ intersects the boundary of R, then the duration of a PC step shall be shorter than one unit of time, since the step length is $\mu(z) < r$. In each step the robot scans around to see if a boundary exists within distance r; hence we shall assume that the time spent at z is proportional to $(\mu(z))^2$, where $\mu(z)$ is half the maximum radius not exceeding r of a circle around z within R. The reason for making the step length half the possible maximum is to avoid the chance of the robot going to ∂R, where it will get stuck forever since $\mu(z)$ vanishes on the boundary.

We model the robot as a point that covers a circle of radius r around itself. Due to the random nature of PC, no deterministic bound can be stated on the cover time; we shall, however, draw some bounds on the *expected* cover time and its *variance*, and both will be given as functions of the electrical resistance of

[4] A JAVA simulator of the PC process is web-accessible through: *http://www.cs.technion.ac.il/~wagner/pub/mac.html*

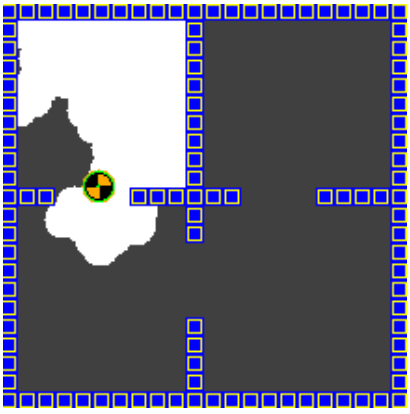


Fig. 1. A lonesome PC robot; grey area has not yet been covered.

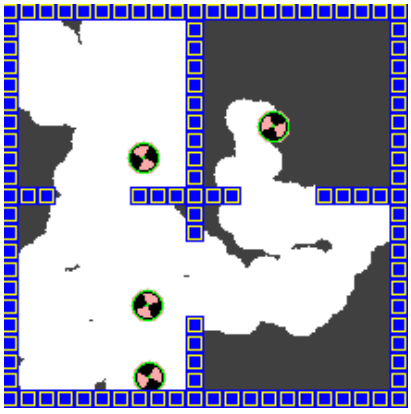


Fig. 2. Four PC robots working together. A fellow robot is considered as an obstacle, hence no collisions should occur according to the PC rule.

a conductive material in the shape of R. This resistance can be further related to the geometrical properties of the robot and the region. More specifically, we prove the following:

1. **Expected time of complete coverage : $\mathbf{E}\left[T^{\mathsf{PC}}\right]$**, the expected time until full coverage of R - a unit-grid polygon of size n by a PC robot which covers a radius of $\sqrt{2}$, is bounded by

$$2n\rho \leq \mathbf{E}\left[T^{\mathsf{PC}}\right] \leq 2n\rho \log n, \tag{4}$$

   where ρ is the electrical resistance of R (assuming a material of unit *sheet-resistance*, to be defined in the sequel). Note that the resistance $\rho = \rho(R)$ can sometimes be bounded in terms of the geometrical properties of the shape, and can always be numerically approximated. E.g. if R is a $\sqrt{n} \times \sqrt{n}$ square then its resistance is $O(\log n)$, when measured between a bottom left and a top right squares. In case of an $a \times b$ rectangle with $a << b$, $\rho = O(b/a)$. Recall from Corollary 2 that *any* covering path should have at least $\lceil n/\sqrt{27} \rceil$ steps.
2. **Variance in the cover time: $\mathbf{V}\left[T^{\mathsf{PC}}\right]$**, the variance in time of complete coverage, is bounded from above:

$$\mathbf{V}\left[T^{\mathsf{PC}}\right] \leq 2^{11} n\rho, \tag{5}$$

   which yields an upper bound on the standard deviation of the cover time:

$$\sigma\left[T^{\mathsf{PC}}\right] = \sqrt{\mathbf{V}\left[T^{\mathsf{PC}}\right]} \leq 32\sqrt{2n\rho}. \tag{6}$$

Our results can be extended to more general shapes, but this involves various types of cumbersome details that will be ommited in this extended abstract. Note that The above results are achieved without using any sensors except collision detectors, (the robot cannot distinguish "tiles" or "grid squares") and thus have almost no vulnerability to noise. It can be used as is, or be combined with a sensor-based algorithm to achieve a tradeoff between cover time and coverage guarantee.

3.1 Analysis of the Cover Time by PC

There is a wealth of results in the literaure for cover times by random walk on graphs, a sample of which was mentioned in the introduction. Our case is different, however, since the robot can occupy any point in the continuum of the region, rather than being bounded to a finite set of such points. One may wish to partition the region into squares, and then consider a random walk on a graph with the set of squares as its vertex set; but this will not do because the transition probabilities are not constant; rather, they depend on the precise location of the robot within a square (i.e. the process is not time-homogeneous). Hence we shall use continuous arguments to analyse the process.

We first observe that the PC process is a strong Markov process, since the probability of visiting a location depends only on the previous location but not on the earlier history - the robot has no memory. It was proved in [24] that under such a process, if $Q = \{q_1, q_2, \ldots, q_n\}$ is a collection of subsets of a set R, then $\mathbf{E}\left[T(q_1, q_2, \ldots, q_n)\right]$, the expected time for visiting some point of every subset in Q (starting from anywhere in R) is bounded as follows:

$$h_{max} \leq \mathbf{E}\left[T(q_1, q_2, \ldots, q_n)\right] \leq h_{max} \sum_{i=1}^{n} 1/i, \tag{7}$$

where $h_{max} = \max_{x \in (R \setminus Q), 1 \leq i \leq n} \{h_i(x)\}$, and $h_i(x)$ is the expected time to first reach subset q_i upon starting from $x \in R$. Let us denote the set of unit-squares in R by $S = \{s_1, s_2, \ldots, s_n\}$. This partition is not known to the robot, but will serve us in our analysis. In order to establish bounds on the average cover time of the PC process, we further observe that (since the robot's covering radius is $r = \sqrt{2}$) if the robot has visited all the n squares in R, then R is totally covered. See Figure 3 for an example. Clearly, if a robot is located anywhere within such a square, the whole square is covered (actually, some of the neighbor squares are also partially covered, but this does not make any harm to our upper bound result). Thus, visiting all the small squares is sufficient to guarantee a full coverage of R. On the other hand, in order to cover R starting from any point in it, the robot should make, at least once, the tour between the two farthest squares in R. Let us define the *hitting time* (also known as *access time* or *first-passage time*) from a point $x \in R$ to a square s_j, denoted $h_j(x)$, as the expected time of a PC process that starts at x and ends upon first reaching a point in square s_j. We also define $C_{i,j}$, the *commute time* between squares s_i and s_j as the average time of a round trip from s_i to s_j and back, i.e. $C_{i,j} = C_{j,i} = \max_{x \in s_i, y \in s_j} \{h_j(x) + h_i(y)\}$. It is thus

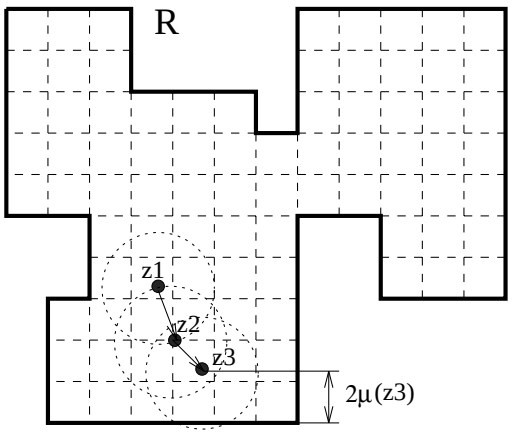


Fig. 3. A grid polygon R, partitioned into unit squares, and a possible sequence of PC steps which take random continuous locations z_1, z_2, z_3, thus covering the dashed circles. In this case, $\mu(z_2) > \mu(z_3)$, and hence the step size at time $t = 2$ is greater than at time $t = 3$. The dashed circles designate the covered area. Note that, since the covering radius is always $2^{1/2}$ while the grid size is 1, it is sufficient to visit all squares in order to guarantee a coverage of R.

implied by Equation 7 (using $\sum_{i=1}^{n}(1/i) < 2\log n$) and the above observations that the expected cover time of R can be bounded:

$$\max_{s_i,s_j \in R} \{C_{i,j}\} \leq \mathbf{E}\left[T^{\mathsf{PC}}\right] \leq 2(\log n) \max_{s_i,s_j \in R} \{C_{i,j}\} . \tag{8}$$

In order to find the maximum commute time ($C_{i,j}$) in R, we now show that the commute time between any squares s_i, s_j in R is proportional to the product of the number of squares in R and the electrical resistance between s_i and s_j. The following Lemma is a continuous analog to [11] which related the commute time of a random walk on a graph with its electrical resistance, considering each edge as a 1-Ohm resistor.

Lemma 3. *$C_{i,j}$, the commute time between squares s_i and s_j in R, obeys Equation $C_{i,j} = 2n\rho_{i,j}$, where n is the size of R and $\rho_{i,j}$ is the electrical resistance between squares s_i and s_j, assumming R to be made of a uniform material with unit sheet resistance* [5].

Proof: Let us denote the maximum step size by r. In a step, the PC robot selects a random angle and goes in that direction. The length of the step is $\mu(z)$, half the maximum radius not exceeding r of a circle around z within R. As explained

[5] The *sheet resistance* of a material is defined as the voltage across a square of the material caused by one unit of current (i.e. one Ampere) that is flowing between two parallel edges of the square. The sheet resistance is commonly expressed in units of Ohms per square.

before, we assume the time spent at z to be $(\mu(z)/r)^2$ which is one unit in an internal point of R (i.e. where $\mu(z) = r$), and less near the boundary, where $\mu(z) < r$ and steps are shorter (see Figure 3). If $z \notin s_j$, then the expected time to reach square s_j from z is just the average of the step length plus the access time over a $\mu(z)$-circle around z, i.e.

$$h_j(z) = (\mu(z)/r)^2 + \frac{1}{2\pi}\int_{\theta=0}^{2\pi} h_j(z + \mu(z)e^{i\theta})d\theta, \tag{9}$$

where $z + \mu(z)e^{i\theta}$ refers to a point at distance $\mu(z)$ from z and angle θ to the x axis, in the complex notation. Clearly if $z \in s_j$ then $h_j(z) = 0$.

Now consider R as a flat surface of a uniformly resistive material with unit sheet resistance, and assume that a current of $I_0 = 4/r^2$ Amperes per unit of area is uniformly injected into R, and $4n/r^2$ Amperes are rejected from R via the square s_j. Let us also denote the electric potential at point z relative to square s_j by $\phi_j(z)$. Since there are no current sources within R, we know from the Divergence Theorem (see, e.g. [19]) that for any closed surface, the amount of current entering the surface should equal the current exitting through it (i.e. the total current through the surface should vanish). Due to symmetry and uniformity of the resistance, the average potential around a circle of radius μ can be calculated:

Proposition 4. *The average potential difference between the center and the circumference of a circle of radius μ on a uniform surface with unit sheet resistance, into which I_0 Amperes of current are uniformly injected per unit area, is*

$$\overline{\phi(\mu) - \phi(0)} \stackrel{\text{def.}}{=} \frac{1}{2\pi}\int_{\theta=0}^{2\pi}(\phi(\mu e^{i\theta}) - \phi(0))d\theta = \frac{I_0\mu^2}{4}. \tag{10}$$

The proof of Proposition 4 is deferred to the Appendix. Choosing $I_0 = 4/r^2$ one gets $\overline{\phi(\mu) - \phi(0)} = (\mu/r)^2$ and hence (writing μ for $\mu(z)$ and $\phi_j(z)$ for the potential at z when the potential in square s_j is kept at zero):

$$\frac{1}{2\pi}\int_{\theta=0}^{2\pi}(\phi_j(z) - \phi_j(z + \mu e^{i\theta}))d\theta = (\mu/r)^2, \tag{11}$$

or

$$\phi_j(z) = (\mu/r)^2 + \frac{1}{2\pi}\int_{\theta=0}^{2\pi}\phi_j(z + \mu e^{i\theta})d\theta. \tag{12}$$

From the equivalence of Equations 9 and 12, and the uniqueness [6] of the ex-

[6] The function $h_j(z)$ is uniquely determined by

$$\begin{aligned} h_j(z) &= \sum_{t=1}^{\infty} t \cdot \mathsf{Prob}\{\text{square } s_j \text{ is reachable from } z \text{ in } t \text{ steps}\} \\ &= \sum_{t=1}^{\infty} t \cdot \frac{1}{(2\pi)^t}\int_{\theta_1=0}^{2\pi}\int_{\theta_2=0}^{2\pi}\cdots\int_{\theta_t=0}^{2\pi}\Delta(\theta_1,\theta_2,\ldots,\theta_t)\ d\theta_1 d\theta_2 \ldots d\theta_t, \end{aligned}$$

where $\Delta(\theta_1, \theta_2, \ldots, \theta_t) = 1$ if the sequence of angles $\theta_1, \theta_2, \ldots, \theta_t$ leads from point z to (some point of) square s_j, and 0 otherwise.

pectation function $h_j(z)$, we see that $h_j(z)$ is equal to the potential difference $\phi_j(z) - \phi_j(s_j)$ if $4r^{-2}$ units of current are injected into each unit of area, and $4nr^{-2}$ units of current are rejected from s_j. In a similar way one can show that $h_i(z) = \phi_i(z) - \phi_j(s_i)$, if $4/r^2$ units of current are injected into each unit of area, and $4n/r^2$ units of current are rejected from s_i. Now if we reverese the direction of all currents in the second case, we get that $h_i(z) = \phi_j(s_i) - \phi_i(z)$, if $4/r^2$ units of current are rejected from each unit of area across R, and $4n/r^2$ units of current are injected into s_i. Due to linearity of resistive electrical systems, we can superpose both sheets together, thus making all currents cancel each other, except the $4n/r^2$ Amperes injected at s_i and rejected from s_j. This, together with Ohm's law [7], implies that $C_{i,j}$, the commute time between squares s_i and s_j obeys

$$\begin{aligned} C_{i,j} &= \max_{x \in s_i, y \in s_j} \{h_j(x) + h_i(y)\} \\ &= \max_{x \in s_i, y \in s_j} \{\phi_j(x) - \phi_i(y)\} = \frac{4n}{r^2} \rho_{i,j}, \end{aligned} \tag{13}$$

where $\rho_{i\,j}$ is the maximal electrical resistance between squares s_i and s_j in R. This resistance is measured by injecting a 1-Ampere current into one square, say s_i, while rejecting it from s_j. Then the maximum potential diiference between a point in s_j to one in s_j is equal to $\rho_{i,j}$.

Substituting $r = \sqrt{2}$ in Equation 13 yields the Lemma. □

We now combine the above results to obtain

Theorem 5. $2n\rho \leq \mathbf{E}\left[T^{\mathsf{PC}}\right] \leq 2n\rho \log n$, *where n is the size of R and ρ - its resistance.*

Proof: immediate, by substituting Lemma 3 in Equation 8. □

A corollary is implied for a square room:

Corollary 6. *If R is a square $a \times a$ room, then*

$$c_1 a^2 \log a \leq \mathbf{E}\left[T^{\mathsf{PC}}\right] \leq c_2 a^2 \log^2 a, \tag{14}$$

where c_1, c_2 are small constants.

Proof (sketch): We use the fact that the resistance of a square is $\Theta(\log a)$ [8]. Then we also note that for an $a \times a$ room, $n = a^2$, which, substituted into Theorem 5, implies the corollary. □

[7] Ohm's law says that the voltage drop between two points is equal to the product of the current flowing between the points and the point to point resistance.

[8] It is of interest to mention a lumped circuit analogy: a square $m \times m$ mesh of 1-Ohm resistors is known [11] to have resistance $\Theta(\log n)$.

3.2 An upper bound on the variance of T^{PC}

In order for our results to be useful, we now show that the variance of the cover time, denoted $\mathbf{V}\left[T^{\mathsf{PC}}\right]$, is also bounded from above and hence there is only a limited spread of the covering time around its average. It has been proved in [3] that the variance in the cover time of a set S is at most constant times the expected time of covering the last item in the set, i.e.

$$\mathbf{V}\left[T^{\text{cover of } S}\right] \leq c_0 \cdot \mathbf{E}\left[T^{\text{cover of the last item in } S}\right], \tag{15}$$

where c_o is a constant [9] less than 2^{10}. Applying it to our case, we can use the maximum access time as an upper bound to the cover time of the last item (i.e. a yet-unvisited square), so we get:

$$\mathbf{V}\left[T^{\mathsf{PC}}\right] \leq 2^{10} \max_{s_i, s_j \in R} \{C_{i,j}\} \leq 2^{11} n\rho, \tag{16}$$

which implies that the standard deviation is at most $32\sqrt{2n\rho}$.

4 A Universal Traversal Sequence of Angles

Let us define a universal traversal sequence of angles (UTSA) for a family of planar sets $\mathcal{F}$ as a finite sequence of real numbers $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_M)$, all in $[0, 2\pi)$, such that if a PC robot takes the turn α_t in step t, it is guaranteed to cover *any* shape from $\mathcal{F}$, independent of the starting point. In this section we shall show that if $\mathcal{F}$ is the set of all n-size unit-grid polygons, (i.e. polygons made of n attached 1×1 squares), then such a sequence exists and has a length polynomial in n. For this purpose we follow the probabilistic method invented by Erdös and used in [2] to prove that a sequence of length $O(n^3 \log n)$ exists that covers any edge-labelled k-regular graph [10] with n vertices.

Theorem 7. *There exists a sequence of* $2n^4 \log n$ *angles that guarantees covering of any rectilinear gridded polygon of size* n*.*

Proof: First let us observe that if $\mathcal{F}$ is the set of all n-size unit-grid polygons, then $|\mathcal{F}| < 2^{n^2}$ (since all polygons of size n can be enclosed by an $n \times n$ square). We next apply Theorem 5 to obtain an upper bound of $t = 2n^2 \log n$ on the expected cover time of any polygon in $\mathcal{F}$, using the fact that the resistance ρ obeys $\rho \leq n$ for such polygons. Hence, after a sequence of t random turns, the

[9] This value of the constant does not appear in [3], but can be calculated based on the analysis done there.

[10] a graph is *k-regular* if exactly k edges emanate from every vertex. It is *edge-labelled* if the edges emanating from each vertex are numbered in some order.

probability of complete coverage is at least $1/2$, and after an mt-long sequence it is at least $1 - 2^{-m}$. On the other hand,

$$\begin{aligned} &\mathsf{Prob}\,\{\exists R \in \mathcal{F} \text{ s.t. } R \text{ is not covered by a random } mt\text{-long sequence }\} \\ &\leq \sum_{R \in \mathcal{F}} \mathsf{Prob}\,\{\ R \text{ is not covered by a random } mt\text{-long sequence }\} \\ &\leq 2^{-m}\,|\mathcal{F}| \ \leq\ 2^{n^2-m}. \end{aligned} \tag{17}$$

Hence if we choose $m > n^2$ then the probability for existence of an mt-long sequence that does *not* cover all polygons in $\mathcal{F}$ is less than one, i.e. there exists such a sequence which *does* guarantee covering of all polygons in $\mathcal{F}$, and hence there is a $(2n^4 \log n)$-long sequence of angles which is a UTSA for $\mathcal{F}$. □

Note that finding a universal sequence of length $O(4^n)$ is easy - just traverse the quaternary tree of height n with the starting point as the root and with four sons to each vertex, each representing a turning angle from $\{0, \pi/2, \pi, 3\pi/2\}$. Backtracking is possible thanks to the "compass" that our robot has. Clearly, not all steps will be of length r because of walls and obstacles, but eventually all squares will be reached.

5 Summary

We have shown that the expected cover time by a process of random steps in a continuous polygon is related to the electrical resistance of the polygon. The setting of continuous space is more relevant to robotics than the discrete structure of graphs, since robots move continuously, and even if a discrete partition is dictated by some external signs (e.g. a tiled floor), it is still hard for a low-cost robot to precisely identify those signs. The problem of continuous covering has various implications for both theory and practice. The analysis suggested in this paper can serve as an inspiration for further research in several directions, some of which are described below.

1. **Cooperating PC robots** In a multi-robot setting we just add robots and let them all follow the same PC rule. It is intriguing to see what if a more significant communication is enabled, e.g. if a collision with another robot or with the wall makes the future steps biased against the (alleged) location of other robots/walls.
2. **Finding a "short" universal traversal sequence of angles:** We have shown the existence of a polynomial-length universal sequence of angles (UTSA) for gridded polygons. However we do not know how to find one. The similar question for graphs is also wide open, with the only exceptions (known to us) being paths and cycles [9], [8]. Intuitively, one may think that finding a UTSA in our case is easier, since the robot is assumed to have a kind of "compass", while in the UTS problem for graphs, edges are arbitrarily ordered.

6 Acknowledgement

We would like to thank David Aldous for his advice with respect to the results in [3].

References

1. Arkin E. M., Hassin R., "Approximation Algorithms for the Geometric Covering Salesman Problem," *Discrete Applied Math. 55,* pp 197-218, 1994.
2. Aleliunas R., Karp R.M., Lipton R. J., Lovasz L., Rakoff C., "Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems,"in *20'th Annual Symposium on Foundations of Computer Science*, p. 218-223,San Juan, Puerto Rico, October 1979.
3. Aldous D. J. , "Threshold Limits for Cover Times," *Jornal of Theoretical Probability,* Vol. 4, No. 1, 1991, pp. 197-211.
4. Berger M., *Geomtery II*, Springer-Verlag, Berlin-Heidelberg 1987.
5. Giralt G., Weisbin C., (Editors), Special issue on autonomous robots for planetary exploration,*Autonomous Robots,* 2 (1995) pp. 259-362.
6. Balch T., Arkin R. C., "Communication in reactive multiagent robotic systems,"*Autonomous Robots,* 1 (1994) pp. 27-52.
7. Baeza-Yates R., Culberson J. C., Rawlins G. J. E. , "Searching in the Plane," *Information and Computation*, 106 (1993) pp. 234-252.
8. Bar-Noy A., Borodin A., Karchemer M., Linial N., Werman M., "Bounds on Universal Sequences," *SIAM J. Comput.,* Vol. 18, No. 2, pp.268-277, (1989).
9. Bridgland M.F., "Universal Traversal Sequences for Paths and Cycles," *J. of Alg.,* **8**, (1987), pp.395-404.
10. Broder A. Z., Karlin A. R., Raghavan P., Upfal E., "Trading Space for Time in Undirected $s-t$ Connectivity," *SIAM J. COMPUT.*, Vol. 23, No. 2, pp. 324-334, April 1994.
11. Chandra A. K., Raghavan P., Ruzzo W. L., Smolensky R., Tiwari P., "The Electrical Resistance of a Graph Captures its Commute and Cover Times," *Proc. 21st ACM STOC,* (1989), pp. 574-586.
12. Chin W. P., Ntafos S., "Optimum Watchman Routes," *2'nd Annual Symposium on Computational Geometry*, Yorktown Heights, NY, June 2-4, 1986, pp. 24-33.
13. Doyle P. G., Snell J. L., *Random Walks and Electric Networks,* Mathematical Association of America, Washington, D. C., 1984.
14. Dudek G., Jenkin M., Milios E., Wilkes D., "Robotic Exploration as Graph Construction,"*IEEE Trans. on Robotics and Automation* , Vol. 7, No. 6, Dec. 1991.
15. Deng X., Mirzaian A., "Competitive Robot Mapping with Homogeneous Markers,"*IEEE Trans. on Robotics and Automation* , Vol. 12, No. 4, Aug. 1996.
16. Erdmann M., "Randomization in robot tasks,"Int. J. Robot. Res.,11(5):399-436, October 1992. Hall P., *Introduction to the Theory of Coverage Processes,*John Wiley & Sons, New York, 1988
17. Hofner C., Schmidt G., "Path planning and guidance techniques for an autonomous mobile cleaning robot,"*Robotics and Autonomous Systems (1995)*, 14:199-212.
18. Hert S., Tiwari S., Lumelsky V., "A terrain covering algorithm for an AUV,"*Auton. Robots* , Vol.3, No.2-3 June-July 1996, pp. 91-119
19. Kaplan W., *Advanced Calculus,* 3'rd Ed., Addison-Wesley, Reading, MA, 1984.

20. Kuipers B., Byun Y. T., "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Robotics and Autonomous Systems (1981)*, 8:47-63.
21. Kershner R., "The number of circles covering a set," *Amer. J. Math. (1939)*, 61:665-671.
22. LaValle S. M., Hutchinson S. A., "Evaluating Motion Strategies under Nondeterministic or Probabilistic Uncertainties in Sensing and Control," *Proc. of the 1996 IEEE Intl.Conference on Robotics and Automation*, pp. 3034-3039.
23. Lovasz L., "Random Walks on Graphs - a Survey," in: *Combinatorics, Paul Erdös is Eighty, Part 2* Ed. D. Miklos, V. T. Sos, T. Szony, Janos Bolyai Mathmatical Society, Budapest, 1996, Vol. 2, pp. 353-398.
24. Matthews P., "Covering Problems for Brownian Motion On Spheres," *The Annals of Probability,* 1988, Vol. 16, No. 1, pp. 189-199.
25. Nash-Williams C. St. J. A., "Random walk and electric currents in networks," Proc. Camb. Phil. Soc., 55:181-194, 1959.
26. Pach J. (Ed.), *New Trends in Discrete and Computational Geometry,*Springer-Verlag, Berlin Heidelberg 1993.
27. Parker L. E., "On the design of behavior-based multi-robot teams," *Advanced Robotics,* Vol. 10, No. 6, pp. 547-578 (1996).
28. Sagan H., *Space-Filling Curves,*Springer-Verlag, New York, 1994.
29. Wagner I. A., Bruckstein A. M., "Cooperative Cleaners - a Study in Ant-Robotics,"in A. Paulraj, V. Roychowdhury, C. D. Schaper - ed.,*Communications, Computation, Control, and Signal Processing: A Tribute to Thomas Kailath*, Kluwer Academic Publishers, The Netherlands, 1997, pp. 289-308.
30. Wagner I. A., Lindenbaum M., Bruckstein A. M., "On-Line Graph Searching by a Smell-Oriented Vertex Process,"Working notes of *AAAI'97 Workshop on On-Line Search,*July 28, 1997, Providence, Rhode Island, pp. 122-125.
31. Wagner I. A., Lindenbaum M., Bruckstein A. M., "Smell as a Computational Resource - A Lesson We Can Learn from the Ant," Proceedings of the *4'th Israeli Symposium on the Theory of Computing and Systems*, Jerusalem, June 10-12, 1996.
32. Yaguchi H., "Robot introduction to cleaning work in the East Japan Railway Company," *Advanced Robotics*, Vol. 10, no. 4, pp. 403-414 (1996).

Appendix: Potential Difference Across A Uniformly-Resistive Circle

Proof of Proposition 4 :
Consider a circle of radius μ and unit sheet-resistance, and assume that a current of I_0 Amperes per unit area is uniformly injected into the circle. We seek for the average potential difference (or "voltage drop") between the center of the circle and its circumference, defined by

$$\overline{\phi(0) - \phi(\mu)} = \frac{1}{2\pi} \int_{\theta=0}^{2\pi} (\phi(0) - \phi(\mu e^{i\theta}))d\theta. \tag{18}$$

Consider a ring of radius u and infinitesimal width du (see Figure 4).

We know (from the Theorem of Divergence) that, since there are no sources or sinks of current on the surface, all the current injected into the u-circle should flow out across its boundary and into the ring. This amount of current is $I_0 \pi u^2$.

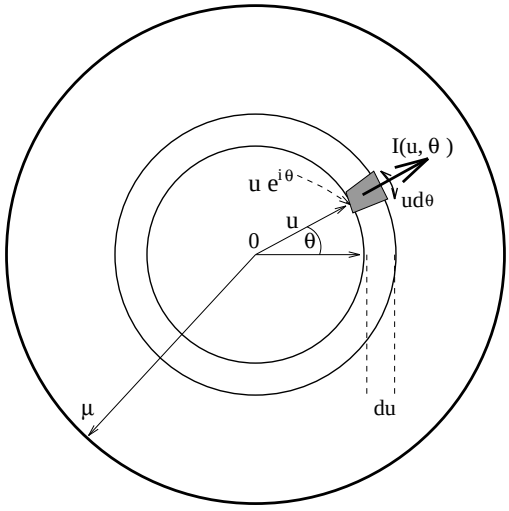


Fig. 4. An infinitesimal ring within a circle. The average voltage drop across the ring is obtained by integrating over small trapezoids like the one in gray, through which the centrifugal current $I(u,\theta)$ is flowing.

Let us denote by $I(u,\theta)$ the centrifugal current flowing at $ue^{i\theta}$ in direction θ, by $d\phi(u,\theta)$ the voltage drop between the inner and outer edges of an infinitesimal part of the ring, and by $\overline{d\phi(u)}$ the average voltage drop across the ring. One can now write

$$\overline{d\phi(u)} = \frac{1}{2\pi}\int_{\theta=0}^{2\pi} d\phi(u,\theta)d\theta = \frac{1}{2\pi}\int_{\theta=0}^{2\pi} \frac{I(u,\theta)dud\theta}{ud\theta}$$

(the resistance of a rectangle is length/width)

$$= \frac{du}{2\pi u}\int_{\theta=0}^{2\pi} I(u,\theta)d\theta = \frac{du}{2\pi u}\pi u^2 I_0 = \frac{I_0 udu}{2} \quad . \tag{19}$$

Note that the voltage drop across the ring due to the current flowing into the ring itself is proportional to the product of this current ($o((u+du)^2 - u^2) = o(udu)$) and the ring's resistance ($(o(du/u))$, hence is $o((du)^2)$, and vanishes in integration. Thus the total voltage difference can be found by integrating along u:

$$\overline{\phi(0) - \phi(\mu)} = \int_{u=0}^{\mu} d\overline{\phi(u)}du = \int_{u=0}^{\mu} \frac{I_0 u}{2}du = \frac{I_0\mu^2}{4} \quad . \tag{20}$$

□

This article was processed using the LaTeX macro package with LLNCS style

Fringe analysis of synchronized parallel algorithms on 2–3 trees*

R. Baeza-Yates[1], J. Gabarró[2], and X. Messeguer[2]

[1] Departamento de Ciencias de la Computación, Universidad de Chile
[2] Dep. LSI, Universitat Politècnica de Catalunya, Barcelona

Abstract. We are interested in the *fringe* analysis of synchronized parallel insertion algorithms on 2–3 trees, namely the algorithm of W. Paul, U. Vishkin and H. Wagener (PVW). This algorithm inserts k keys into a tree of size n with parallel time $O(\log n + \log k)$.
Fringe analysis studies the distribution of the bottom subtrees and it is still an open problem for parallel algorithms on search trees. To tackle this problem we introduce a new kind of algorithms whose two extreme cases seems to upper and lower bounds the performance of the PVW algorithm.
We extend the fringe analysis to parallel algorithms and we get a rich mathematical structure giving new interpretations even in the sequential case. The process of insertions is modeled by a Markov chain and the coefficients of the transition matrix are related with the expected local behavior of our algorithm. Finally, we show that this matrix has a power expansion over $(n+1)^{-1}$ where the coefficients are the binomial transform of the expected local behavior. This expansion shows that the parallel case can be approximated by iterating the sequential case.

Keywords: Fringe analysis, Parallel algorithms, 2-3 trees, Binomial transform.

1 Introduction

Fringe analysis studies the distribution of the bottom subtrees or fringe of trees and has been applied to most search trees in the sequential case [EZG+82,BY95]

We are interested on the fringe analysis of the synchronized parallel algorithms on 2–3 trees designed by W. Paul, U. Vishkin and H. Wagener [PVW83]. This algorithm inserts k keys randomly selected with k processors in time $O(\log n + \log k)$ into a 2–3 tree of size n. The fringe analysis in this case is still open and the main drawback is the reconstructing phase that is composed by waves of synchronized processors which modifies the tree bottom-up.

In this paper we propose a new synchronized parallel algorithm, denoted MacroSplit, that bounds the [PVW83] one in the following sense: the distribution

* Partially supported by ACI-CONICYT through Catalunya-Chile Cooperation Program (DOG 2320-30.1.1997) and RITOS network (CYTED) and ESPRIT Long Term Research Project no. 20244–ALCOM IT and DGICYT under grant PB95-0787 (project KOALA) and CICIT TIC97-1475-CE and CIRIT 1997SGR-00366.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 131-144, 1998.

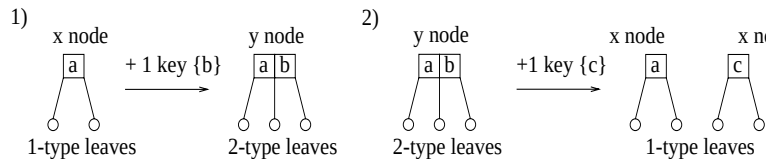


Fig. 1. The transformation of x and y bottom nodes after insertion of one key. In (1) the key b hits a bottom node x (containing the key a). Node x transforms into a node y (having keys a and b). We have $X_{t+1} = X_t - 2$ and $Y_{t+1} = Y_t + 3$. In case (2) the key c hits a bottom node y containing a and b. This the node y splits into 2 nodes x containing a and c respectively, while b is inserted in the parent node recursively. Now $X_{t+1} = X_t + 4$ and $Y_{t+1} = Y_t - 3$.

of the fringe derived from the [PVW83] algorithm is upper and lower bounded by the distribution derived from two extreme cases of our algorithm. The key idea is that our algorithm reconstructs the tree with only one wave meanwhile [PVW83] needs a pipeline of waves.

We have extended the fringe analysis from the sequential case into the parallel case with significant improvements . As later on is showed, the direct extensions of this technique on two concrete cases (the parallel insertion of two and three keys) suggest the inapplicability of this technique on cases greater than these simple ones. We have overcome this drawback with two facts allowing us the analysis of the generic case (the insertion of k keys):

- The random insertion of keys generates a *binomial* distribution on the bottom nodes. This fact allows us the probabilistic analysis of the parallel algorithm.
- The fringe evolution is determined by the expected local behavior of the algorithm. This fact gives a new understanding to fringe analysis.

The rest of the paper is organized as follows. In section 2 we recall the fringe analysis of the sequential case. In section 3 we introduce the MacroSplit algorithms. Section 4 contains the direct extension of the fringe analysis for the parallel introduction of two and three keys. Section 5 contains the analysis of the generic case and section 6 the analysis of two concrete cases of this generic case. Finally section 7 contains the conclusions.

2 Sequential case

The fringe of a tree is composed by the subtrees on the bottom part of a tree. Our fringe is composed by trees of height one. A bottom node with one key is called and x node, and a bottom node with two keys is called an y node. These nodes separate leaves into 1-type leaves if their parents are x nodes and 2-type leaves if their parents are y nodes.

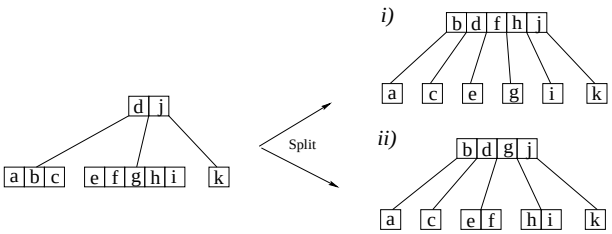


Fig. 2. Choices for MacroSplit rules. In (i) the MaxMacroSplit rule creates a maximum number of splits. In (ii) the MinMacroSplit rule creates the minimum number. Intermediate strategies are allowed.

Let X_t and Y_t be the random variables associated to the number of 1-type leaves and 2-type leaves respectively at the step t. We assume that $X_t + Y_t = n+1$ being n the number of keys of the tree. When a new key falls into a bottom node this node is transformed according the rules given in figure 1. The probability that a key hits a bottom node x is $\frac{X_t}{n+1}$ and for a node y is $\frac{Y_t}{n+1}$. The conditional expectations verify

$$E(X_{t+1} \mid X_t, Y_t, 1) = \frac{X_t}{n+1}(X_t - 2) + \frac{Y_t}{n+1}(X_t + 4) = \left(1 - \frac{2}{n+1}\right) X_t + \frac{4}{n+1} Y_t$$

$$E(Y_{t+1} \mid X_t, Y_t, 1) = \frac{X_t}{n+1}(Y_t + 3) + \frac{Y_t}{n+1}(Y_t - 3) = \frac{3}{n+1} X_t + \left(1 - \frac{3}{n+1}\right) Y_t$$

The expected number of leaves (conditioned to the random insertion of one key) at the step t can be modeled by [EZG+82,BY95]:

$$\begin{pmatrix} E(X_{t+1} \mid 1) \\ E(Y_{t+1} \mid 1) \end{pmatrix} = T_{n,1} \begin{pmatrix} E(X_t \mid 1) \\ E(Y_t \mid 1) \end{pmatrix}$$

As the conditional expectations verify $E(X_{t+1} \mid 1) = E(E(X_{t+1} \mid X_t, Y_t, 1) \mid 1)$ and $E(Y_{t+1} \mid 1) = E(E(Y_{t+1} \mid X_t, Y_t, 1) \mid 1)$ we get from the preceding expression the 1-OneStep transition matrix

$$T_{n,1} = \left(1 + \frac{1}{n+1}\right) I + \frac{1}{n+1} \begin{pmatrix} -3 & 4 \\ 3 & -4 \end{pmatrix} \qquad \text{being} \qquad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

In order to compare with the parallel case we consider the sequential insertion of k keys given by $T^{\mathsf{Seq}}_{n,k} = T_{n+k-1,1} \cdots T_{n,1}$. It is easy to prove

$$T^{\mathsf{Seq}}_{n,k} = \left(1 + \frac{k}{n+1}\right) I + \frac{k}{n+1} \begin{pmatrix} -3 & 4 \\ 3 & -4 \end{pmatrix} + O\left(\frac{1}{n^2}\right) I$$

k	x node	y node
1	y	xx
2	xx	xy
3	xy	xxx or yy
4	xxx or yy	xxy
5	xxy	$xxxx$ or xyy
6	$xxxx$ or xyy	$xxxy$ or yyy

Table 1. MacroSplit possibilities for x and y bottom nodes once k keys are inserted.

3 MacroSplit parallel insertion algorithms

We introduce a parallel insertion algorithm based on the idea of MacroSplit. On this algorithm an array of ordered keys $a[1\ldots]$ is inserted into a 2-3 tree having n leaves. The MacroSplit insertions algorithm has two main successive phases.

Percolation Phase. In a top-down strategy, the set of keys to be inserted is split into several packets and these packets are routed down. Finally, these packets are attached to the leaves [PVW83].

Reconstruction Phase. In a bottom-up phase the packets attached to the leaves are really inserted and the tree is reconstructed. This reconstruction is based in just *one unique* wave moving bottom up. First, the packets are incorporated at the bottom internal nodes of the tree. In successive steps the wave moves up, decreasing the depth one unit at each time. The evolution of this unique wave needs the usage of rules so called MacroSplit rules (see Figure 2).

The MacroSplit algorithm can be seen as a "height level" description of the parallel insertion algorithm given by W. Paul, U. Vishkin and H. Wagener in [PVW83] which take place by splitting a MacroSplit step into several more basic steps chained together in a pipeline.

Let us see why we have several MacroSplit algorithms for a large k. At most, k keys can reach a node. If the node stores more than two keys, it must split using a MacroSplit rule. Table 1 show us several split possibilities for x and y bottom nodes. For instance, the first row show us the splits of the x and y nodes when $k = 1$(see Figure 1). In this case there is just one possibility. The fourth row show us how x and y nodes can be split when $k = 4$. In this case a bottom node x can be split into 3 nodes x or into 2 nodes y. Later on we will consider two extreme cases. The MaxMacroSplit algorithm will maximize the number of splits at each step and the MinMacroSplit algorithm will minimize this number. When $k = 1$ or 2 both algorithms coincides (see table 1).

Consider that at the $t+1$ step k random keys (we asume a uniform distribution of them) fall in parallel into a fringe with X_t leaves of 1-type and Y_t leaves 2-type such that $X_t + Y_t = n + 1$. The expected values of X_{t+1} and Y_{t+1} after the insertions depends on two facts.

$(\cdot,\cdot)$	$P(\cdot,\cdot)$	$E(X_{t+1}\|X_t,Y_t,2,(.,.))$	$E(Y_{t+1}\|X_t,Y_t,2,(.,.))$
(x,x)	$\frac{X_t}{n+1}\frac{2}{n+1}$	X_t+2	Y_t
(x_1,x_2)	$\frac{X_t}{n+1}\frac{X_t-2}{n+1}$	X_t-4	Y_t+6
(x,y)	$2\frac{X_t}{n+1}\frac{Y_t}{n+1}$	X_t+2	Y_t
(y,y)	$\frac{Y_t}{n+1}\frac{3}{n+1}$	X_t+2	Y_t
(y_1,y_2)	$\frac{Y_t}{n+1}\frac{Y_t-3}{n+1}$	X_t+8	Y_t-6

Table 2. Parallel insertion of two keys

- The concrete form of the MacroSplit algorithm. This algorithm explicites how many leaves of 1-type and 2-type will be generated by bottom nodes when they receive some number of keys.
- The preceding values of X_t and Y_t.

We deal with a Markov chain and the evolution can be analyzed through the so called k-OneStep transition matrix $T_{n,k}$

$$\begin{pmatrix} E(X_{t+1} \mid k) \\ E(Y_{t+1} \mid k) \end{pmatrix} = T_{n,k} \begin{pmatrix} E(X_t \mid k) \\ E(Y_t \mid k) \end{pmatrix}$$

4 Parallel insertion of 2 and 3 keys

In this section we compute $T_{n,2}$ and $T_{n,3}$ following directly the technique applied before to sequential insertions [EZG+82] and we discuss the viability of this approach.

4.1 Direct extensions

First, let us consider the case $k = 2$. We have only one MacroSplit algorithm (see Table 1). The expected number of leaves is characterized by 2-OneStep $T_{n,2}$ transition matrix:

$$\begin{pmatrix} E(X_{t+1} \mid 2) \\ E(Y_{t+1} \mid 2) \end{pmatrix} = T_{n,2} \begin{pmatrix} E(X_t \mid 2) \\ E(Y_t \mid 2) \end{pmatrix}.$$

We compute the probabilities of the different splits by an exhaustive case analysis (see Table 2). As at most two keys can reach the same bottom node, we have no election in the split, *i.e.* the transformation of bottom nodes is unique (second row of table 1). Both keys can be either at the same bottom node or at different bottom nodes, and in each case bottom nodes can be of type x or y. Let $P(x,x)$ be

the probability that both keys reach the same x node, $P(x_1, x_2)$ the probability to reach different x nodes and so on for the remainder probabilities $P(x, y)$ and $P(y_1, y_2)$. We denote the generic case as $P(\cdot, \cdot)$, being $(.,.)$ the generic pair of nodes accessed.

As $E(X_{t+1} \mid 2) = E(E(X_{t+1} \mid X_t, Y_t, 2))$ we compute the expected number of 1-type leaves as $E(X_{t+1}|X_t, Y_t, 2) = \sum_{(.,.)} P(\cdot, \cdot)\, E(X_{t+1}|X_t, Y_t, 2, (.,.))$ being $E(X_{t+1}|X_t, Y_t, 2, (.,.))$ the expected number of 1-type leaves when 2 keys reach node $(\cdot, \cdot)$ conditioned to X_t and Y_t. For instance, if both keys reach different x nodes then it holds

$$P(x_1, x_2) = \frac{X_t}{n+1}\frac{X_t - 2}{n+1}$$

and $E(X_{t+1}|X_t, Y_t, 2, (x1, x2)) = X_t - 4$ (table 2 contains the other values). In the appendix we give the proof of the following lemma.

Lemma 1. *The conditional expectations verify*

$$E(X_{t+1} \mid X_t, Y_t, 2) = \left(1 - \frac{4}{n+1} + \frac{12}{(n+1)^2}\right) X_t + \left(\frac{8}{n+1} - \frac{18}{(n+1)^2}\right) Y_t$$

$$E(Y_{t+1} \mid X_t, Y_t, 2) = \left(1 - \frac{6}{n+1} + \frac{18}{(n+1)^2}\right) Y_t + \left(\frac{6}{n+1} - \frac{12}{(n+1)^2}\right) X_t$$

As the conditional expectations are linear in X_t and Y_t and $E(X_{t+1} \mid 2) = E(E(X_{t+1} \mid X_t, Y_t, 2))$, $E(Y_{t+1} \mid 2) = E(E(Y_{t+1} \mid X_t, Y_t, 2))$ we have:

Lemma 2. *The 2-OneStep transition matrix is:*

$$T_{n,2} = \left(1 + \frac{2}{n+1}\right) I + \frac{2}{n+1}\begin{pmatrix} -3 & 4 \\ 3 & -4 \end{pmatrix} + \frac{1}{(n+1)^2}\begin{pmatrix} 12 & -18 \\ -12 & 18 \end{pmatrix}$$

Consider briefly the case $k = 3$. Now there are two possibilities (third row of table 1). We have selected the second transformation. This corresponds to the MinMacroSplit algorithm. As before, an exhaustive case analysis give us:

Lemma 3. *In the case of the MinMacroSplit algorithm, the 3-OneStep transition matrix $T_{n,3}$ is:*

$$\left(1 + \frac{3}{n+1}\right) I + \frac{3}{n+1}\begin{pmatrix} -3 & 4 \\ 3 & -4 \end{pmatrix} + \frac{3}{(n+1)^2}\begin{pmatrix} 12 & -18 \\ -12 & 18 \end{pmatrix} + \frac{1}{(n+1)^3}\begin{pmatrix} -48 & 54 \\ 48 & -54 \end{pmatrix}$$

4.2 Discussion of the cases 2 and 3

Based on the preceding cases can point several facts and questions:

1. The exhaustive case analysis (generalizing the sequential approach [EZG+82]) for larger k, $k = 5, 6, \ldots$, becomes intractable.

2. For $k = 1, 2, 3$ the expectations $E(X_{t+1} \mid X_t, Y_t, k)$ and $E(X_{t+1} \mid X_t, Y_t, k)$ are linear in X_t and Y_t. It is unclear why non-linear terms always disappears. Note that we assume this point of view in the equation k-OneStep transition matrix $T_{n,k}$.
3. The intuitive meaning of the coefficients appearing in the expectations is unclear. For instance, the term $1 - \frac{4}{n+1} + \frac{12}{(n+1)^2}$ appearing in $E(X_{t+1} \mid X_t, Y_t, 2)$ in lemma 1 does not have any direct explanation in terms of the MacroSplit algorithm.
4. By *local behavior* of the algorithm we mean what happens when i keys hit just *one* bottom node x or y (table 1) . By *global behavior* we mean the evolution of X_t and Y_t. The previous exhaustive analysis does not give a clear cut between the *local* and the *global behavior* of the MacroSplit algorithm.
5. Note that
   - Lemmas 2 and 3 can be envisaged as a *power expansion* over $(n+1)^{-1}$ of the transition matrix.
   - The matrices appearing when $k = 2$ also appears for $k = 3$ (see lemmas 2 and 3).

   This suggest us a power expansion of the k-OneStep of the form

$$T_{n,k} = \left(1 + \frac{k}{n+1}\right) I + \frac{\gamma_1(k)}{n+1} \begin{pmatrix} -3 & 4 \\ 3 & -4 \end{pmatrix} + \frac{\gamma_2(k)}{(n+1)^2} \begin{pmatrix} 12 & -18 \\ -12 & 18 \end{pmatrix} + \frac{\gamma_3(k)}{(n+1)^3} \begin{pmatrix} -48 & 54 \\ 48 & -54 \end{pmatrix} + \cdots$$

   Moreover, a little bit of though suggest us $\gamma_i(k) = \binom{k}{i} \cdots$
6. The different coefficients appearing into the matrices reflect the behavior of the MacroSplit algorithm. We search for a precise meaning of this intuitive fact.

In the following we solve all these questions.

5 Behavior of the MacroSplit algorithms

In order to study the expected behavior of an x or y node belonging to a fringe of $n+1$ leaves when k keys are inserted at a given step, we need to know the characteristics of the MacroSplit algorithm we are using.

5.1 Local behavior

We would like to know how many 1-type and 2-type leaves are generated when i keys fall in the same step into a unique node x or y. To deal with this fact we introduce the following definition.

Definition 4. At the bottom level, the local behavior of the MacroSplit algorithm is given by the following functions:

- The $\mathcal{X}_x(i)$ is the number of 1-type leaves after the insertion of i keys into a unique x node (for instance, $\mathcal{X}_x(0) = 2$, $\mathcal{X}_x(1) = 0$, ...). In the same way, $\mathcal{X}_y(i)$ is the number of 1-type leaves after the insertion of i keys into an y node(for instance, $\mathcal{X}_y(0) = 0$, $\mathcal{X}_y(1) = 4$, ...).
- Dually, $\mathcal{Y}_x(i)$ is the number of 2-type leaves after the insertion of i keys into an x node. Finally, $\mathcal{Y}_y(i)$ is the number of 2-type leaves after the insertion of i keys into an y node.

These coefficients verify $\mathcal{X}_x(i) + \mathcal{Y}_x(i) = 2 + i$ and $\mathcal{X}_y(i) + \mathcal{Y}_y(i) = 3 + i$.

5.2 Distribution function

Assume that random k keys fall (in parallel) into a fringe having $n+1$ leaves. First of all, let us isolate just one bottom node x and one key to insert. Fixed x, it has two leaves, and one new key can be inserted into this node in two different positions (corresponding to the left of each leaf). Therefore just one key hits a node x with probability $\frac{2}{n+1}$.

Now we consider what happens with this node x when k keys are inserted. Let N_x the a random variable denoting the number of keys falling into a fixed bottom node x. As the set of k keys is random, this variable follows the binomial distribution,

$$P\{N_x = i\} = \binom{k}{i}\left(\frac{2}{n+1}\right)^i\left(1-\frac{2}{n+1}\right)^{k-i} = b\left(i,k,\frac{2}{n+1}\right)$$

such that $b\,(i,k,p) = \binom{k}{i}p^i\,(1-p)^{k-i}$. Recall that the expected value is kp.

5.3 Expected local behavior

The number of 1-type leaves generated by the keys falling into a unique node x is given by the random variable $X_x = \mathcal{X}_x(N_x)$. The expected number of 1-type leaves generated by one x bottom node when a batch of k keys is inserted into a fringe having $n+1$ leaves is:

$$E(X_x \mid k) = \sum_{i=0}^{k} P\{N_x = i\}\mathcal{X}_x(i) = \sum_{i=0}^{k} b\left(i,k,\frac{2}{n+1}\right)\mathcal{X}_x(i)$$

The number of 2-type leaves generated by the node x is $Y_x = \mathcal{Y}_x(N_x)$, then the expected value is is

$$E(Y_x \mid k) = \sum_{i=0}^{k} b\left(i,k,\frac{2}{n+1}\right)\mathcal{Y}_x(i)$$

Let us fix a bottom node y. In this case, one key hits this node with probability $\frac{3}{n+1}$. Let N_y be another random variable denoting the number of keys falling into this node y, clearly $P\{N_y = i\} = b\left(i,k,\frac{3}{n+1}\right)$. In this case we have the random variables $X_y = \mathcal{X}_y(N_y)$ and $Y_y = \mathcal{Y}_y(N_y)$ having the expected values

$$E(X_y \mid k) = \sum_{i=0}^{k} b\left(i,k,\frac{3}{n+1}\right)\mathcal{X}_y(i) \qquad E(Y_y \mid k) = \sum_{i=0}^{k} b\left(i,k,\frac{3}{n+1}\right)\mathcal{Y}_y(i)$$

Note that these expected values depend of the concrete local behavior of the algorithm. The expected number of leaves generated by just one bottom node when k random keys are inserted in parallel into a fringe having $n+1$ is:

$$E(X_x + Y_x \mid k) = 2(1 + \frac{k}{n+1}) \qquad \text{and} \qquad E(X_y + Y_y \mid k) = 3(1 + \frac{k}{n+1})$$

5.4 Global behavior

We relate the local behavior with the global one by means of the matrix transition:

Definition 5. Given a fringe with $n+1$ leaves and a MacroSplit algorithm, we define the k-OneStep transition matrix as:

$$T_{n,k} = \begin{pmatrix} \frac{1}{2}E(X_x \mid k) & \frac{1}{3}E(X_y \mid k) \\ \frac{1}{2}E(Y_x \mid k) & \frac{1}{3}E(Y_y \mid k) \end{pmatrix}$$

The proof of the following lemma is given in the appendix.

Lemma 6. *Given a fringe with X_t leaves of 1-type and Y_t leaves of 2-type, when k random keys are inserted into it in one step we have*

$$\begin{pmatrix} E(X_{t+1} \mid X_t, Y_t, k) \\ E(Y_{t+1} \mid X_t, Y_t, k) \end{pmatrix} = T_{n,k} \begin{pmatrix} X_t \\ Y_t \end{pmatrix}$$

The proof of the following theorem is given in the appendix.

Theorem 7. *When k random keys are inserted in one step we have:*

$$\begin{pmatrix} E(X_{t+1} \mid k) \\ E(Y_{t+1} \mid k) \end{pmatrix} = T_{n,k} \begin{pmatrix} E(X_t \mid k) \\ E(Y_t \mid k) \end{pmatrix}$$

From the note 5 of the section 4.2 in which we have conjectured a power expansion form for the transition matrix, it will be interesting to have a k-OneStep transition matrix (definition 5) like $T_{n,k} = \left(1 + \frac{k}{n+1}\right) I + \cdots$. We can prove:

Lemma 8. *Let I be the two dimensional identity matrix, the k-OneStep verifies:*

$$T_{n,k} = \left(1 + \frac{k}{n+1}\right) I + \begin{pmatrix} -\frac{1}{2}E(Y_x \mid k) & \frac{1}{3}E(X_y \mid k) \\ \frac{1}{2}E(Y_x \mid k) & -\frac{1}{3}E(X_y \mid k) \end{pmatrix}$$

5.5 Power expansion on the transition matrix

Let us recall the *binomial transform* $\mathcal{B}$ recently developed by P. Poblete, J. Munro and Th. Papadakis [PMP95]. Let $\langle F_i \rangle_{i \geq 0}$ be a sequence of real numbers, the binomial transform is the sequence $\langle \hat{F}_j \rangle_{j \geq 0}$ defined as

$$\hat{F}_j = \mathcal{B}_j F_i = \sum_{i=0}^{j} (-1)^i \binom{j}{i} F_i.$$

This transformation verifies $F_i = \mathcal{B}_i \hat{F}_j$. In the following we will use the following weighted form of the binomial transforms of $\langle \mathcal{Y}_x(i) \rangle_{i \geq 0}$ and $\langle \mathcal{X}_y(i) \rangle_{i \geq 0}$:

Definition 9. Let consider the coefficients $\alpha_j = -2^{j-1}\hat{\mathcal{Y}}_x(j)$ and $\beta_j = -3^{j-1}\hat{\mathcal{X}}_y(j)$.

Let us develop the relationship of the preceding coefficients with the local expected values of the k-OneStep appearing in the lemma 8. The proof of the following lemma is given in the appendix.

Lemma 10.

$$E(Y_x \mid k) = \mathcal{B}_k\left(\left(\frac{2}{n+1}\right)^j \mathcal{B}_j \mathcal{Y}_x(i)\right) = -2\sum_{j=0}^{k} \frac{(-1)^j}{(n+1)^j}\binom{k}{j}\alpha_j$$

$$E(X_y \mid k) = \mathcal{B}_k\left(\left(\frac{3}{n+1}\right)^j \mathcal{B}_j \mathcal{X}_y(i)\right) = -3\sum_{j=0}^{k} \frac{(-1)^j}{(n+1)^j}\binom{k}{j}\beta_j$$

From lemmas 6 and 10 we get the following expansion

Theorem 11. *The k-OneStep transition matrix can be rewritten as*

$$T_{n,k} = \left(1 + \frac{k}{n+1}\right) I + \sum_{j=0}^{k} \frac{(-1)^j}{(n+1)^j}\binom{k}{j} \begin{pmatrix} \alpha_j & -\beta_j \\ -\alpha_j & \beta_j \end{pmatrix},$$

Note that $T_{n,k} = T^{\mathsf{Seq}}_{n,k} + O\left(1/n^2\right)\ I$.

6 Two extreme MacroSplit algorithms

We have shown that the k-OneStep transition matrix depends of the concrete MacroSplit algorithm. In this section we develop two extreme cases of this algorithm: one denoted MaxMacroSplit algorithms that makes the maximum number of splits and creates the maximum number of x nodes and another denoted MinMacroSplit algorithm that makes the minimum number of splits and creates the maximum number of y nodes. These two extreme cases seems to bound the behavior of the whole pipeline in the W. Paul, U. Vishkin and H. Wagener [PVW83] insertion algorithm.

6.1 The MaxMacroSplit and MinMacroSplit algorithms

Assume that an even i number of keys are attached to a node x ($i = 6$ in the case 1 of the figure 3). This wide node splits by yielding $i+2$ 1-type leaves (8 in the preceding case) and 0 2-type leaves. Then $\mathcal{X}_x(i) = i+2$ and $\mathcal{Y}_x(i) = 0$. On the other hand, an odd number i of keys are attached ($i = 7$ in case 2 of the figure 3). In this case the split only creates one node y, then $\mathcal{Y}_x(i) = 3$ and $\mathcal{X}_x(i) = i-1$ (3 and 6 respectively in the figure). Note that $\mathcal{X}_x(i)+\mathcal{Y}_x(i) = i+2$. We summarize the previous paragraph into the following lemma.

Lemma 12. *The MaxMacroSplit algorithm has the following characterization: (1) The* local behavior *is given by:*

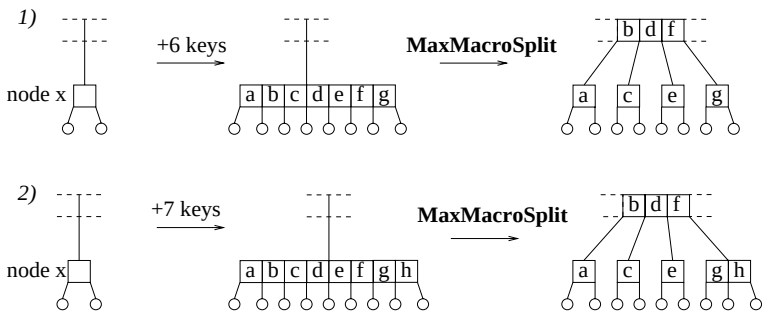


Fig. 3. Application of MaxMacroSplit rule on a node x

- *For even i we have* $\mathcal{X}_x(i) = i+2$, $\mathcal{Y}_x(i) = 0$, $\mathcal{X}_y(i) = i$, $\mathcal{Y}_y(i) = 3$.
- *For odd i we have* $\mathcal{X}_x(i) = i-1$, $\mathcal{Y}_x(i) = 3$, $\mathcal{X}_y(i) = i+3$, $\mathcal{Y}_y(i) = 0$.

(2) The expected local behavior *is*

$$E(X_x \mid k) = kp + \frac{1}{2} + \frac{3}{2}(q-p)^k \quad E(Y_x \mid k) = \frac{3}{2} - \frac{3}{2}(q-p)^k \quad \textit{for} \quad p = \frac{2}{n+1}$$
$$E(X_y \mid k) = kp + \frac{3}{2} - \frac{3}{2}(q-p)^k \quad E(Y_y \mid k) = \frac{3}{2} + \frac{3}{2}(q-p)^k \quad \textit{for} \quad p = \frac{3}{n+1}$$

(3) The power expansion *verifies* $\alpha_0 = \beta_0 = 0$, $\beta_1 = 4$. *For* $j > 0$ *we have* $\alpha_j = -3\,4^{j-1}$ *and for* $j > 1$ *we have* $\beta_j = -3\,6^{j-1}$.

When we consider a minimum number of splits we have the following

Lemma 13. *The MinMacroSplit algorithm has the following characterization: (1) The* local behavior *is given by:*

- *For* $i \mod 3 = 0$ *we have* $\mathcal{X}_x(i) = 2$, $\mathcal{Y}_x(i) = i$, $\mathcal{X}_y(i) = 0$, $\mathcal{Y}_y(i) = i+3$.
- *For* $i \mod 3 = 1$ *we have* $\mathcal{X}_x(i) = 0$, $\mathcal{Y}_x(i) = i+2$, $\mathcal{X}_y(i) = 4$, $\mathcal{Y}_y(i) = i-1$.
- *For* $i \mod 3 = 2$ *we have* $\mathcal{X}_x(i) = 4$, $\mathcal{Y}_x(i) = i-2$, $\mathcal{X}_y(i) = 2$, $\mathcal{Y}_y(i) = i+1$.

(2) Let be

$$\phi = \mathcal{R}e\left(\frac{2-3p+p\sqrt{3}\mathbf{i}}{2}\right)^k \qquad \textit{and} \qquad \varphi = \sqrt{3}\,\mathcal{I}m\left(\frac{2-3p+p\sqrt{3}\mathbf{i}}{2}\right)^k$$

The expected local behavior *is determined by:*

$$E(X_x \mid k) = 2 - \frac{4}{3}\varphi \qquad E(Y_x \mid k) = pk + \frac{4}{3}\varphi \qquad \textit{for} \quad p = \frac{2}{n+1}$$
$$E(X_y \mid k) = 2 - 2\phi + \frac{2}{3}\varphi \quad E(Y_y \mid k) = pk + 1 + 2\phi - \frac{2}{3}\varphi \quad \textit{for} \quad p = \frac{3}{n+1}.$$

(3) For $j > 2$, the power expansion coefficients of the MinMacroSplit algorithm verify $\alpha_{j+6} = 12^3\alpha_j$ *and* $\beta_{j+6} = 12^3\beta_j$

6.2 Relationship with PVW's algorithm

Let us see how the MaxSplit and MinSplit algorithms bound the fringe behavior of the insertion algorithm given in [PVW83]. On it, a *macro step* contains the whole insertion of the k keys. Let X_t^{PVW}, Y_t^{PVW} the fringe distribution before the pipeline starts and let be X_{t+1}^{PVW}, Y_{t+1}^{PVW} be the fringe once the pipeline has finished. A rough bound is given in the following conjecture.

Conjecture 14. *Let $X_t^{MaxSplit}$, $Y_t^{MaxSplit}$ be the fringe in the MaxMacroSplit algorithm. Let $X_t^{MinSplit}$, $Y_t^{MinSplit}$ be the fringe in the MinMacroSplit algorithm and Let X_t^{PVW}, Y_t^{PVW} be the fringe in the macro step algorithm of W. Paul, U. Vishkin and H. Wagener, we have:*

$$E(X_t^{MinSplit} \mid k) \le E(X_t^{PVW} \mid k) \le E(X_t^{MaxSplit} \mid k)$$
$$E(Y_t^{MaxSplit} \mid k) \le E(Y_t^{PVW} \mid k) \le E(Y_t^{MinSplit} \mid k)$$

7 Conclusion

We have analyzed the MacroSplit parallel insertion algorithms (Theorem 7) and we have proved that the coefficients of the k-OneStep, determining the global behavior of the algorithm, are given by the expected local behavior. We have developed the power expansion (theorem 11) proving that the MacroSplit algorithm can be approximated by the iterative sequential algorithm with an error determined by $O(1/n^2)$ (being n the size of the tree). The coefficients of the expansion are proportional to the binomial transform of the expected local behavior

We have conjectured (conjecture 14) that the [PVW83] algorithm is bounded by the two extreme algorithms MaxMacroSplit and MinMacroSplit and we have computed (lemmas 12 and 13) the main values of these algorithms. In the limiting case (very large trees) all these algorithms have the same performance.

Acknowledgement: We thank M. Sánchez Busquets to point out an error in an earlier version of the paper.

References

[BY95] R.A. Baeza-Yates. Fringe analysis revisited. *ACM Computing Surveys*, 27(1):109–119, 1995.

[EZG+82] B. Eisenbarth, N. Ziviani, G.H. Gonnet, K. Mehlhorn, and D. Wood. The theory of fringe analysis and its application to 2–3 trees and B-trees. *Information and Control*, 55(1-3):125–174, 1982.

[PMP95] P.V. Poblete, J.I. Munro, and T. Papadakis. The binomial transform and its application to the analysis of skip lists. In *ESA 95*, pages 1–10. Springer-Verlag, 1995.

[PVW83] W. Paul, U. Vishkin, and H. Wagener. Parallel dictionaries on 2–3 trees. In J. Díaz, editor, *Proc. 10th ICALP, LNCS 154*, pages 597–609. Springer-Verlag, 1983.

A Appendix

A.1 Proof of lemma 1

The conditional expectation $E(X_{t+1}|X_t, Y_t, 2)$ is:

$$\begin{aligned}
&\sum_{(\cdot,\cdot)} P(\cdot,\cdot)\ E(X_{t+1}|X_t, Y_t, 2, (.\ ,.)) \\
=&\frac{1}{(n+1)^2}\Big(2X_t(X_t+2) + X_t(X_t-2)(X_t-4) + 2X_tY_t(X_t+2) \\
&\qquad + 3Y_t(X_t+2) + Y_t(Y_t-3)(X_t+8)\Big) \\
=&X_t + \frac{1}{(n+1)^2}\Big(12X_t - 4X_t^2 + 4X_tY_t + 8Y_t^2 - 18Y_t\Big) \\
=&X_t + \frac{1}{(n+1)^2}\Big(12X_t - 4X_t(X_t+Y_t) + 8Y_t(X_t+Y_t) - 18Y_t\Big)
\end{aligned}$$

The Y_{t+1} term has a similar development.

A.2 Proof of lemma 6

Let us consider a fringe having X_t leaves of 1-type and Y_t leaves of 2-type and $X_t + Y_t = n + 1$. In this fringe, the number of x bottom nodes is $X_t/2$. The number of y bottom nodes is $Y_t/3$. Now we insert k random keys in just one step and we are interested in the value of X_{t+1}. Let $\mathcal{N}_{x,i}$ be the number of x nodes getting i keys and let $\mathcal{N}_{y,i}$ be the number of y nodes getting i keys. We have

$$X_{t+1} = \sum_{i=0}^{k} \mathcal{N}_{x,i} \cdot \mathcal{X}_x(i) + \sum_{i=0}^{k} \mathcal{N}_{y,i} \cdot \mathcal{X}_y(i)$$

Recall that X_t and Y_t are fixed. As an x node gets i keys with probability $P\{N_x = i\}$ and the number of x nodes is $\frac{1}{2}X_t$, the random variable $\mathcal{N}_{x,i}$ follows the binomial distribution

$$P\{\mathcal{N}_{x,i} = j \mid X_t, Y_t, k\} = b\Big(j, \frac{1}{2}X_t, P\{N_x = i\}\Big)$$

Then the expected number of x nodes receiving exactly i keys each one is $E(\mathcal{N}_{x,i} \mid X_t, Y_t, k) = P\{N_x = i \mid k\}\frac{X_t}{2}$. Similarly, the expected number of y nodes is $E(\mathcal{N}_{y,i} \mid X_t, Y_t, k) = P\{N_y = i \mid k\}\frac{Y_t}{3}$.

We study the expected behavior of X_{t+1} when k keys are inserted at random.

$$\begin{aligned}
E(X_{t+1} \mid X_t, Y_t, k) &= \sum_{i=0}^{k} E(\mathcal{N}_{x,i} \mid X_t, Y_t, k)\mathcal{X}_x(i) + \sum_{i=0}^{k} E(\mathcal{N}_{y,i} \mid X_t, Y_t, k)\mathcal{X}_y(i) \\
&= \frac{X_t}{2}\sum_{i=0}^{k} P\{N_x = i \mid k\}\mathcal{X}_x(i) + \frac{Y_t}{3}\sum_{i=0}^{k} P\{N_y = i \mid k\}\mathcal{X}_y(i) \\
&= E(X_x \mid k)\frac{X_t}{2} + E(X_y \mid k)\frac{Y_t}{3}
\end{aligned}$$

The computation for $E(Y_{t+1} \mid X_t, Y_t, k)$ is similar.

A.3 Proof of theorem 7

From the preceding lemma we have

$$E(X_{t+1} \mid X_t, Y_t, k) = E(X_x \mid k)\frac{X_t}{2} + E(X_y \mid k)\frac{Y_t}{3}$$

As $E(X_t + 1 \mid k) = E(E(X_{t+1} \mid X_t, Y_t, k) \mid k)$ we have

$$E(X_t + 1 \mid k) = \frac{1}{2}E(E(X_x \mid k)X_t \mid k) + \frac{1}{3}E(E(X_y \mid k)Y_t \mid k)$$

As X_x and X_t are independent $E(E(X_x \mid k)X_t \mid k) = E(X_x \mid k)E(X_t \mid k)$ and the proof is done.

A.4 Proof of lemma 10

Recall that

$$E(Y_x \mid \ell) = \sum_{\ell=0}^{k} P\{X_x = i\}\mathcal{Y}_x(i) = \sum_{\ell=0}^{k} \binom{k}{\ell}\Big(\frac{2}{n+1}\Big)^{\ell}\Big(1 - \frac{2}{n+1}\Big)^{k-\ell}\mathcal{Y}_x(i)$$

Consider the sequence $\langle E(Y_x \mid \ell)\rangle_{\ell \geq 0}$, given $p + q = 1$ the binomial transform verifies $\mathcal{B}_j \sum_i \binom{\ell}{i} p^i q^{\ell - i} F_i = p^j \hat{F}_j$ therefore:

$$\hat{E}(Y_x \mid j) = \mathcal{B}_j E(Y_x \mid \ell) = \Big(\frac{2}{n+1}\Big)^j \mathcal{B}_j \mathcal{Y}_x(\ell) = \Big(\frac{2}{n+1}\Big)^j \hat{\mathcal{Y}}_x(j)$$

Now we apply the property $F_k = \mathcal{B}_k \mathcal{B}_j F_\ell$ and

$$E(Y_x \mid k) = \mathcal{B}_k \hat{E}(Y_x \mid j) = \mathcal{B}_k \mathcal{B}_j E(Y_x \mid \ell) = \mathcal{B}_k\Big(\Big(\frac{2}{n+1}\Big)^j \mathcal{B}_j \mathcal{Y}_x(i)\Big)$$

Using linearity $\mathcal{B}_k \gamma F_j = \gamma \mathcal{B}_k F_j$ and $\alpha_j = -2^{j-1}\mathcal{B}_j \mathcal{Y}_x(i)$ we have

$$\begin{aligned} E(Y_x \mid k) &= 2\mathcal{B}_k\Big(\Big(\frac{1}{n+1}\Big)^j 2^{j-1}\mathcal{B}_j \mathcal{Y}_x(i)\Big) = 2\mathcal{B}_k\Big(\Big(\frac{1}{n+1}\Big)^j \alpha_j\Big) \\ &= -2\sum_{j=0}^{k}(-1)^j \binom{k}{j}\frac{\alpha_j}{(n+1)^j} \end{aligned}$$

The case $E(x_y \mid k)$ is quite similar.

On Balls and Bins with Deletions

Richard Cole[1], Alan Frieze[2], Bruce M. Maggs[3], Michael Mitzenmacher[4], Andréa W. Richa[5], Ramesh Sitaraman[6], and Eli Upfal[7]

[1] Courant Institute, New York University, New York, NY 10012.
cole@cs.nyu.edu
Supported by NSF grants CCR-9503309 and CCR-9800085.
[2] Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA.
af1p@andrew.cmu.edu
Supported by NSF grant CCR-9530974.
[3] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.
bmm@cs.cmu.edu
Supported in part by the Air Force Materiel Command. (AFMC) and ARPA under Contract F196828-93-C-0193, by ARPA Contracts F33615-93-1-1330 and N00014-95-1-1246, and by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute and Sun Microsystems. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, ARPA, CMU, or the U.S. Government.
[4] Compaq Systems Research Center, Palo Alto, CA 94301.
michaelm@pa.dec.com
[5] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
aricha@cs.cmu.edu
Supported in part by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute and Sun Microsystems.
[6] Department of Computer Science, University of Massachusetts, Amherst, MA 01003
ramesh@cs.umass.edu
Supported in part by an NSF CAREER Award No. CCR-97-03017.
[7] Department of Computer Science, Brown University
eli@cs.brown.edu
Supported in part by NSF grant CCR-9731477.

Abstract. We consider the problem of extending the analysis of balls and bins processes where a ball is placed in the least loaded of d randomly chosen bins to cover deletions. In particular, we are interested in the case where the system maintains a fixed load, and deletions are determined by an adversary before the process begins. We show that with high probability the load in any bin is $O(\log \log n)$. In fact, this result follows from recent work by Cole et al. concerning a more difficult problem of routing in a butterfly network.

The main contribution of this paper is to give a different proof of this bound, which follows the lines of the analysis of Azar, Broder, Karlin, and Upfal for the corresponding static load balancing problem. We also give a specialized (and hence simpler) version of the argument from the

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 145-158, 1998.

paper by Cole et al. for the balls and bins scenario. Finally, we provide an alternative analysis also based on the approach of Azar, Broder, Karlin, and Upfal for the special case where items are deleted according to their age. Although this analysis does not yield better bounds than our argument for the general case, it is interesting because it utilizes a two-dimensional family of random variables in order to account for the age of the items. This technique may be of more general use.

1 Introduction

A standard question in load balancing is to consider what the distribution of balls in bins looks like when m balls are thrown into n bins. In particular, when n balls are thrown into n bins, it is well known that the maximum load is approximately $\log n / \log \log n$ with high probability. The seminal paper of Azar, Broder, Karlin, and Upfal asked a related question: suppose the balls are placed sequentially, and each ball is placed in the least loaded of d bins chosen independently and uniformly at random [4]. In this case, they find that the maximum load is $\log \log n / \log d + O(1)$ with high probability; more detailed analysis of the distribution in this case is undertaken in [10]. This work has led to a number papers analyzing related load balancing schemes, including for example [1, 2, 5–9, 11, 12].

Note that the above result applies to a *static* problem, where a fixed number of balls are distributed. An interesting related question is to consider the *dynamic* situation where balls can be deleted as well as inserted into the system over time. Indeed, the original paper by Azar, Broder, Karlin, and Upfal examines the dynamic situation where at each step a random ball is deleted and re-inserted in the system [4]. Related work by, for example, Mitzenmacher [8, 9, 11] and Adler et al. [1] examines deletions via connections with queueing theoretical models.

Here we focus on a model where an adversary may specify a deletion sequence in advance. Our first and main result is to extend the proof of [4] to handle a polynomial length sequence of insertions and deletions, where the maximum load in the system is always at most n balls. We then note that an even more general result, in which re-insertions can occur, is already essentially contained in the results of [6] (a re-insertion causes a ball to choose among the same bins as on its first insertion). This work considered a similar problem related to routing on a butterfly network. We restate this proof for the balls and bins setting, where it becomes significantly simpler. Finally, we consider a special case in which deletions are always of the item that has been longest in the system. We again use a variant of the two-choice argument from [4], this time making use of a two-dimensional family of random variables, similar in spirit to the work of [11].

We emphasize that the interest of this work lies in the techniques used rather than the result, which is already implicit in the work of [6].

2 Adversarial deletions: polynomially many steps

In this section, we demonstrate that the original proof of Azar, Broder, Karlin, and Upfal in [4] can be extended to handle deletions under an appropriate adversarial model for polynomially many steps. We first define the underlying process.

For a vector $\boldsymbol{v} = (v_1, v_2, \ldots)$, let $P_d(\boldsymbol{v})$ be the following process: at time steps 1 through n, n balls are placed into n bins sequentially, with each ball going into the least loaded of d bins chosen independently and uniformly at random. After these balls are placed, deletions and insertions alternate, so that at each subsequent time step $n + j$, first the ball inserted at time v_j is removed, and then a new ball is placed into the least loaded of d bins chosen independently and uniformly at random. (Actually we do not require this alternation; the main point is that we have a bound, n, on the number of balls in the system at any point. The alternation merely makes notation more convenient.)

We assume the vector $\boldsymbol{v}$ is suitably defined so that at each step an actual deletion occurs; that is, the v_j are unique and $v_j \leq n + j - 1$. Otherwise $\boldsymbol{v}$ is arbitrary, although we emphasize that it is chosen before the process begins and does not depend on the random choices made during the process.

We adopt some of the notation of [4]. Each ball is assigned a fixed *height* upon entry, where the height is the number of balls in the bin, including itself. The height of the ball placed at time t is denoted by $h(t)$. The load of a bin at time t refers to the number of balls in the bin at that time. We let $\mu_{\geq k}(t)$ denote the number of balls that have height at least k at time t, and $\nu_{\geq k}(t)$ be the number of bins that have load at least k at time t. Note that if a bin has load k, it must contain some ball of height at least k. Hence $\mu_{\geq k}(t) \geq \nu_{\geq k}(t)$ for all times t. Finally, $B(n, p)$ refers to a binomially distributed random variable based on n trials each with probability p of success.

Before giving the proof, we sketch the main ideas. The flavor of the proof is to show that with high probability the number of bins containing at least i balls is doubly exponentially decreasing for sufficiently large i. The bound on the number of bins containing at least $i + 1$ balls is obtained from the bound on the number of bins containing at least i balls. Establishing the proper conditioning between the number of bins with i and $i + 1$ balls makes the proof challenging.

A key idea is to avoid seeking a direct bound on the number of bins containing at least i balls. Rather, following [4], we use the fact that the number of balls of height at least i bounds the number of bins containing at least i balls. This leads us to obtain bounds on the distribution of ball heights which, with high probability, hold for polynomially many steps. A concern is that here the adversarial choice of deletions might lead this bound to be too weak. On the other hand, the adversary is constrained, for the full sequence of deletions must be chosen up front, and this allows the result.

The key difference between our result on that of [4] is that they find a dominating distribution of heights on one set of n balls, whereas we use a distribution that applies to every set of n balls present in the system as it evolves. As it happens, the bounds are essentially the same; the most significant changes lie in

the end game, where we must bound the number of bins containing more than $\log\log n/\log d$ balls.

Theorem 1. *For any fixed constants c_1 and c_2, with probability at least $1 - o(1/n^{c_1})$ the maximum load of a bin achieved by process $P_d(\boldsymbol{v})$ over $T = n^{c_2}$ steps is* $\log\log n/\log d + O((c_1 + c_2)/d)$

Proof. The argument extends the original Theorem 4 of [4], by determining a distribution on the heights of the balls that holds for polynomially many steps, regardless of which n balls are in the system at any point in time.

Let $\mathcal{E}_i$ be the event that $\nu_{\geq i}(t) \leq \beta_i$ for time steps $t = 1, \ldots, T$, where the β_i will be revealed shortly. We want to show that at time t, $1 \leq t \leq T$,

$$\Pr(\mu_{\geq i+1} > \beta_{i+1} \mid \mathcal{E}_i)$$

is sufficiently small. That is, given $\mathcal{E}_i$, we want $\mathcal{E}_{i+1}$ to hold as well. This probability is hard to estimate directly. However, we know that since the d choices for a ball are independent, we have

$$\Pr(h(t) \geq i + 1 \mid \nu_{\geq i}(t-1)) = \frac{\big(\nu_{\geq i}(t-1)\big)^d}{n^d}.$$

We would like to bound for each time t the distribution of the number of time steps j such that $h(j) \geq i + 1$ and the ball inserted at time step j has not been deleted by time t. In particular, we would like to bound this distribution by a binomial distribution over n events with success probability $(\beta_i/n)^d$. But this is difficult to do directly as the events are not independent.

Instead, we fix i and define the binary random variables Y_t for $t = 1, \ldots, T$, where

$$Y_t = 1 \textbf{ iff } h(t) \geq i + 1 \textbf{ and } \nu_{\geq i}(t-1) \leq \beta_i.$$

(The value Y_t is 1 if and only if the height of the ball t is at least $i + 1$ despite the fact that the number of boxes that have load at least i is currently below β_i.)

Let ω_j represent the choices available to the j'th ball. Clearly

$$\Pr(Y_t = 1 \mid \omega_1, \ldots, \omega_{t-1}, v_1, \ldots, v_{t-n}) \leq \frac{\beta_i^d}{n^d} \stackrel{\text{def}}{=} p_i.$$

Consider the situation immediately after a time step t' where a new ball has entered the system. Then there are n balls in the system, that entered at times $u_1, u_2, \ldots, u_n$. Let $I(t')$ be the set of times $\{u_1, u_2, \ldots, u_n\}$. Then

$$\sum_{t \in I(t')} Y_t = \sum_{i=1}^{n} Y_{u_i};$$

that is, the summation over $I(t')$ is implicitly over the values of Y_t for the balls in the system at time t'. (This statement differs from the result of [4]; the important

point here is that we can bound $\sum_{t\in I(t')} Y_t$ regardless of what n balls are in the system. Note these balls are fixed for any time t by the deletion sequence $\boldsymbol{v}$.)

We may conclude that at any time $t' \leq T$

$$\Pr\left(\sum_{t\in I(t')} Y_t \geq k\right) \leq \Pr(B(n,p_i) \geq k). \tag{1}$$

Observe that conditioned on $\mathcal{E}_i$, we have $\mu_{\geq i+1}(t') = \sum_{t\in I(t')} Y_t$. Therefore

$$\Pr(\mu_{\geq i+1}(t') \geq k \mid \mathcal{E}_i) = \Pr\left(\sum_{t\in I(t')} Y_t \geq k \mid \mathcal{E}_i\right) \leq \frac{\Pr(B(n,p_i)\geq k)}{\Pr(\mathcal{E}_i)}. \tag{2}$$

Thus:

$$\Pr(\neg\mathcal{E}_{i+1} \mid \mathcal{E}_i) \leq \frac{T\Pr(B(n,p_i)\geq k)}{\Pr(\mathcal{E}_i)}$$

Since

$$\Pr(\neg\mathcal{E}_{i+1}) \leq \Pr(\neg\mathcal{E}_{i+1} \mid \mathcal{E}_i)\Pr(\mathcal{E}_i) + \Pr(\neg\mathcal{E}_i),$$

we have

$$\Pr(\neg\mathcal{E}_{i+1}) \leq T\Pr(B(n,p_i)\geq k) + \Pr(\neg\mathcal{E}_i). \tag{3}$$

We can bound large deviations in the binomial distribution with the formula (see for instance [3], Appendix A.)

$$\Pr(B(n,p_i) \geq ep_in) \leq e^{-p_in}. \tag{4}$$

We may then set $\beta_6 = \frac{n}{2e}$, and subsequently

$$\beta_i = \frac{e\beta_{i-1}^d}{n^{d-1}} \text{ for } i \geq 7.$$

Note that the β_i are chosen so that $\Pr(B(n,p_i)\geq\beta_{i+1}) \leq e^{-p_in}$.

With these choices $\mathcal{E}_6$ holds with certainty, as there cannot be more than $n/2e$ bins with 6 balls. For $i \geq 6$,

$$\Pr(\neg\mathcal{E}_{i+1}) \leq \frac{T}{n^{c_1+c_2+1}} + \Pr(\neg\mathcal{E}_i) \leq \frac{1}{n^{c_1+1}} + \Pr(\neg\mathcal{E}_i),$$

provided that $p_in \geq (c_1+c_2+1)\ln n$.

Let i^* be the smallest value for which $p_{i^*-1}n \leq (c_1+c_2+1)\ln n$. Note that $i^* \leq \ln\ln n/\ln d + O(1)$. Note that the preceding argument can not be used to bound the number of bins with height at least i^*, as the Chernoff bounds are no longer powerful enough; hence we must tackle the tail more directly. This requires some careful attention to detail. In fact we will show that, given $\mathcal{E}_{i^*-1}$, then with probability $1-o(1/n^{c_1})$, there are no balls with height $i^*+\lceil(c_1+c_2+2)/(d-1)\rceil+1$ over the course of the entire process.

Let $\mathcal{F}_1$ be the event that $\nu_{\geq i^*}(t) \leq (ec_1 + ec_2 + e) \ln n$ for all times $t \leq T$. In other words, at every time there are not too many bins with height at least i^*. Then

$$\Pr(\neg\mathcal{F}_1) \leq \Pr(\neg\mathcal{F}_1 \mid \mathcal{E}_{i^*-1}) \Pr(\mathcal{E}_{i^*-1}) + \Pr(\neg\mathcal{E}_{i^*-1}),$$

and again by (1), (2), and (4)

$$\Pr(\neg\mathcal{F}_1 \mid \mathcal{E}_{i^*-1}) \Pr(\mathcal{E}_{i^*-1}) \leq T \Pr(B(n, (c_1+c_2+1)\ln n/n) \geq (ec_1+ec_2+e)\ln n) \leq \frac{1}{n^{c_1+1}}.$$

Let $\mathcal{F}_2$ be the event that $\nu_{\geq i^*+1}(t) \leq c_1 + c_2 + 2$ for all times $t \leq T$. In other words, at every time there are no more than a constant number of bins with load at least $i^* + 1$. Again,

$$\Pr(\neg\mathcal{F}_2) \leq \Pr(\neg\mathcal{F}_2 \mid \mathcal{F}_1) \Pr(\mathcal{F}_1) + \Pr(\neg\mathcal{F}_1).$$

Here

$$\Pr(\neg\mathcal{F}_2 \mid \mathcal{F}_1) \Pr(\mathcal{F}_1) \leq T \Pr(B(n, ((ec_1 + ec_2 + e)\ln n/n)^d) \geq c_1 + c_2 + 2).$$

The binomial expression can be checked to be $o(1/n^{c_1+c_2+1})$, and hence this last term is also $O(1/n^{c_1+1})$.

Conditioned on $\mathcal{F}_2$, we must now show that throughout the process there are no bins with load $i^* + \lceil (c_1 + c_2 + 2)/(d-1) \rceil + 1$ with sufficiently high probability. Let this event be given by $\mathcal{G}$. Consider any specific time step z. For there to be be any bins with load $i^* + \lceil (c_1 + c_2 + 2)/(d-1) \rceil + 1$ at time z, at least $\lceil (c_1 + c_2 + 2)/(d-1) \rceil$ of the balls in the system must have landed in bins with height at least $i^* + 1$. Hence

$$\Pr(\neg\mathcal{G}) \leq \Pr(\neg\mathcal{G} \mid \mathcal{F}_2) \Pr(\mathcal{F}_2) + \Pr(\neg\mathcal{F}_2).$$

Here

$$\Pr(\neg\mathcal{G} \mid \mathcal{F}_2) \Pr(\mathcal{F}_2) \leq T \Pr\left(B(n, ((c_1 + c_2 + 2)/n)^d) \geq \lceil (c_1 + c_2 + 2)/(d-1) \rceil\right).$$

The binomial expression can be checked to be $o(1/n^{c_1+c_2+1})$, and hence this last term is also $O(1/n^{c_1+1})$.

To conclude (abusing notation), we have

$$\begin{aligned}
\Pr(\neg\mathcal{G}) &\leq O\left(\frac{1}{n^{c_1+1}}\right) + \Pr(\neg\mathcal{F}_2) \\
&\leq O\left(\frac{1}{n^{c_1+1}}\right) + \Pr(\neg\mathcal{F}_1) \\
&\leq O\left(\frac{1}{n^{c_1+1}}\right) + \Pr(\neg\mathcal{E}_{i^*-1}) \\
&\leq O\left(\frac{1}{n^{c_1+1}}\right) + \sum_{i=1}^{i^*-1} \frac{1}{n^{c_1+1}}.
\end{aligned}$$

Here the last bound comes from the recurrence (3). That is, $\Pr(\neg\mathcal{G})$ is dominated by the sum of $O(\log\log n)$ events each with with probability $O(1/n^{c_1+1})$; the theorem follows.

Note that the probability bounds of Theorem 1 can be improved by choosing a smaller value of i^*, so that the strong Chernoff bounds hold, and then increasing the maximum load allowed appropriately. Similarly, we can improve the theorem so that a superpolynomial number of steps can be handled, although this requires increasing the bound on the maximum load above $\log\log n/\log d + O(1)$ to for example $(1 + o(1))\log\log n/\log d$.

3 Adversarial deletions: a witness tree argument

In this section, we provide a simple witness tree argument for the balls-and-bins problem with deletions and re-insertions. A similar argument appears in [6] for the more difficult problem of routing circuits in a butterfly network. Therefore, our result is not new, in that it follows naturally from the argument in [6]. Rather, our goal is to present a self-contained and simplified version of the proof for the simpler balls-and-bins situation. For convenience, we focus on the case $d = 2$.

We consider a variation $Q_d(\boldsymbol{v}, \boldsymbol{w})$ of the process $P_d(\boldsymbol{v})$. Again, the process begins with n insertions, followed by alternating insertions and deletions, with $\boldsymbol{v}$ specifying the balls to be deleted. We now use $\boldsymbol{w}$ to represent insertions, however. We assign each ball an *identification number*, and without loss of generality we assume the first n balls have ID numbers 1 through n. At time $n + j$, the ball with ID number w_j is inserted. If this ball has never been inserted before, then it is placed in the least loaded of d bins chosen independently and uniformly at random. If the ball has been inserted before, it is placed in the least loaded of the d bins chosen for its first insertion – that is the bin choices of a ball are fixed after it is first inserted in the system. We assume that $\boldsymbol{v}$ and $\boldsymbol{w}$ are consistent, so there is only one ball with a given ID number in the system at a time. Note also that $\boldsymbol{v}$ and $\boldsymbol{w}$ must be chosen by the adversary before the process begins, without reference to the random choices made during the course of the process.

This scenario appears when, for example, we use a (random) hash function for the two bin choices of every ball. As before, when a ball is (re-)inserted, the algorithm places the ball in the bin with the smaller load.

The main theorem of this section is stated below. The constants of the theorem have been chosen for convenience and have not been optimized. Note that the techniques used to prove this theorem can be generalized to show that if each ball makes d bin choices for some constant d, then the maximum load of any bin is $O(\log\log n)$ with high probability. The result can also be extended for non-constant d as well.

Theorem 2. *At any time t, with probability at least $1 - 1/n^{\Omega(\log\log n)}$, the maximum load of a bin achieved by process $Q_2(\boldsymbol{v}, \boldsymbol{w})$ is $4\log\log n$.*

Proof. We prove the theorem in two parts. First, we show that if there is a bin r at time t with 4ℓ balls, where $\ell = \log\log n$, there exists a degree ℓ pruned witness tree. Next, we show that, with high probability, no degree ℓ pruned witness tree exists.

Constructing a witness tree. In a witness tree, each node represents a bin and each edge (r_i, r_j, t_e) represents a ball that was inserted at time t_e whose two bin choices are r_i and r_j. Suppose that some bin r has load 4ℓ at time t. We construct the witness tree as follows. The root of the tree corresponds to bin r. Let $b_1, \ldots, b_{4\ell}$ be the balls in r at time t. Let r_i be the other bin choice associated with ball b_i (one of the choices is bin r). The root r has 4ℓ children, one corresponding to each bin r_i. Let $t_i < t$ be the last time b_i was (re-)inserted into the system. Without loss of generality, assume that $t_1 < t_2 < \ldots < t_{4\ell}$. Note that the height of ball b_i when it was inserted at time t_i is at least i. Therefore, the load of bin r_i, the other choice of b_i, is at least $i - 1$ at time t_i. We use this fact to recursively grow a tree rooted at each r_i.

The witness tree we have described is irregular. However, it contains as a subgraph an ℓ-ary tree of height ℓ such that

- The root in level 0 has ℓ children that are internal nodes.
- Each internal node on levels 1 to $\ell - 2$ has two children that are internal nodes and $\ell - 2$ children that are leaves.
- Each internal node on level $\ell - 1$ has ℓ children that are leaves.

For convenience we refer to this subtree as the actual witness tree henceforth.

Constructing a pruned witness tree. If the nodes of the witness tree are guaranteed to represent distinct bins, proving our probabilistic bound is a relatively easy matter. However, this is not the case; a bin may reappear several times in a witness tree, leading to dependencies that are difficult to resolve. This makes it necessary to *prune* the tree so that each node in the tree represents a distinct bin. Consequently, the balls represented by the edges of the pruned witness tree are also distinct. In this regard, note that a ball appears at most once in a pruned witness tree, even if it was (re-)inserted multiple times in the sequence.

We visit the nodes of the witness tree iteratively in *breadth-first* search order starting at the root. As we proceed, we remove (i.e., prune) some nodes of the tree and the subtrees rooted at these nodes – what remains is the pruned witness tree. We start by visiting the root. In each iteration, we visit the next node v in breadth-first order that has not been pruned. Let $B(v)$ denote the nodes visited *before* v.

- If v represents a bin that is different from the bins represented by nodes in $B(v)$, we do nothing.
- Otherwise, prune all nodes in the subtree rooted at v. Then, we mark the edge from v to its parent as a *cutoff edge*.

Note that the cutoff edges are not part of the pruned witness tree. The procedure continues until either no more nodes remain to be visited or there are ℓ cutoff edges. In the latter case, we apply a final pruning by removing all nodes that are yet to be visited. The tree that results from this pruning process is the pruned witness tree. After the pruning is complete, we make a second pass through the tree and construct a set C of *cutoff balls*. Initially, C is set to $\emptyset$. We visit

the cutoff edges in BFS order and for each cutoff edge (u, v) we add the ball corresponding to (u, v) to C, if this ball is distinct from all balls currently in C and if $|C| \leq \lceil p/2 \rceil$, where p is the total number of cutoff edges.

Lemma 1. *The pruned witness tree constructed above has the following properties.*

1. *All nodes in the pruned witness represent distinct bins.*
2. *All edges in the pruned witness tree represent distinct balls. (Note that cutoff edges are not included in the pruned witness tree.)*
3. *The cutoff balls in C are distinct from each other, and from the balls represented in the pruned witness tree.*
4. *There are $\lceil p/2 \rceil$ cutoff balls in C, where p is the number of cutoff edges.*

Proof. The first three properties follow from the construction. We prove the fourth property as follows. Let b be a ball represented by some cutoff edge, and let v and w be its bin choices. Since v and w can appear at most once as nodes in the pruned witness tree, ball b can be represented by at most two cutoff edges. Thus, there are $\lceil p/2 \rceil$ distinct cutoff balls in C.

Enumerating pruned witness trees.

We bound the probability that a pruned witness tree exists by bounding both the number of possible pruned witness trees and the probability that each such tree could arise. First, we choose the shape of the pruned witness tree. Then, we traverse the tree in breadth-first order and bound the number of choices for the bins for each tree node and the balls for each tree edge; we also bound the associated probability that these choices came to pass. Finally, we consider the number of choices for cutoff balls in C and the corresponding probability that they arose. Multiplying these quantities together yields the final bound – it is important to note here that we can multiply term together only because all the balls and the bins in the pruned witness tree and the cutoff balls in C are distinct.

Choosing the shape of the pruned witness tree. Assume that there are p cutoff edges in the pruned tree. The number of ways of selecting the p cutoff edges is at most

$$\binom{\ell^2 2^\ell}{p} \leq \ell^{2p} 2^{\ell p},$$

since there are at most $\ell^2 2^\ell$ nodes in the pruned witness tree.

Ways of choosing balls and bins for the nodes and edges of the pruned witness tree. The enumeration proceeds by considering the nodes in BFS order. The number of ways of choosing the bin associated with the root is n. Assume that you are considering the i^{th} internal node v_i of the pruned witness tree whose bin has already been chosen to be r_i. Let v_i have δ_i children. We evaluate the number of ways of choosing a distinct bin for each of the δ_i children of v_i and choosing a distinct ball for each of the δ_i edges incident on v_i and weight it by multiplying by the appropriate probability. We call this product E_i.

There are at most $\binom{n}{\delta_i}$ ways of choosing distinct bins for each of the δ_i children of v_i. Also, since there are at most n balls in the system at any point in time, the number of ways to choose distinct balls for the δ_i edges incident on v_i is also at most $\binom{n}{\delta_i}$. (Note that the n balls in the system may be different for each v_i; however, there are still at most $\binom{n}{\delta_i}$ possibilities for the ball choices for any vertex.) There are $\delta_i!$ ways of pairing the balls and the bins, and the probability that a chosen ball chooses bin r_i and a specific one of δ_i bins chosen above is $2/n^2$. Thus,

$$E_i \leq \binom{n}{\delta_i}\binom{n}{\delta_i}\delta_i!\left(\frac{2}{n^2}\right)^{\delta_i} \leq 2^{\delta_i}/\delta_i!. \tag{5}$$

Let m be number of internal nodes v_i in the pruned witness tree such that $\delta_i = \ell$. Using the bound in Equation 5 for only these m nodes, the number of ways of choosing the bins and balls for the nodes and edges respectively of the pruned witness tree weighted by the probability these choices occurred is at most $n \cdot (2^\ell/\ell!)^m$.

Ways of choosing the cutoff balls in C. Using Lemma 1, we know that there are $\lceil p/2 \rceil$ distinct cutoff balls in C. The number of ways of choosing the balls in C is at most $n^{\lceil p/2 \rceil}$, since at any time step there are at most n balls in the system to choose from. Note that a cutoff ball has both its bin choices in the pruned witness tree. Therefore, the probability that a given ball is a cutoff ball is at most

$$\binom{\ell^2 2^\ell}{2}\frac{2}{n^2} \leq \ell^4 2^{2\ell}/n^2.$$

Thus the number of choices for the $\lceil p/2 \rceil$ cutoff balls in C weighted by the probability these cutoff balls occurred is at most

$$n^{\lceil p/2 \rceil}(\ell^4 2^{2\ell}/n^2)^{\lceil p/2 \rceil} \leq (\ell^4 2^{2\ell}/n)^{\lceil p/2 \rceil}.$$

Putting it all together. The probability at time t of there existing a pruned witness tree with p cutoff edges, and m internal nodes with $\ell = \log\log n$ children, is at most

$$\begin{aligned} \ell^{2p}2^{\ell p} \cdot n \cdot (2^\ell/\ell!)^m \cdot (\ell^4 2^{2\ell}/n)^{\lceil p/2 \rceil} &\leq n \cdot (2^\ell/\ell!)^m \cdot (\ell^8 2^{4\ell}/n)^{\lceil p/2 \rceil} \\ &\leq n \cdot (2e/\log\log n)^{m \log\log n} \cdot (\log\log^8 n \log^4 n/n)^{\lceil p/2 \rceil}. \end{aligned} \tag{6}$$

Observe that either the number the cutoff edges, p, equals ℓ or the number of internal nodes with ℓ children, m, is at least $2^{\ell-2} = \log n/4$. Thus, in either case, the bound in Equation 6 is $1/n^{\Omega(\log\log n)}$. Further, since there are at most $\ell^2 2^\ell$ values for p, the total probability of a pruned witness tree is at most $\ell^2 2^\ell \cdot 1/n^{\Omega(\log\log n)}$ which is $1/n^{\Omega(\log\log n)}$. This completes the proof of the theorem.

4 Deletions based on item age

We now consider an alternative scenario, where items are deleted in order of their insertion time. We define a process P'_d based on phases: in the first phase,

there are n insertions, where again all insertions are made by putting a ball into the least loaded of d bins chosen independently and uniformly at random. In subsequent phases, there are first n insertions, and then the items inserted in the previous phase are deleted.

One way to view this process is as a modified version of the process $P_d(\boldsymbol{v})$ in the case where $\boldsymbol{v} = (1, 2, 3, \ldots)$. The difference here is that deletions and insertions do not alternate; instead, deletion requests are batched and acted on at the end of a phase. (Unfortunately, we do not yet have an argument showing bounds no this modified version of $P_d((1, 2, \ldots))$ necessarily also hold on the original process, even though it seems natural to conclude that batching deletions until a later time can only worsen performance.)

This phase-based system allows us to regard the state of each bin as a two-dimensional variable. A bin is said to be in state (i, j) if it has i balls that will be deleted in the next deletion phase and at least j balls that have been inserted in the current insertion phase. Such two-dimensional models have previously proven useful for dynamic variations of load balancing problems [11]. We prove bounds for this phased-based system; for convenience we consider only the case $d = 2$.

Theorem 3. *For any fixed constant c, the maximum load of a bin achieved by process $P_2'(1, 2, \ldots)$ over $T = n^c$ steps is $\log\log n/\log 2 + O(1)$ with high probability.*

Proof (Sketch). We vary Theorem 1 so that it works on phases. That is, consider a time interval of n insertions preceding a deletion phase. Let $X_{i,j}(t)$ be the number of bins with i balls that will be deleted and at least j newly inserted balls after the tth insertion. Note that $X_{i,j}(0) = 0$ unless $j = 0$.

Our goal is to show there is a simple "stable" bounding distribution with the following property: if we begin with $X_{i,0}(0) \le \beta_i$ for some appropriate sequence β_i, then after n insertions and n deletions, we again have $X_{i,0}(0) \le \beta_i$ with suitably high probability. This will imply that the process continues for polynomially many steps before the load becomes too high, assuming we begin properly.

Suppose

$$X_{i,0}(0) \le \frac{\alpha n}{i}\gamma^{2^i}$$

for sufficiently large $i \ge L$, where L and γ are suitable constants. (For example, we may take $L = 20$, $\alpha = 1/20$, and $\gamma = (1 - \frac{1}{2^L})$; note $\gamma^{2^L} \approx \frac{1}{e}$.) It can be checked that this condition holds after the first n insertions in a straightforward manner. We will show that

$$X_{i,j}(n) \le \frac{\alpha n}{i+j}\gamma^{2^{i+j}}$$

for $i + j \ge L$ with high probability.

Further, let $\hat{X}_{i,0} = \sum_{k=0}^{\infty} X_{k,i}(n)$; $\hat{X}_{i,0}$ is just the number of bins with i balls after the insertions and deletions complete. We will also show that

$$\hat{X}_{i,0}(0) \le \frac{\alpha n}{i}\gamma^{2^i}$$

for $i \geq L$ with high probability; this will allow us to continue the process for polynomially many steps. (Actually, technically we only require these conditions hold for up to the point where $i+j$ is $\log\log n/\log 2 + O(1)$, so that there are still $\Omega(\log n)$ bins of this height. Once the number of bins at a state becomes sufficiently small so that Chernoff bounds no longer apply, we must use a more explicit tail argument, as in Theorem 1. This affects the $O(1)$ term of the theorem, which depends on c. We skip these details here.)

We prove this inductively on $i+j$ in a similar fashion to the induction in Theorem 1. Define

$$y_{i,j} \stackrel{\text{def}}{=} \frac{\alpha}{i+j}\gamma^{2^{i+j}}.$$

Let $\mathcal{E}_g$ be the event that $X_{i,j}(n) \leq y_{i,j}$ for all $i+j=g$. Now for a fixed pair $(i,j+1)$ with $i+j=g$ consider a series of binary random variables Y_t for $t=1,\ldots,n$, where $Y_t=1$ iff the tth ball lands in a bin in state $(i,j+1)$ (after its entry) and $\mathcal{E}_g$. (The value Y_t is 1 if the height of the ball t is at least $i+j+1$ and i balls in its bin are to deleted, despite the fact that the number of such bins has not grown too large.)

Let ω_i represent the choices available to the ith ball. Then

$$\Pr(Y_t = 1 \mid \omega_1,\ldots,\omega_{t-1}) \leq y_{i,j}^2 + 2\sum_{\substack{k+l=g\\(k,l)\neq(i,j)}} y_{i,j}y_{k,l} + 2\sum_{k>g} y_{i,j}y_{k,0} = P_{i,j}.$$

We simplify the above expression:

$$\begin{aligned} P_{i,j} &\leq \left(\frac{\alpha^2\gamma^{2^{i+j+1}}}{(i+j)^2}\right)\left(1+2(i+j)+2\sum_{k>g}\gamma^{2^k-2^{i+j}}\right)\\ &\leq 3\alpha\left(\frac{\alpha\gamma^{2^{i+j+1}}}{i+j}\right). \end{aligned}$$

In the last inequality, we bounded the last summation using the specified value for γ and the fact that $g \geq L$.

By the Chernoff bound $\sum_{t=1}^{n} Y_t \leq \frac{1}{2}ny_{i,j}$ with probability at least $1-1/n^{2c+2}$ as long as $P_{i,j} \geq (2c+2)\ln n$. But here $\sum_{t=1}^{n} Y_t = X_{i,j}(n)$ conditioned on $\mathcal{E}_g$. Hence, as long as $i+j=o(\log n)$,

$$\Pr(\neg\mathcal{E}_{g+1} \mid \mathcal{E}_g) = o\left(\frac{1}{n^{c+1}}\right).$$

Similarly, conditioned on all $\mathcal{E}_g$ for $g \geq L$, for $i \geq L+1$,

$$\begin{aligned} \hat{X}_{i,0} &= \sum_{k\geq 0} X_{k,i}(n)\\ &\leq \sum_{k\geq 0}\frac{\alpha\gamma^{2^{i+k}}}{2(i+k)}\\ &\leq \frac{\alpha}{i}\gamma^{2^i}. \end{aligned}$$

Hence, with high probability, the inductive argument goes through, handling all levels of height at least $L+1$. As the probability of failure is $o(\frac{1}{n^{c+1}})$ in any phase, we can go through n^c phases before a failure with high probability.

Note, however, than in using the assumption that the previous phase was well bounded to obtain the bound for the next one, we have "lost" the appropriate bound for $\hat{X}_{L,0}$. This is because to bound $\hat{X}_{L,0}(0)$ we would need to have an initial bound on $X_{L-1,0}(0)$, which we lack. This problem can be easily handled by noting that the number of bins with at least L balls at any point in the process is stochastically dominated by the number of bins with at least L balls when $4n$ bins are thrown into the n bins; this is because we could consider the distribution when each of the at most $2n$ balls in the system has a twin, and a ball or its twin goes into each of the two bins a ball chooses from. This distribution clearly dominates the distribution present in the system. For $L = 20$ and the parameters chosen we can conclude that at the beginning of every phase $\mathcal{E}_L$ holds with probability exponentially small in n, so the inductive proof goes through.

As for Theorem 1, the probability of failure can be made lower by increasing the bound on the maximum load appropriately.

References

1. M. Adler, P. Berenbrink, and K. Schröder. Analyzing an infinite parallel job allocation process. To appear in *ESA 98*.
2. M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995, pp. 238–247.
3. N. Alon and J. H. Spencer. **The Probabilistic Method**. John Wiley and Sons, 1992.
4. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, 1994, pp. 593–602.
5. P. Berenbrink, F. Meyer auf der Heide, and K. Schröder. Allocating weighted jobs in parallel. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997, pp. 302–310.
6. R. Cole, B. M. Maggs, F. Meyer auf der Heide, M. Mitzenmacher, A. W. Richa, K. Schröder, R. K. Sitaraman, and B. Vöcking. Randomized Protocols for Low-Congestion Circuit Routing in Multistage Interconnection Networks. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 378–388.
7. A. Czumaj and V. Stemann. Randomized allocation processes. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, 1997, pp. 194–203.
8. M. Mitzenmacher. Density dependent jump markov processes and applications to load balancing. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, 1996, pp. 213–223.
9. M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997, pp. 292–301.

10. M. Mitzenmacher. Studying balanced allocations with differential equations. Technical Note 1997-024, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, October 1997.
11. M. Mitzenmacher, How useful is old information? In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, 1997, pp. 83-91. Extended version available as Digital Systems Research Center Technical Note 1998-003.
12. V. Stemann. Parallel balanced allocations. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996, pp. 261–269.

"Balls into Bins" — A Simple and Tight Analysis

Martin Raab and Angelika Steger

Institut für Informatik
Technische Universität München
D-80290 München
{raab|steger}@informatik.tu-muenchen.de

Abstract. Suppose we sequentially throw m balls into n bins. It is a natural question to ask for the maximum number of balls in any bin. In this paper we shall derive sharp upper and lower bounds which are reached with high probability. We prove bounds for all values of $m(n) \geq n/\text{polylog}(n)$ by using the simple and well-known method of the first and second moment.

1 Introduction

Suppose that we sequentially throw m balls into n bins by placing each ball into a bin chosen independently and uniformly at random. It is a natural question to ask for the maximum number of balls in any bin. This very simple model has many applications in computer science. Here we name just two of them:

Hashing: The balls-into-bins model may be used to analyze the efficiency of hashing-algorithms. In the case of the so called *separate chaining*, all keys that hash to the same location in the table are stored in a linked list. It is clear that the lengths of these lists are a measure for the complexity. For a well chosen hash-function (*i.e.* a hash-function which assigns the keys to all locations in the table with the same probability), the lengths of the lists have exactly the same distribution as the number of balls in a bin.

Online Load Balancing: With the growing importance of parallel and distributed computing the load balancing problem has gained considerable attention during the last decade. A typical application for online load balancing is the following scenario: consider n database-servers and m requests which arise independently at different clients and which may be handled by any server. The problem is to assign the requests to the servers in such a way that all servers handle (about) the same number of requests.
Of course, by introducing a central dispatcher one can easily achieve uniform load on the servers. However, within a distributed setting the use of such a central dispatcher is highly undesired. Instead randomized strategies have been applied very successfully for the development of good and efficient load balancing

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 159-170, 1998.

algorithms. In their simplest version each request is assigned to a server chosen independently and uniformly at random. If all requests are of the same size the maximum load of a server then corresponds exactly to the maximum number of balls in a bin in the balls-into-bins model introduced above.

1.1 Previous results

Balls and bins games have been intensively studied in the literature, cf. e.g. [JK77]. The estimation of the maximum number of balls in any bin was originally mainly studied within the context of hashing functions. In particular, GONNET [Gon81] determined for the case $m = n$ that the expected number of balls in the bin containing the maximum number of balls is $\Gamma^{-1}(n)(1 + O\left(\frac{1}{\log \Gamma^{-1}(n)}\right))$. One can check that Gonnet's result implies that the maximum load of any bin is with high probability $\frac{\log n}{\log \log n}(1 + o(1))$. In his dissertation MITZENMACHER [Mit96] also included a simpler proof of the fact that the maximum load is $\Theta(\frac{\log n}{\log \log n})$. He also obtains some results for the case $m < n/\log n$. For the case $m \geq n \log n$ it was well known that the maximum load of any bin is $\Theta(\frac{m}{n})$, *i.e.* of the order of the mean. However, the precise deviation from the mean seems not to have been studied before.
We note that for the online load balancing also different models of balls into bin games have been studied. We note in particular the approach of AZAR et al. [ABKU92]. They study the following model: each ball picks d bins uniformly at random and places itself in those bin containing fewest balls. For the case $m = n$ [ABKU92] showed that in this model the maximum load of any bin drops exponentially from $\frac{\log n}{\log \log n}(1+o(1))$ to $\frac{\log \log n}{\log d}(1+o(1))$. Compare also CZUMAJ and STEMANN [CS97] for more results in this direction.

1.2 Our results

In this paper we apply the first and second moment method, a well-known tool within the theory of random graphs, cf. *e.g.* [Bol85], to obtain a straightforward proof of the fact that the maximum number of balls in a bin is $\frac{\log n}{\log \log n}(1+o(1))$ for $m = n$ with probability $1 - o(1)$.
Besides being a lot more elementary than GONNET's proof method the big advantage of our method is that it also easily generalizes to the case where $m \neq n$ balls are placed into n bins In particular, this allows to also analyze the case $m \gg n$, which can neither be handled by GONNET's approach nor by MITZENMACHER's. (Both are based on approximating the Binomial distribution $B(m, \frac{1}{n})$ by a Poisson distribution, which only gives tight bounds if $m \cdot \frac{1}{n}$ is a constant.) The case $m \gg n$ is particularly important for the load-balancing scenario mentioned above. Here it *e.g.* measures how the unsymmetry between different servers grows over time when more and more requests arrive.
Our results are summarized in the following theorem:

Theorem 1. *Let M be the random variable that counts the maximum number of balls in any bin, if we throw m balls independently and uniformly at random into n bins. Then* $\Pr[M > k_\alpha] = o(1)$ *if* $\alpha > 1$ *and* $\Pr[M > k_\alpha] = 1 - o(1)$ *if* $0 < \alpha < 1$, *where*

$$k_\alpha = \begin{cases} \frac{\log n}{\log \frac{n \log n}{m}} \left(1 + \alpha \frac{\log^{(2)} \frac{n \log n}{m}}{\log \frac{n \log n}{m}}\right), & \text{if } \frac{n}{\operatorname{polylog}(n)} \leq m \ll n \log n, \\ (d_c - 1 + \alpha) \log n, & \text{if } m = c \cdot n \log n \text{ for some constant } c, \\ \frac{m}{n} + \alpha \sqrt{2 \frac{m}{n} \log n}, & \text{if } n \log n \ll m \leq n \cdot \operatorname{polylog}(n), \\ \frac{m}{n} + \sqrt{\frac{2m \log n}{n} \left(1 - \frac{1}{\alpha} \frac{\log^{(2)} n}{2 \log n}\right)}, & \text{if } m \gg n \cdot (\log n)^3. \end{cases}$$

Here d_c denotes a suitable constant depending only on c, cf. the proof of Lemma 3.

The paper is organized as follows: in § 2 we give a brief overview of the first and second moment method, in § 3 we show how to apply this method within the balls-into-bins scenario and obtain in § 4 the $\frac{\log n}{\log \log n}(1 + o(1))$ bound for $m = n$. In § 5 we then present some more general tail bounds for Binomial random variables and combine them with the first and second moment method to obtain a proof of Theorem 1.

1.3 Notations

Throughout this paper m denotes the number of balls and n the number of bins. The probability that a ball is thrown into a fixed bin is given by $p := 1/n$. We define q by $q := 1 - p$. We shall denote the iterated log by $\log^{(\cdot)}$, *i.e.* $\log^{(1)} x = \log x$ and $\log^{(k+1)} x = \log(\log^{(k)} x)$ for all $k \geq 1$. In this paper logarithms are to the base e.
Asymptotic notations ($O(\cdot)$, $o(\cdot)$ and $\omega(\cdot)$) are always with respect to n; $f \ll g$ means $f = o(g)$ and $f \gg g$ means $f = \omega(g)$. We use the term $\operatorname{polylog}(x)$ to denote the class of functions $\bigcup_{k \geq 1} O\left((\log x)^k\right)$. We say that an event $\mathcal{E}$ occurs with high probability if $\Pr[\mathcal{E}] = 1 - o(1)$.

2 The first and second moment method

Let X be a non-negative random variable. Then Markov's inequality implies that $\Pr[X \geq 1] \leq \mathrm{E}[X]$. Hence, we have

$$\mathrm{E}[X] = o(1) \qquad \Longrightarrow \qquad \Pr[X = 0] = 1 - o(1). \tag{1}$$

Furthermore, Chebyshev's inequality implies that

$$\Pr[X = 0] \leq \Pr[|X - \mathrm{E}[X]| \geq \mathrm{E}[X]] \leq \frac{\operatorname{Var}[X]}{(\mathrm{E}[X])^2} = \frac{\mathrm{E}[X^2]}{(\mathrm{E}[X])^2} - 1.$$

Hence, in order to show that $\Pr[X = 0] = o(1)$ we just have to verify that

$$\mathrm{E}\left[X^2\right] = (1 + o(1))(\mathrm{E}[X])^2. \tag{2}$$

While it is often quite tedious to verify (2), it is relatively easy if we can write X as the sum of (not necessarily independent) 0-1 variables $X_1, \ldots, X_n$ that satisfy

$$\mathrm{E}[X_j] = \mathrm{E}[X_1] \quad \text{and} \quad \mathrm{E}[X_i X_j] \leq (1 + o(1))(\mathrm{E}[X_1])^2 \quad \forall 1 \leq i < j \leq n. \tag{3}$$

Then

$$\mathrm{E}\left[X^2\right] = \mathrm{E}\left[\left(\sum_{i=1}^{n} X_i\right)^2\right] = \mathrm{E}\left[\sum_i \underbrace{X_i^2}_{=X_i} + \sum_{i \neq j} X_i X_j\right] \leq \mathrm{E}[X] + (1+o(1))(\mathrm{E}[X])^2$$

and we can combine (1) and (2) to obtain

$$\Pr[X = 0] = \begin{cases} 1 - o(1), & \text{if } \mathrm{E}[X] = o(1), \\ o(1), & \text{if } \mathrm{E}[X] \to \infty. \end{cases} \tag{4}$$

This is the form which we will use for the analysis of the balls and bins scenario.

3 Setup for the analysis

Let $Y_i = Y_i(m, n)$ be the random variable which counts the number of balls in the ith bin if we throw m balls independently and uniformly at random into n bins. Clearly, Y_i is a binomially distributed random variable: we express this fact by writing $Y_i \sim B(m, 1/n)$ respectively $\Pr[Y_i = k] = b(k; m, 1/n) := \binom{m}{k} (\frac{1}{n})^k (1 - \frac{1}{n})^{m-k}$. Let $X_i = X_i(m, n, \alpha)$ be the random variable, which indicates if Y_i is at least $k_\alpha = k_\alpha(m, n)$ (the function from Theorem 1) and let $X = X(m, n, \alpha)$ be the sum over all X_i's, *i.e.*:

$$X := \sum_{i=1}^{n} X_i \quad \text{and} \quad X_i := \begin{cases} 1, & \text{if } \; Y_i \geq k_\alpha, \\ 0, & \text{otherwise.} \end{cases}$$

Clearly,

$$\mathrm{E}[X_i] = \Pr[B(m, 1/n) \geq k_\alpha] \qquad \text{for all } i = 1, \ldots, n,$$

and

$$\mathrm{E}[X] = n \cdot \Pr[B(m, 1/n) \geq k_\alpha]. \tag{5}$$

In order to apply (4) we therefore need good bounds for the tail of the binomial distribution. Before we obtain these for the general case of all $m = m(n)$ we consider the special case $m = n$.

4 The case $m = n$

The aim of this section is to present a self contained proof of the fact that if $m = n$ the maximum number of balls in a bin is $\frac{\log n}{\log^{(2)} n}(1 + o(1))$ with high probability. We will do this by showing that

$$\Pr\left[\exists \text{ at least one bin with } \geq \alpha\tfrac{\log n}{\log^{(2)} n} \text{ balls}\right] = \begin{cases} 1 - o(1), & \text{if } 0 < \alpha < 1, \\ o(1), & \text{if } \alpha > 1. \end{cases} \tag{6}$$

Note that the claim of equation (6) is slightly weaker than the corresponding one from Theorem 1. We consider this case first, as here the calculations stay slightly simpler. So, in the rest of this section we let $k_\alpha := \alpha\frac{\log n}{\log^{(2)} n}$.
Recall from § 2 that in order to do this we only have to show that condition (3) is satisfied for the random variables X_i introduced in the previous section and that

$$\mathrm{E}[X] = n \cdot \Pr\left[B(n, \tfrac{1}{n}) \geq \alpha\tfrac{\log n}{\log^{(2)} n}\right] \to \begin{cases} \infty, & \text{if } 0 < \alpha < 1, \\ 0, & \text{if } \alpha > 1. \end{cases} \tag{7}$$

The fact that $\mathrm{E}[X_i] = \mathrm{E}[X_1]$ for all $1 \leq i \leq n$ follows immediately from the definition of the X_i's. The proof of the second part of (3) is deferred to the end of this section. Instead we start with the verification of (7). For that we prove a small lemma on the binomial distribution. We state it in a slightly more general form then necessary, as this version will be helpful later-on.

Lemma 1. *Let $p = p(m)$ depend on m. Then for all $h \geq 1$*

$$\Pr[B(m,p) \geq mp + h] = \left(1 + O\left(\frac{mp}{h}\right)\right) \cdot b(mp + h; m, p).$$

Proof. Observe that for all $k \geq mp + h$:

$$\frac{b(k+1; m, p)}{b(k; m, p)} = \frac{(m-k)p}{(k+1)(1-p)} \leq \frac{((1-p)m - h)p}{(mp + h + 1)(1-p)} =: \lambda.$$

One easily checks that $\lambda < 1$ for $h \geq 1$. Thus

$$\sum_{k \geq mp+h} b(k; m, p) \leq b(mp + h; m, p) \cdot \sum_{i \geq 0} \lambda^i = \frac{1}{1-\lambda} \cdot b(mp + h; m, p).$$

As $\frac{1}{1-\lambda} \leq 1 + \frac{mp}{h}$ the claim of the lemma follows. □

We apply Lemma 1 for "m" $= n$, "p" $= \frac{1}{n}$ and "$mp + h$" $= k_\alpha$. Subsequently, we use STIRLING's formula $x! = (1 + o(1))\sqrt{2\pi x}e^{-x}x^x$ to estimate the binomial coefficient. Together we obtain:

$$\begin{aligned} \mathrm{E}[X] = n \cdot \Pr\left[B(n, \tfrac{1}{n}) \geq k_\alpha\right] &= n \cdot (1 + o(1)) \cdot b(k_\alpha; n, \tfrac{1}{n}) \\ &= n \cdot (1 + o(1)) \binom{n}{k_\alpha} \left(\frac{1}{n}\right)^{k_\alpha} \left(1 - \frac{1}{n}\right)^{n - k_\alpha} \end{aligned}$$

$$
\begin{aligned}
&= n \cdot (1+o(1)) \frac{1}{e\sqrt{2\pi k_\alpha}} \left(\frac{e}{k_\alpha}\right)^{k_\alpha} \\
&= n \cdot e^{\alpha \frac{\log n}{\log^{(2)} n} \cdot \left(1 - \log \alpha - \log^{(2)} n + \log^{(3)} n + o(1)\right)} \\
&= n^{1-\alpha+o(1)},
\end{aligned}
$$

which implies the statement of equation (7).
To complete the proof for the case $m = n$ we still have to verify that $\mathrm{E}[X_i X_j] \leq (1+o(1))(\mathrm{E}[X_1])^2$ for all $i \neq j$. In order to keep the proof elementary we shall proceed similarly as in the proof of Lemma 1. A more elegant version can be found in § 5.

$$
\begin{aligned}
\mathrm{E}[X_i X_j] &= \Pr[Y_i \geq k_\alpha \wedge Y_j \geq k_\alpha] \\
&= \sum_{k_1=k_\alpha}^{n-k_\alpha} \sum_{k_2=k_\alpha}^{n-k_1} \binom{n}{k_1} \underbrace{\binom{n-k_1}{k_2}}_{\leq \binom{n}{k_2}} \left(\frac{1}{n}\right)^{k_1+k_2} \underbrace{\left(1-\frac{2}{n}\right)^{n-(k_1+k_2)}}_{\leq \left(1-\frac{1}{n}\right)^2} \\
&\leq \sum_{k_1=k_\alpha}^{n} \sum_{k_2=k_\alpha}^{n} \binom{n}{k_1} \binom{n}{k_2} \left(\frac{1}{n}\right)^{k_1+k_2} \left(1-\frac{1}{n}\right)^{2n-2(k_1+k_2)} \\
&\leq \left[b(k_\alpha; n, \frac{1}{n}) \left(1-\frac{1}{n}\right)^{-k_\alpha} \sum_{i=0}^{\infty} \lambda^i \right]^2
\end{aligned}
$$

where λ is defined as $\lambda := \frac{b(k_\alpha+1; n, \frac{1}{n})(1-\frac{1}{n})^{k_\alpha}}{b(k_\alpha; n, \frac{1}{n})(1-\frac{1}{n})^{k_\alpha+1}}$. As $\lambda = o(1)$ and $b(k_\alpha; n, \frac{1}{n}) = (1+o(1))\mathrm{E}[X_1]$ (cf. Lemma 1) this concludes the proof of (6).

5 The general case

For the proof of Theorem 1 we will follow the same pattern as in the proof of the previous section. The main difference is that in various parts we need better bounds. We start by collecting some bounds on the tails of the binomial distribution.

5.1 Tails of the binomial distribution

The binomial distribution is very well studied. In particular it is well-known that the binomial distribution $B(m,p)$ tends to the normal distribution if $0 < p < 1$ is a fixed constant and m tends to infinity. If on the other hand $p = p(m)$ depends on m in such a way that mp converges to a constant λ for m tending to infinity, then the corresponding binomial distribution $B(m,p)$ tends to the Poisson distribution with parameter λ. For these two extreme cases also very good bounds on the tail of the binomial distributions are known. In the context of our "balls and bins" scenario, however, we are interested in the whole spectrum of values $p = p(m)$.

In this section we collect some bounds on the tails of the binomial distribution which are tailored to the proof of Theorem 1.
For values of $p(m)$ such that mp tends to infinity one can analyze the proof of the theorem of DeMoivre–Laplace to get asymptotic formulas for $\Pr[B(m,p) \geq mp + h]$ for all values h that are not "too" large:

Theorem 2 (DeMoivre–Laplace). *Assume $0 < p < 1$ depends on n such that $pqm = p(1-p)m \to \infty$ for $m \to \infty$. If $0 < h = x(pqm)^{1/2} = o((pqm)^{2/3})$ and $x \to \infty$ then*

$$\Pr[B(m,p) \geq mp + h] = (1 + o(1)) \cdot \frac{1}{x\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

For an explicit proof of this version of the DeMoivre–Laplace Theorem see *e.g.* [Bol85].
The probability that a binomial distributed random variable $B(m,p)$ obtains a value of size at least $mp(1+\epsilon)$ for some constant $\epsilon > 0$ is usually estimated using the so-called Chernoff bounds. Recall, however, that Chernoff bounds provide only an upper bound.
With the help of Lemma 1 in the previous section we are now in the position to prove the tail bounds for those special cases of the binomial distribution which we will need further-on.

Lemma 2. *a) If $mp + 1 \leq t \leq (\log m)^\ell$, for some positive constant ℓ, then*

$$\Pr[B(m,p) \geq t] = e^{t(\log mp - \log t + 1) - mp + O\left(\log^{(2)} m\right)}.$$

b) If $t = mp + o\left((pqm)^{\frac{2}{3}}\right)$ and $x := \frac{t-mp}{\sqrt{pqm}}$ tends to infinity, then

$$\Pr[B(m,p) \geq t] = e^{-\frac{x^2}{2} - \log x - \frac{1}{2}\log 2\pi + o(1)}.$$

Proof. a) Using Stirling's formula $x! = (1 + o(1))\sqrt{2\pi x} e^{-x} x^x$ we obtain:

$$b(t; m, p) = (1 + o(1)) \frac{1}{\sqrt{2\pi t}} \left(\frac{mp}{t}\right)^t \left(1 + \frac{t - mp}{m - t}\right)^{m-t}.$$

Together with Lemma 1 we thus get for $\log \Pr[B(m,p) \geq t]$ the following expression:

$$\log\left(1 + O\left(\tfrac{mp}{t-mp}\right)\right) + t(\log mp - \log t + 1) - mp - \tfrac{\log t}{2} + O\left(\tfrac{(t-mp)^2}{m-t}\right) - O(1)$$

The term $O\left(\frac{(t-mp)^2}{m-t}\right)$ gets arbitrarily small if $mp \leq t = o(\sqrt{m})$ because $\frac{(t-mp)^2}{m-t} \leq \frac{t^2}{m(1-o(1))} = o(1)$. By assumption $mp + 1 \leq (\log m)^\ell$. That is, $\log(1 + O(mp)) = O(\log \log m)$.
b) This case is simply a reformulation of the DeMoivre–Laplace theorem. □

5.2 Proof of Theorem 1

We follow the setup outlined in § 2. That is, we only have to show that the variables X_i satisfy condition (3) and that the expectation of $X = \sum X_i$ tends either to infinity or to zero depending on the fact whether α is smaller or greater than 1. We start with the later.

Lemma 3. *Let k_α be defined as in Theorem 1. Then*

$$\log \mathrm{E}[X] \to \begin{cases} \infty, & \text{if } 0 < \alpha < 1, \\ -\infty, & \text{if } \alpha > 1, \end{cases}$$

for all values $m = m(n) \geq n/\mathrm{polylog}(n)$.

Proof. The case $\frac{n}{\mathrm{polylog}(n)} \leq m \ll n \log n$.
We first note that it suffices to consider non-negative α's. Assume that $m = \frac{n \log n}{g}$ where $g = g(n)$ tends to infinity arbitrarily slowly and $g(n) \leq \mathrm{polylog}(n)$. Then

$$k_\alpha = \frac{\log n}{\log g}\left(1 + \alpha \frac{\log^{(2)} g}{\log g}\right).$$

From equation (5) and Lemma 2 case a) (we leave it to the reader to verify that this case may be applied) it follows that

$$\begin{aligned} \log \mathrm{E}[X] &= \log n + k_\alpha \left(\log^{(2)} n - \log g - \log k_\alpha + 1\right) - \frac{\log n}{g} + O\left(\log^{(2)} m\right) \\ &= \frac{\log n}{\log g}\left(\log g + \left(1 + \alpha \frac{\log^{(2)} g}{\log g}\right)\left(1 - \log g + \log^{(2)} g\right) + o(1)\right) \\ &= (1 - \alpha + o(1)) \frac{\log n \cdot \log^{(2)} g}{\log g}, \end{aligned}$$

which yields the desired result.

The case $m = c \cdot n \log n$.
Let $k_\alpha := (d_c - 1 + \alpha)$. By Lemma 2 we get:

$$\log \mathrm{E}[X] = \log n \left(1 + (d_c - 1 + \alpha)\left(\log c - \log\left(d_c - 1 + \alpha\right) + 1\right) - c + o(1)\right).$$

As a consequence, for $\alpha = 1$ $\log \mathrm{E}[X]$ is exactly then $o(\log n)$ when d_c is a solution of

$$f_c(x) := 1 + x\left(\log c - \log x + 1\right) - c = 0.$$

For all $c > 0$ this equation admits exactly two real zeros x_1, x_2. One of these solutions is smaller than c and is therefore not the one we are looking for. That is, we define d_c as the (unique) solution of $f_c(x) = 0$ that is greater than c. In the neighborhood of the solutions x_1 and x_2, $f_c(x)$ changes its sign. This means that for $d_c - 1 + \alpha$ for a given $\alpha > 1$, $\log \mathrm{E}[X]$ tends to $-\infty$, whereas for $d_c - 1 + \alpha$ for an $0 < \alpha < 1$, $\log \mathrm{E}[X]$ tends to ∞.

The case $n \log n \ll m \leq n \cdot \text{polylog}(n)$.
Assume that $m = gn \log n$ where $g = g(n) \leq \text{polylog}(n)$ tends to infinity arbitrarily slowly. Then

$$k_\alpha = g \log n \left(1 + \alpha \sqrt{\frac{2}{g}}\right).$$

From Lemma 2 case a) it follows that

$$\begin{aligned}
\log \mathrm{E}[X] &= \log n + k_\alpha \left(\log g + \log^{(2)} n - \log k_\alpha + 1\right) - g \log n + O\left(\log^{(2)} n\right) \\
&= g \log n \left(\frac{1}{g} + \left(1 + \alpha \sqrt{\frac{2}{g}}\right)\left(1 - \alpha \sqrt{\frac{2}{g}} + \frac{\alpha^2}{g} + o\left(\frac{1}{g}\right)\right) - 1 + o\left(\frac{1}{g}\right)\right) \\
&= \log n \left(1 - \alpha^2 + o(1)\right).
\end{aligned}$$

One easily to checks, that we didn't hurt the conditions of Lemma 2.

The case $m \gg n(\log n)^3$.
For this case we shall use the theorem of DeMoivre–Laplace. Recall that in this case

$$k_\alpha = \frac{m}{n} + \sqrt{\frac{2m \log n}{n}\left(1 - \frac{1}{\alpha} \frac{\log^{(2)} n}{2 \log n}\right)}.$$

Using the notations of Lemma 2 case b) we set

$$x := \frac{k_\alpha - mp}{\sqrt{pqm}} = \sqrt{2 \log n \left(1 - \frac{1}{2\alpha} \frac{\log^{(2)} n}{\log n}\right)\left(1 + \frac{1}{n-1}\right)}.$$

Applying DeMoivre–Laplace we obtain:

$$\begin{aligned}
\log \mathrm{E}[X] &= \log n - \frac{x^2}{2} - \log x - \log \sqrt{2\pi} + o(1) \\
&= \log^{(2)} n \cdot \left(\frac{1}{2\alpha} - \frac{1}{2} + o(1)\right).
\end{aligned}$$

We still need to check that we didn't violate the conditions of DeMoivre–Laplace, *i.e.* that $k_\alpha - \frac{m}{n} = o\left((pqm)^{\frac{2}{3}}\right)$, but this is true if $\frac{m}{n \log n} = \omega\left(\log^2 n\right)$. □

In order to show that the variables X_i satisfy the second part of condition (3) (note that the first part is trivially true) we start with two simple lemmas.

Lemma 4. *Let* $p \leq \frac{1}{4}$ *and* m *be such that* $p^2 m = o(1)$. *Then*

$$\Pr\left[B(m(1-p), \tfrac{p}{1-p}) \geq t\right] \leq (1 + o(1)) \cdot \Pr[B(m,p) \geq t] \qquad \textit{for all } 0 \leq t \leq m.$$

Proof. We will show that for all $1 \leq t \leq m$ we have $b(t; m(1-p), \frac{p}{1-p}) \leq (1 + o(1))b(t; m, p)$. Clearly, this then completes the proof of the lemma. So consider an arbitrary, but fixed $1 \leq t \leq m$. For $t > m(1-p)$ we have $b(t; m(1-p), \frac{p}{1-p}) = 0$ so we might as well assume that $t \leq m(1-p)$. Then

$$\begin{aligned}
b(t; m(1-p), \frac{p}{1-p}) &= \binom{m(1-p)}{t} \cdot \left(\frac{p}{1-p}\right)^t \cdot \left(1 - \frac{p}{1-p}\right)^{m(1-p)-t} \\
&= \binom{m}{t} \cdot \prod_{i=0}^{t-1} \underbrace{\frac{m(1-p)-i}{m-i}}_{\leq 1-p} \cdot \left(\frac{p}{1-p}\right)^t \cdot \underbrace{\left(1 - \frac{p}{1-p}\right)^{m(1-p)}}_{\leq (1-p)^m} \cdot \left(1 - \frac{p}{1-p}\right)^{-t} \\
&\leq \binom{m}{t} \cdot p^t \cdot (1-p)^{m-t} \cdot \left(\frac{1-p}{1-\frac{p}{1-p}}\right)^t \\
&= b(t; m, p) \cdot \underbrace{\left(1 + \frac{p^2}{1-2p}\right)^t}_{\leq e^{2p^2 m}} \\
&= b(t; m, p) \cdot (1 + o(1)).
\end{aligned}$$

□

Lemma 5. *Let $p = o(1)$ and m, t be such that $x := \frac{t-mp}{\sqrt{mp(1-p)}}$ satisfies $x \to \infty$, $x = o((mp(1-p))^{1/6})$ and $xp = o(1)$. Then*

$$\Pr\left[B(m(1-p), \tfrac{p}{1-p}) \geq t\right] \leq (1 + o(1))\Pr\left[B(m, p) \geq t\right].$$

Proof. Observe that the assumptions of the lemma are such that we may apply case b) of Lemma 2 to compute $\Pr[B(m,p) \geq t]$. Observe furthermore that we may also apply this case of Lemma 2 to bound $\Pr\left[B(m(1-p), \frac{p}{1-p}) \geq t\right]$, as here the corresponding x-value is

$$\bar{x} = \frac{t - m(1-p) \cdot \frac{p}{1-p}}{\sqrt{m(1-p) \cdot \frac{p}{1-p} \cdot (1 - \frac{p}{1-p})}} = \frac{t-mp}{\sqrt{mp(1-p)}} \cdot \sqrt{\frac{1-p}{1-\frac{p}{1-p}}} = x \cdot \sqrt{1 + \frac{p^2}{1-2p}}.$$

Together we deduce

$$\begin{aligned}
\Pr\left[B(m(1-p), \frac{p}{1-p}) \geq t\right] &= e^{-\frac{x^2}{2}(1+\frac{p^2}{1-2p}) - \log x - \frac{1}{2}\log(1+\frac{p^2}{1-2p}) - \frac{1}{2}\log 2\pi + o(1)} \\
&= \Pr\left[B(m,p) \geq t\right] \cdot e^{-O(p^2x^2) - O(p^2) + o(1)} \\
&= \Pr\left[B(m,p) \geq t\right] \cdot (1 + o(1)).
\end{aligned}$$

□

Corollary 1. *Let $m = m(n)$ and $p = \frac{1}{n}$ be such that $m \geq \log n$, and let k_α denote the value from Theorem 1. Then*

$$\Pr\left[B(m(1-p), \tfrac{p}{1-p}) \geq k_\alpha\right] \leq (1+o(1)) \cdot \Pr\left[B(m,p) \geq k_\alpha\right].$$

Proof. One easily checks that for all $m = o(n^2)$ Lemma 4 applies and that for all $m \gg n(\log n)^3$ Lemma 5 applies. □

Lemma 6. *Let $X_1, \ldots, X_n$ be defined as in § 3. Then for all $1 \leq i < j \leq n$*

$$\mathrm{E}\left[X_i X_j\right] \leq (1+o(1)) \cdot \left(\mathrm{E}\left[X_1\right]\right)^2.$$

Proof. Using the notation from § 3 we have

$$\begin{aligned}
\mathrm{E}\left[X_i X_j\right] &= \Pr\left[Y_i \geq k_\alpha \wedge Y_j \geq k_\alpha\right] \\
&= \sum_{\substack{k_1+k_2 \leq m \\ k_1,k_2 \geq k_\alpha}} \binom{m}{k_1}\binom{m-k_1}{k_2} p^{k_1+k_2}\,(1-2p)^{m-k_1-k_2} \\
&= \sum_{k_1=k_\alpha}^{m-k_\alpha} \binom{m}{k_1} p^{k_1}(1-p)^{m-k_1} \cdot \sum_{k_2=k_\alpha}^{m-k_1} \binom{m-k_1}{k_2} \left(\tfrac{p}{1-p}\right)^{k_2} \left(1-\tfrac{p}{1-p}\right)^{m-k_1-k_2} \\
&= \sum_{k_1=k_\alpha}^{m-k_\alpha} \binom{m}{k_1} p^{k_1}(1-p)^{m-k_1} \cdot \Pr\left[B(m-k_1, \tfrac{p}{1-p}) \geq k_\alpha\right].
\end{aligned}$$

As $k_1 \geq k_\alpha \geq mp$ we observe that

$$\begin{aligned}
\Pr\left[B(m-k_1, \tfrac{p}{1-p}) \geq k_\alpha\right] &\leq \Pr\left[B(m(1-p), \tfrac{p}{1-p}) \geq k_\alpha\right] \\
&= (1+o(1)) \cdot \Pr\left[B(m,p) \geq k_\alpha\right],
\end{aligned}$$

where the last equality follows from Corollary 1. Hence,

$$\begin{aligned}
\mathrm{E}\left[X_i X_j\right] &= \sum_{k_1=k_\alpha}^{m-k_\alpha} \binom{m}{k_1} p^{k_1}(1-p)^{m-k_1} \cdot (1+o(1)) \cdot \Pr\left[B(m,p) \geq k_\alpha\right] \\
&= (1+o(1)) \cdot \Pr\left[B(m,p) \geq k_\alpha\right] \cdot \underbrace{\sum_{k_1=k_\alpha}^{m-k_\alpha} \binom{m}{k_1} p^{k_1}(1-p)^{m-k_1}}_{\leq \Pr[B(m,p) \geq k_\alpha]} \\
&\leq (1+o(1)) \cdot \left(\Pr\left[B(m,p) \geq k_\alpha\right]\right)^2.
\end{aligned}$$

As $\Pr\left[B(m,p) \geq k_\alpha\right] = \mathrm{E}\left[X_1\right]$, this completes the proof of the lemma. □

6 Conclusion

In this paper we derived an asymptotic formula for the maximum number of balls in any bin, if $m = m(n)$ balls are thrown randomly into n bins, for all values of $m \geq n/\text{polylog}(n)$. Our proof is based on the so-called first and second moment.
The result for $m = n$ was well-known before. However, our method gave a much simpler proof compared to those which were previously available in the literature. To the best of our knowledge the result for the case $m \gg n$ is new. In our opinion it is a challenging open problem to study the behavior of the modified balls into bins scenario as introduced in [ABKU92] for the case $m \gg n$ as well. Intensive computational experiments seem to indicate that in this case the difference of the maximum load in any bin from the mean m/n should be independent of m. We intend to settle this problem in a forthcoming paper.

References

[ABKU92] Y. Azar, A.Z. Broder, A.R. Karlin, and E. Upfal. On-line load balancing (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science*, pages 218–225, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[Bol85] B. Bollobás. *Random graphs.* Academic Press, New York-San Francisco-London-San Diego, 1985.

[CS97] A. Czumaj and V. Stemann. Randomized allocation processes. In *38th Annual Symposium on Foundations of Computer Science*, pages 194–203, 1997.

[Gon81] G.H. Gonnet. Expected length of the longest probe sequence in hash code searching. *J. ACM*, 28(2):289–304, 1981.

[JK77] N. Johnson and S. Kotz. *Urn Models and Their Applications.* John Wiley and Sons, 1977.

[Mit96] M.D. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing.* PhD thesis, Computer Science Department, University of California at Berkeley, 1996.

Tornado Codes: Practical Erasure Codes Based on Random Irregular Graphs

Michael Luby

International Computer Science Institute and Digital Fountain, Inc.

Abstract. We introduce Tornado codes, a new class of erasure codes. These randomized codes have linear-time encoding and decoding algorithms. They can be used to transmit over lossy channels at rates extremely close to capacity. The encoding and decoding algorithms for Tornado codes are both simple and faster by orders of magnitude than the best software implementations of standard erasure codes. We expect Tornado codes will be extremely useful for applications such as reliable distribution of bulk data, including software distribution, video distribution, news and financials distribution, popular web site access, database replication, and military communications.

Despite the simplicity of Tornado codes, their design and analysis are mathematically interesting. The design requires the careful choice of a random irregular bipartite graph, where the structure of the irregular graph is extremely important. We model the progress of the decoding algorithm by a simple AND-OR tree analysis which immediately gives rise to a polynomial in one variable with coefficients determined by the graph structure. Based on these polynomials, we design a graph structure that guarantees successful decoding with high probability.

This talk is based on:

"Practical Loss-Resilient Codes", STOC '97, Michael Luby, Michael Mitzenmacher, Amin Shokrollahi, Dan Spielman, Volker Stemann.

and

"Analysis of Random Processes via And-Or Tree Evaluation", SODA '98, Michael Luby, Michael Mitzenmacher, Amin Shokrollahi

and

"A Digital Fountain Approach to Reliable Distribution of Bulk Data", SIGCOMM '98, John Byers, Michael Luby, Michael Mitzenmacher, Ashu Rege

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 171, 1998.

Using Approximation Hardness to Achieve Dependable Computation*

Mike Burmester[1], Yvo Desmedt[1,2], Yongge Wang[3]

[1] Department of Mathematics, Royal Holloway – University of London, Egham, Surrey TW20 OEX, UK, m.burmester@rhbnc.ac.uk.
[2] Center for Cryptography, Computer and Network Security, Department of EE & CS, University of Wisconsin – Milwaukee, P.O. Box 784, WI 53201 Milwaukee, USA, desmedt@cs.uwm.edu.
[3] Department of EE & CS, University of Wisconsin – Milwaukee, P.O. Box 784, WI 53201 Milwaukee, USA, wang@cs.uwm.edu.

Abstract. Redundancy has been utilized to achieve fault tolerant computation and to achieve reliable communication in networks of processors. These techniques can only be extended to computations solely based on functions in one input in which redundant hardware or software (servers) are used to compute intermediate and end results. However, almost all practical computation systems consist of components which are based on computations with multiple inputs. Wang, Desmedt, and Burmester have used AND/OR graphs to model this scenario. Roughly speaking, an AND/OR graph is a directed graph with two types of vertices, labeled $\wedge$-*vertices* and $\vee$-*vertices*. In this case, processors which need all their inputs in order to operate could be represented by $\wedge$-vertices, whereas processors which can choose one of their "redundant" inputs could be represented by $\vee$-vertices. In this paper, using the results for hardness of approximation and optimization problems, we will design dependable computation systems which could defeat as many malicious faults as possible. Specifically, assuming certain approximation hardness result, we will construct k-connected AND/OR graphs which could defeat a ck-active adversary (therefore a ck-passive adversary also) where $c > 1$ is any given constant. This result improves a great deal on the results for the equivalent communication problems.

1 Introduction

Redundancy has been utilized to achieve reliability, for example to achieve fault tolerant computation and to achieve reliable communication in networks of processors. One of the primary objectives of a redundant computation system is

* Research supported by DARPA F30602-97-1-0205. However the views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advance Research Projects Agency (DARPA), the Air Force, of the US Government.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 172-186, 1998.

to tolerate as many faults (accidental or malicious) as possible. Hence, one of the crucial requirements in designing redundant computation systems is to use the least resources (redundancy) to achieve dependable computation against the most powerful adversaries. It has been proven (see, e.g., Hadzilacos [15], Dolev [10], Dolev, Dwork, Waarts, and Yung [11], and Beimel and Franklin [4]) that in the presence of a k-passive adversary (respectively k-active adversary) the processors in a network can communicate reliably if and only if the network is $k+1$-connected (respectively $2k+1$-connected).

All these works mentioned above assume processors with one type of input, while in practice it is often the case that processors need more than one type of inputs. For example, for the national traffic control system, we need data from the aviation, rail, highway, and aquatic vehicles, conduits, and support systems by which people and goods are moved from a point-of-origin to a destination point in order to support and complete matters of commerce, government operations, and personal affairs. In addition, each component of the traffic control system is again a system consisting of computations with multiple inputs, e.g., the processors of the aviation control system need data from several sources such as the airplane's speed, current position, etc., to determine the airplane's next position. Wang, Desmedt, and Burmester [22] have used AND/OR graphs to model this scenario. Originally AND/OR graphs have been used in the context of artificial intelligence to model problem solving processes (see [17]). Roughly speaking, an AND/OR graph is a directed graph with two types of vertices, labeled $\wedge$-*vertices* and $\vee$-*vertices*. The graph must have at least one input (source) vertex and one output (sink) vertex. In this case, processors which need all their inputs in order to operate could be represented by $\wedge$-vertices, whereas processors which can choose (using some kind of voting procedure) one of their "redundant" inputs could be represented by $\vee$-vertices. A solution graph, which describes a valid computation of the system, is a minimal subgraph of an AND/OR graph with the following properties: If an $\wedge$-vertex is in the solution graph then all of its incoming edges (and incident vertices) belong to the solution graph; If an $\vee$-vertex is in the solution graph then exactly one of its incoming edges (and the incident vertex) belongs to the solution graph. Wang, Desmedt, and Burmester [22] showed that it is **NP**-hard to find vertex disjoint solution graphs in an AND/OR graph (though there is a polynomial time algorithm for finding vertex disjoint paths in networks of processors with one type of inputs). This result shows that in order to achieve dependable computation, the computation systems (networks of processors) must be designed in such a way that it is easy for the honest stations/agents to find the redundant information in the systems. A similar analysis as for the case of networks of processors with one type of inputs shows that in the presence of a k-passive adversary (respectively k-active adversary) the computation system modeled by an AND/OR graph is dependable if and only if the underlying graph (that is, the AND/OR graph) is $k+1$-connected (respectively $2k+1$-connected) and both the input vertices and the output vertex know the set of vertex disjoint solution graphs in the AND/OR graph. Later in

this paper, we will use $G_{\wedge\vee}$ to denote AND/OR graphs and G to denote standard undirected graphs unless specified otherwise.

What happens if we want to tolerate more powerful adversaries? Adding more channel is costly, so we suggest a simpler solution: designing the AND/OR graph in such a way that it is hard for the adversary to find a vertex separator of the maximum set of vertex disjoint solution graphs (that is, find at least one vertex on each solution graph in the maximum set of vertex disjoint solution graphs), whence the adversary does not know which processors to block (or control). In order to achieve this purpose, we need some stronger results for approximation and optimization problems. There have been many results (see, e.g., [1, 21] for a survey) for hardness of approximating an **NP**-hard optimization problem within a factor c from "below". For example, it is hard to compute an independent set[1] V' of a graph $G(V, E)$ (note that here G is a graph in the standard sense instead of being an AND/OR graph) with the property that $|V'| \geq \frac{k}{c}$ for some given factor c, where k is the size of the maximum independent set of G. But for our problem, we are more concerned with approximating an **NP**-hard optimization problem from "above". For example, given a graph $G(V, E)$, how hard is it to compute a vertex set V' of G with $|V'| \leq ck$ such that V' *contains* an optimal independent set of G, where k is the size of the optimal independent set of G? We show that this kind of approximation problem is also **NP**-hard. Then we will use this result to design dependable computation systems such that with k redundant computation paths we can achieve dependable computation against a ck-active (Byzantine style) adversary (therefore against a ck-passive adversary also), where $c > 1$ is any given constant. This result improves a great deal on the equivalent communication problems (see our discussion on related works below).

The organization of this paper is as follows. We first prove in Section 2 the following result: For any given constant $c > 1$, it is **NP**-hard to compute a vertex set V' of a given graph $G(V, E)$ with the properties that $|V'| \leq ck$ and V' contains an optimal independent set of $G(V, E)$, where k is the size of the optimal independent set of $G(V, E)$. Section 3 surveys a model for fault tolerant computation and describes the general threats to dependable computation systems. In Section 4 we demonstrate how to use AND/OR graphs with trap-doors to achieve dependable computation against passive (and active) adversaries. In Section 5 we outline an approach to build AND/OR graphs with trap-doors. We conclude in Section 6 with remarks towards practical solutions and we present some open problems.

Related work

Achieving processor cooperation in the presence of faults is a major problem in distributed systems. Popular paradigms such as Byzantine agreement have been studied extensively. Dolev [10] (see also, Dolev, Dwork, Waarts, and Yung [11]) showed that a necessary condition for achieving Byzantine agreement is that the

[1] An independent set in a graph $G(V, E)$ is a subset V' of V such that no two vertices in V' are joined by an edge in E.

number of faulty processors in the system is less than one-half of the connectivity of the system's network (note that in order to achieve Byzantine agreement, one also needs that $n > 3k$ where n is the number of processors in the network and k is the number of faulty processors). Hadzilacos [15] has shown that even in the absence of malicious failures connectivity $k + 1$ is required to achieve agreement in the presence of k faulty processors. Beimel and Franklin [4] have shown that if authentication techniques are used, then Byzantine agreement is achievable only if the graph of the underlying network is $k+1$ connected and the union of the authentication graph and the graph of the underlying network is $2k + 1$ connected in the presence of k faulty processors. All these works assume processors with one type of inputs. Recently, Wang, Desmedt, and Burmester [22] have considered the problem of dependable computation with multiple inputs, that is, they considered the networks of processors where processors may have more than one type of inputs. While there is a polynomial time algorithm for finding vertex disjoint paths in networks of processors with one type of inputs, Wang, Desmedt, and Burmester's work shows that the equivalent problem in computation with multiple inputs is **NP**-hard.

Approximating an **NP**-hard optimization problem within a factor of $1 + \varepsilon$ means to compute solutions whose "cost" is within a multiplicative factor $1+\varepsilon$ of the cost of the optimal solution. Such solution would suffice in practice, if ε were close enough to 0. The question of approximability started receiving attention soon after **NP**-completeness was discovered [14, 20] (see [14] for a discussion). The most successful attempt was due to Papadimitriou and Yannakakis [18], who proved that MAX-3SAT (a problem defined by them) is complete for MAX-SNP (a complexity class defined by them), in other words, any approximability result for MAX-3SAT transfers automatically to a host of other problems. Among other results, they have shown that there is a constant $\varepsilon > 0$ such that it is **NP**-hard to compute a $\frac{k}{1+\varepsilon}$ size independent set of a given graph G, where k is the size of the maximum independent set of G. The results in [18] have been improved by many other authors, especially, after the emergence of the PCP theorem [2, 3], that is, $PCP(\log n, 1) = \mathbf{NP}$ (for a survey, see, e.g., [1, 21]). For example, Arora, Lund, Motwani, Sudan, and Szegedy have shown that it is **NP**-hard to n^{δ}-approximate an independent set for some $\delta > 0$. However, all these results are related to approximating the independent set from "below", that is, to compute an independence set whose size is smaller than the optimal independent set. We will show that it is easy to convert these results to the results of hardness of approximating an independent set from "above" instead of from "below" as done in [1, 21], that is, it is hard to delete some vertices from a given graph such that the resulting graph contains an optimal independent set of the original graph.

2 Optimization and approximation

In this section we present some graph theoretic results which will be used in later sections. First we remind the reader of the graphs defined in the transformation

from 3SAT to Vertex Cover[2] in Garey and Johnson [14, pp. 54–56] and we give such kind of graphs a special name.

Definition 1. *Let n and m be two positive integers. A graph $G(V, E)$ is called an* $n+m$-SAT-graph *if there are $n+m$ subgraphs $L_1(V_{L_1}, E_{L_1})$, ..., $L_n(V_{L_n}, E_{L_n})$, $T_1(V_{T_1}, E_{T_1})$, ..., $T_m(V_{T_m}, E_{T_m})$ of G with the following properties:*

1. $V = (\cup_{i=1}^{n} V_{L_i}) \cup (\cup_{i=1}^{m} V_{T_i})$.
2. *For each $i \leq n$, $|V_{L_i}| = 2$ and E_{L_i} consists of the one edge connecting the two vertices in V_{L_i}.*
3. *For each $i \leq m$, T_i is a triangle, which is isomorphic to the undirected graph $T = (V_T, E_T)$ where $V_T = \{v_1, v_2, v_3\}$ and $E_T = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$.*
4. *There is a function $f : (\cup_{i=1}^{m} V_{T_i}) \to (\cup_{i=1}^{n} V_{L_i})$ such that the edge set of G is $E = (\cup_{i=1}^{n} E_{L_i}) \cup (\cup_{i=1}^{m} E_{T_i}) \cup \{(v, f(v)) : v \in \cup_{i=1}^{m} V_{T_i}\}$.*

The following results are straightforward from the definitions.

Lemma 1. *Given an $n+m$-SAT-graph $G(V, E)$, the following conditions hold.*

1. *The size of an independent set of G is at most $n+m$.*
2. *The size of a vertex cover of G is at least $n+2m$.*

The following result is proved in [14, pp. 54–56].

Lemma 2. *(see [14]) Given a 3SAT formula C with n variables and m clauses, there is an $n+m$-SAT-graph $G(V, E)$ with the following properties:*

1. *C is satisfiable if and only if there is an independent set of size $n+m$ in $G(V, E)$.*
2. *C is satisfiable if and only if there is vertex cover of size $n+2m$ in $G(V, E)$.*

Corollary 1. *It is* **NP***-hard to decide whether there is an independent set of size $n+m$ in an $n+m$-SAT-graph.*

In addition to the problem of deciding whether an $n+m$-SAT-graph has an independent set of size $n+m$, we are also interested in the following approximation problem: for some constant $\varepsilon > 0$ and each $n+m$-SAT-graph G, can we compute in polynomial time an independent set of size $k/(1+\varepsilon)$ in G, where k is the size of the maximum independent set of G? Papadimitriou and Yannakakis [18] (see also, [13, 2]) have proved the following result (note that their original result is for general graphs though their proof is for $n+m$-SAT-graphs).

Definition 2. *For a rational number $\varepsilon > 0$, an algorithm is said to* compute $(1+\varepsilon)$-approximation *to the maximum independent set if given any graph G its output is an independent set of G with size at least $k/(1+\varepsilon)$ where k is the size of the maximum independent set of G.*

[2] A vertex cover of a graph $G(V, E)$ is a subset V' of V such that every edge in E is incident to a vertex in V'.

Theorem 1. *(see [18, 13]) There is a constant $\varepsilon > 0$ such that approximating an independent set of an $n + m$-SAT-graph $G(V, E)$ within a factor $1 + \varepsilon$ is* **NP***-hard.*

Arora, Lund, Motwani, Sudan, and Szegedy [2] have proved the following stronger result.

Theorem 2. *(see [2]) There is a constant $\delta > 0$ such that approximating an independent set of a graph G within a factor n^δ is* **NP***-hard, where n is the number of vertices in G.*

Note that Theorem 2 is only for general graphs. The following variants of Theorems 1 and 2 are useful for our discussions.

Theorem 3. *(see [18, 13]) There is a constant $\varepsilon > 0$ and a polynomial time algorithm to construct for each 3SAT clause C an $n + m$-SAT-graph G with the following properties:*

1. *If C is satisfiable, then G has an independent set of size $n + m$.*
2. *If C is not satisfiable, then $k < \frac{n+m}{1+\varepsilon}$, where k is the size of the maximum independent set in G.*

Proof. It follows from the proof of Theorem 1. □

Theorem 4. *(see [2, 5]) There is a constant $\delta > 0$ and a series of pairs of positive integers $(s_1, c_1), (s_2, c_2), \ldots$ such that $\frac{c_n}{s_n} \geq n^\delta$ for large enough n and from each 3SAT clause C we can construct in a polynomial time a graph G with the following properties:*

1. *If C is satisfiable, then $k \geq c_n$, where k is the size of the maximum independent set in G and n is the number of vertices in G.*
2. *If C is not satisfiable, then $k \leq s_n$, where k is the size of the maximum independent set in G and n is the number of vertices in G.*

Proof. It follows from the proof of Theorem 2. □

Given a graph $G(V, E)$, an edge set $E' \subseteq E$ is said to be *independence eligible* if there is an independent set $V' = \{u :$ there is a $v \in V$ such that the unordered pair $(u, v) \in E'\}$ of size $|E'|$ in G. Note that given an independence eligible edge set E', it is easy to compute an independent set of size $|E'|$ (by a standard algorithm of computing a satisfying assignment of a 2SAT formula).

Theorem 5. *Let ε be the constant in Theorem 3. Then it is* **NP***-hard to compute an edge set E' of a given $n + m$-SAT-graph G with the following properties:*

1. *$|E'| \leq (1 + \varepsilon)k$, where k is the size of a maximum independent set of G.*
2. *E' contains an independence eligible edge set E'' such that $|E''| = k$.*

Proof. It follows from Theorem 3. □

Theorem 6. *There is a constant $\varepsilon > 0$ such that it is* **NP***-hard to compute an edge set $E' \subseteq E$ of a graph $G(V, E)$, with the following properties:*

1. *$|E'| \leq kn^{\varepsilon}$, where k is the size of the maximum independent set of G and $n = |V|$.*
2. *E' contains an independence eligible edge set E'' such that $|E''| \geq \frac{k}{2}$.*

Proof. Let s_n, c_n and δ be the constants in Theorem 4. And let $\varepsilon = \frac{\delta}{2}$. We reduce the **NP**-complete problem 3SAT to the problem of this Theorem. For each 3SAT formula C, construct a graph $G(V, E)$ satisfying the conditions of Theorem 4. Let E' be an edge set satisfying the conditions of the Theorem. Then it suffices to show that if $|E'| \geq \frac{c_n}{2}$ then $k \geq c_n$ (therefore C is satisfiable) else $k \leq s_n$ (therefore C is not satisfiable). If $|E'| \geq \frac{c_n}{2}$ then, by the condition that $|E'| \leq kn^{\varepsilon}$, we have $\frac{c_n}{2} \leq kn^{\varepsilon}$. That is,

$$k \geq \frac{c_n}{2n^{\varepsilon}} > \frac{c_n}{n^{\delta}} \geq s_n.$$

Whence $k \geq c_n$. Otherwise $|E'| < \frac{c_n}{2}$, and

$$\frac{k}{2} \leq |E''| \leq |E'| < \frac{c_n}{2}.$$

That is, $k < c_n$. Whence $k \leq s_n$. □

Corollary 2. *There is a constant $\varepsilon > 0$ such that it is* **NP***-hard to compute a vertex set $V' \subseteq V$ of a graph $G(V, E)$ with the following properties:*

1. *$|V'| \leq kn^{\varepsilon}$, where k is the size of the maximum independent set of G and $n = |V|$.*
2. *V' contains an independent set V'' of $G(V, E)$ such that $|V''| \geq \frac{k}{2}$.*

3 General threats and models for dependable computations

General threats A simple attack to defend against is of a restricted adversary (called *passive adversary*) who is allowed only to monitor communication channels and to jam (denial of service) several processors in the computation system, but is *not* allowed to infiltrate/monitor the internal contents of any processor of the computation system. Of course, a more realistic adversary is the *active adversary* (Byzantine faults) that can monitor all communication between processors and which in addition is also trying to infiltrate the internal contents of several processors.

A passive adversary with the power of jamming up to k processors is called a *k-passive adversary*. An active adversary (Byzantine faults) may mount a more sophisticated attack, where he manages to comprise the security of several internal processors of the system, whereby he is now not only capable of monitoring the external traffic pattern and capable of jamming several processors but is also

capable of examining and modifying every message and data (that is, creating bogus messages and data) which passes through (or stored at) these infiltrated processors. Thus, we define a *k-active adversary*, an adversary that can monitor all the communication lines between processors and also manages to examine and to modify the internal contents of up to k processors of the system. (Similar definitions were considered in the literature, see, for example [7, 8, 12, 19] and references therein).

Achieving processor cooperation in the presence of faults is a major problem in distributed systems, and has been studied extensively (see, e.g, [4, 10, 11, 15]). All these works assume processors with one type of inputs. Recently, Wang, Desmedt, and Burmester [22] have considered the problem of dependable computation with multiple inputs, that is, they considered the networks of processors where processors may have more than one type of inputs. While there is a polynomial time algorithm for finding vertex disjoint paths in networks of processors with one type of inputs, Wang, Desmedt, and Burmester's work shows that the equivalent problem in computation with multiple inputs is **NP**-hard. In this paper, we will consider redundant computation systems with multiple inputs which can be modeled by AND/OR graphs which we now briefly survey.

Definition 3. *(see [22]) An* AND/OR graph $G_{\wedge\vee}(V_\wedge, V_\vee, INPUT, output; E)$ *is a directed graph with a set $V_\wedge$ of* $\wedge$-vertices*, a set $V_\vee$ of* $\vee$-vertices*, a set $INPUT$ of* input vertices*, an* output vertex *$output \in V_\vee$, and a set of* directed edges *E. The vertices without incoming edge are input vertices and the vertex without outgoing edge is the output vertex.*

It should be noted that the above definition of AND/OR graphs is different from the standard definition in artificial intelligence (see, e.g., [17]), in that the directions of the edges are opposite. The reason is that we want to use the AND/OR graphs to model redundant computation systems.

Assume that we use the AND/OR graph to model a fault tolerant computation. So, information (for example, mobile codes) must flow from the input vertices to the output vertex. And a valid computation in an AND/OR graph can be described by a *solution graph* (the exact definition will be given below). However, if insider vertices may be faulty or even malicious, then the output vertex cannot trust that the result is authentic or correct. Firstly we assume that there is only one k-passive adversary at any specific time. The theory of fault tolerant computation (see, Hadzilacos [15]), trivially adapted to the AND/OR graph model, tells us that if there are $k+1$ vertex disjoint paths (solution graphs) of information flow in the AND/OR graph then the vertex *output* will always succeed in getting at least one copy of the results. Secondly we assume that there is one k-active adversary at any specific time. Then the theory of fault tolerant computation (see, e.g., Dolev [10], Dolev et al. [11], and Beimel and Franklin [4]) tells us that if there are $2k+1$ vertex disjoint paths (solution graphs) of information flow in the AND/OR graph then the vertex *output* will always succeed in getting at least $k+1$ identical results computed from the input vertices through vertex disjoint solution graphs, if *output* knows the layout of the graph. This

implies that if *output* knows the layout of the graph then it can use a majority vote to decide whether the result is correct or not. It follows that in order to achieve dependable computation with redundancy, it is necessary to find a set of vertex disjoint solution graphs in a given AND/OR graph.

Definition 4. *(see [22]) Let $G_{\wedge\vee}(V_\wedge, V_\vee, INPUT, output; E)$ be an AND/OR graph. A* solution graph *$P = (V_P, E_P)$ is a minimum subgraph of $G_{\wedge\vee}$ satisfying the following conditions.*

1. *$output \in V_P$.*
2. *For each $\wedge$-vertex $v \in V_P$, all incoming edges of v in E belong to E_P.*
3. *For each $\vee$-vertex $v \in V_P$, there is exactly one incoming edge of v in E_P.*
4. *There is a sequence of vertices $v_1, \ldots, v_n \in V_P$ such that $v_1 \in INPUT, v_n = output$, and $(v_i \rightarrow v_{i+1}) \in E_P$ for each $i < n$.*

Moreover, two solution graphs P_1 and P_2 are vertex disjoint *if $(V_{P_1} \cap V_{P_2}) \subseteq (INPUT \cup \{output\})$. An AND/OR graph is called* k-connected *if the following conditions are satisfied.*

1. *There are k vertex disjoint solution graphs in $G_{\wedge\vee}$.*
2. *There do not exist $k+1$ vertex disjiont solution graphs in $G_{\wedge\vee}$.*

In order for an adversary to attack the computation system, s/he does not need to find all vertex disjoint solution graphs in an AND/OR graph. For a passive adversary, s/he can choose to jam one vertex on each solution graph to corrupt the system. An active adversary needs to find one half of the vertices of a *vertex separator* (defined in the following).

Definition 5. *Let $G_{\wedge\vee}$ be a k-connected AND/OR graph, and $P = \{P_1, \ldots, P_k\}$ be a maximum set of vertex disjoint solution graphs in $G_{\wedge\vee}$. A set $S = \{v_1, \ldots, v_k\}$ of vertices in $G_{\wedge\vee}$ is called a* vertex separator *of P if for each solution graph $P_i \in P$ $(i = 1, \ldots, k)$, $v_i \in V_{P_i}$.*

Remark: The problem of finding a vertex separator in an AND/OR graph is **NP**-hard which will be proved in Section 5.

The question we are addressing in this paper is how to design AND/OR graphs with less vertex disjoint solution graphs to achieve dependable computation against more powerful passive or active adversaries.

4 Dependable computation with trap-doors

In this section, we show how to design dependable computation systems with trap-doors such that the following condition is satisfied:

- The computation system modeled by a k-connected AND/OR graph is robust against a k'-active adversary (therefore robust against a k'-passive adversary also) where $k' \leq ck$ and $c > 1$ is any given constant.

The idea is to use the fact that it is **NP**-hard to approximate a vertex separator of an AND/OR graph from "above" (see Section 2 for details about approximating an **NP**-hard optimization problem from "above"). It follows that if one designs the AND/OR graph in such a way that the trusted participants can easily find vertex disjoint solution graphs in it (using some trap-doors), and the input vertices always initiate a computation through all solution graphs in the maximum set of vertex disjoint solution graphs, then dependable computation is possible. The benefit from using trap-doors in a computation system with multiple inputs is obvious. If we do not use trap-doors then, by extending the conventional fault tolerant computation theory (see, e.g., [4, 7, 10, 11, 15]), a k-connected AND/OR graph is only robust against k'-passive adversaries and only robust against k''-active adversaries respectively, when $k' < k$ and $k'' < \frac{k}{2}$. Since if the adversary has the power to jam k vertices in the AND/OR graph and s/he can find a vertex separator of size k, then s/he can jam all of the vertices in the vertex separator and corrupt the system. Indeed, if the adversary has the power to examine and modify messages and data in $\lfloor \frac{k}{2} \rfloor + 1$ processors, then the adversary may let the $\lfloor \frac{k}{2} \rfloor + 1$ faulty processors create and send faulty messages to the output processor claiming that they come from some bogus solution graphs. This will convince the output vertex to accept the bogus message since the majority messages are faulty. However, if we use trap-doors in the design of AND/OR graphs, then with high probability, a k-connected AND/OR graph is robust against k'-active adversaries (therefore against k'-passive adversaries) where $k' \leq ck$ and $c > 1$ is any given constant. The reason is that even though the adversary has the power to jam or control $k' > k$ vertices in the AND/OR graph, he does not know which vertices to corrupt, that is, the corrupted vertices (in his control) will appear on at least half of the k vertex disjoint solution graphs.

So one of the main problems is to design AND/OR graphs in which it is hard on the average case to approximate at least one half of a vertex separator from "above". In Section 5, we will outline an approach to generate such kind of AND/OR graphs. In the remaining part of this section we will demonstrate how to use these AND/OR graphs to achieve dependability.

Protocol I

1. Alice generates a k-connected AND/OR graph $G_{\wedge\vee}$ such that the graph $G_{\wedge\vee}$ can implement the desired computation and such that finding a ck size set of vertices which contains at least one half of the elements of a vertex separator is hard, where $c > 1$ is any given constant. (The details will be presented in Section 5).
2. Using a secure channel, Alice sends the input vertices the method of initiating a computation and sends the output vertex a maximum set of vertex disjoint solution graphs in $G_{\wedge\vee}$.
3. In order to carry out one computation, Alice initiates the computation through all solution graphs in the maximum set of vertex disjoint solution graphs.

4. When the output vertex gets all possible outputs, he compares the results from the k vertex disjoint solution graphs (note that the output vertex knows the maximum set of vertex disjoint solution graphs) and chooses the authentic result using a majority vote.

Note that our above protocol is not secure against a dynamic adversary who after observing one computation will change the vertices he controls. Indeed, it is an interesting open problem to design protocols which are secure against dynamic adversaries.

Now assume that Mallory is a k'-active adversary (or a k'-passive adversary) where $k' \leq ck$ for the constant $c > 1$, and $P = \{P_1, \ldots, P_k\}$ is a maximum set of vertex disjoint solution graphs in the AND/OR graph used in **Protocol I**. Since Mallory does not know how to find a k' size set of vertices which contains at least one half of the elements of a vertex separator for P (finding such a set is very hard), she does not know which vertices to corrupt so that she can generate at least $\lfloor \frac{k}{2} \rfloor + 1$ bogus messages to convince the output vertex to accept (or so that all these k solution graphs will be jammed), even though she has the power to corrupt $k' = ck$ vertices. It follows that the system is robust against a k'-active adversary (therefore robust against a k'-passive adversary also) where $k' \leq ck$.

5 AND/OR graphs with trap-doors

In this section, we outline an approach for constructing AND/OR graphs with trap-doors. We first show that it is **NP**-hard to approximate at least half of the elements of a vertex separator of an AND/OR graph from "above".

Theorem 7. *Given an AND/OR graph $G_{\wedge\vee}(V_\wedge, V_\vee, INPUT, output; E)$, it is* **NP***-hard to compute a vertex set $S' \subseteq (V_\wedge \cup V_\vee)$ with the following properties:*

1. *If $G_{\wedge\vee}$ is k-connected then $|S'| \leq ck$.*
2. *For some vertex separator S of $G_{\wedge\vee}$, $|S \cap S'| \geq \frac{k}{2}$.*

Proof. We reduce the problem of Theorem 6 to the problem of this Theorem.

For a given graph $G'(V', E')$, we construct an AND/OR graph $G''_{\wedge\vee}(V''_\wedge, V''_\vee, INPUT'', output''; E'')$ as follows. Assume that $V' = \{v_1, \ldots, v_n\}$. Let $INPUT'' = \{I_i, I_{i,j} : i, j = 1, \ldots n\}$, $V''_\vee = \{output\}$, $V''_\wedge = \{u_{i,j} : i, j = 1, \ldots n\} \cup \{u_i : i = 1, \ldots, n\}$, and E'' be the set of the following edges.

1. For each $i = 1, \ldots, n$, there is an edge $I_i \rightarrow u_i$.
2. For each pair $i, j = 1, \ldots, n$, there is an edge $I_{i,j} \rightarrow u_{i,j}$.
3. For each pair $i, j = 1, \ldots, n$, such that $(v_i, v_j) \in E'$, there are four edges $u_{i,j} \rightarrow u_i$, $u_{i,j} \rightarrow u_j$, $u_{j,i} \rightarrow u_i$, and $u_{j,i} \rightarrow u_j$.
4. For each i, there is an edge $u_i \rightarrow output''$.

It is clear that two solution graphs P_1 and P_2 in $G''_{\wedge\vee}$ which go through u_i and u_j respectively are vertex disjoint if and only if there is no edge (v_i, v_j) in E'. Hence there is a size k independent set in G' if and only if there are k vertex

disjoint solution graphs in $G''_{\wedge\vee}$. And from k vertex disjoint solution graphs in $G''_{\wedge\vee}$ one can compute in linear time a size k independent set in G'. Whence it is sufficient to show that from each vertex set S' satisfying the conditions of the Theorem, one can compute in polynomial time an edge set $E_{S'} \subseteq E'$ with the following properties:

1. If $G_{\wedge\vee}$ is k-connected (that is, if the optimal independent set in G' has size k) then $|E_{S'}| \leq ck$.
2. $E_{S'}$ contains an independence eligible edge set of size at least $\frac{k}{2}$.

The following algorithm will output an edge set $E_{S'}$ with the above properties. In the following S' is the vertex set satisfying the conditions of the Theorem.

- Let $E_{S'} = \emptyset$. For $i = 1, \ldots, ck$, we distinguish the following two cases:
  1. $s_i = u_j$ for some $j \leq n$. Let $E_{S'} = E_{S'} \cup \{(v_j, v'_j)\}$ where v'_j is any vertex in G' which is incident to v_j.
  2. $s_i = u_{j_1,j_2}$ for some $j_1, j_2 \leq n$. Let $E_{S'} = E_{S'} \cup \{(v_{j_1}, v_{j_2})\}$ if $(v_{j_1}, v_{j_2}) \in E'$ and $E_{S'} = E_{S'}$ otherwise.

By the property of S', it is clear that $E_{S'}$ has the required properties.

By Theorem 6, we have completed the proof of the Theorem. □

In the remaining part of this section, we outline how to construct AND/OR graphs with trap-doors.

Construction First generate a graph $G'(V', E')$ and a number k which satisfy the conditions of Theorem 3 (or Theorem 4). Secondly use the method in the proof of Theorem 7 to generate an AND/OR graph $G''_{\wedge\vee}$ with the property that it is hard to approximate at least half of the elements of a vertex separator of $G''_{\wedge\vee}$ from "above". The AND/OR graph $G_{\wedge\vee}$ is obtained by replacing all vertices $u_{i,j}$ of $G''_{\wedge\vee}$ with the AND/OR graph $G^1_{\wedge\vee}$, where $G^1_{\wedge\vee}$ is the AND/OR graph which can implement the desired computation. As a summary, the construction proceeds as follows.

$$\text{graph } G' \rightarrow \text{AND/OR graph } G''_{\wedge\vee} \overset{G^1_{\wedge\vee}}{\rightarrow} \text{AND/OR graph } G_{\wedge\vee}$$

6 Towards practical solutions

In the previous section, we considered the problem of designing AND/OR graphs with trap-doors. Specifically, we constructed AND/OR graphs which is robust against ck-active adversaries (therefore robust against ck-passive adversaries also). However, these constructions are inefficient and are only of theoretical interests. One of the most interesting open questions is how to efficiently generate hard instances of AND/OR graphs, especially, for arbitrary number k. If we do not require that c be an arbitrary given constant, then Theorem 5 can be used to construct AND/OR graphs which are more "efficient" (though still have

enormous complexity) than the AND/OR graphs constructed in the previous section and which are robust against $(1+\varepsilon)k$-passive adversaries where $\varepsilon < 1$ is a small positive rational number. However, in order to construct AND/OR graphs which are robust against ck-active adversaries for $c > \frac{1}{2}$, we have to use Theorem 6 in our construction. And the size of the graph G in Theorem 6 will be impractical if we want to make the security of the system to be at least as hard as an exhaustive search of a 1024-bit space.

We should also note that, in order to construct the AND/OR graphs in the previous section, we need to construct standard graphs which satisfy the conditions of Theorem 3 (or Theorem 4). That is, we need an algorithm to build graphs whose independent sets are hard to approximate in the average case (note that Theorem 7 only guarantees the worst-case hardness instead of average-case hardness). Whence it is interesting (and open) to prove some average-case hardness results for the corresponding problems.

In the following, we consider the problem of constructing practical average-case hard AND/OR graphs which are robust against $k+c$-passive adversaries, where c is some given constant. Our following construction is based on the hardness of factoring a large integer and we will not use the approximation hardness results.

Construction Let N be a large number which is a product of two primes p and q. We will construct an AND/OR graph $G_{\wedge\vee}$ with the following property: given the number N and a vertex separator for $G_{\wedge\vee}$, one can compute efficiently the two factors p and q. Let $x_1, \ldots, x_t$ and $y_1, \ldots, y_t$ be variables which take values 0 and 1, where $t = \lfloor \log N \rfloor$. And let $(x_t \ldots x_1)_2$ and $(y_t \ldots y_1)_2$ to denote the binary representations of $\sum x_i 2^{i-1}$ and $\sum y_i 2^{i-1}$ respectively. Then use the relation

$$(x_t \ldots x_1)_2 \times (y_t \ldots y_1)_2 = N \tag{1}$$

to construct a 3SAT formula C with the following properties:

1. C has at most $O(t^2)$ clauses.
2. C is satisfiable and, from a satisfying assignment of C, one can compute in linear time a assignment of $x_1, \ldots, x_t, y_1, \ldots, y_t$ such that the equation (1) is satisfied. That is, from a satisfying assignment of C, one can factor N easily.

Now use Lemma 2 to construct an $n+m$-SAT-graph $G'(V', E')$ and a number $k = O(t^2)$ with the property that: from a size k independent set of G' one can compute in linear time a satisfying assignment of C. Lastly, use the method in the proof of Theorem 7 to generate an AND/OR graph $G_{\wedge\vee}$ with the property that, from a vertex separator of $G_{\wedge\vee}$, one can compute in linear time a size k independent set of G' (note that, instead of approximating a vertex separator, here we need to know a whole set of vertex separator). As in the proof of Theorem 7, from a vertex separator of $G_{\wedge\vee}$ one can easily compute a size k independence eligible edge set of G', from which one can compute in linear time a size k independent set of G' (using the method of computing a satisfying assignment of a 2SAT formula).

It is straightforward to see that the above constructed AND/OR graph $G_{\wedge\vee}$ is robust against $k + c$-passive adversaries if factoring N is hard, where c is any given constant.

References

1. S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems.* PhD Thesis, CS Division, UC Berkeley, August, 1994.
2. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In: *Proceedings of 33rd IEEE Symposium on Foundations of Computer Science*, pp. 13–22, 1992.
3. S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of **NP**. In: *Proceedings of 33rd IEEE Symposium on Foundations of Computer Science*, pp. 2–13, 1992.
4. A. Beimel and M. Franklin. Reliable communication over partially authenticated networks. In. *Proceedings of the WDAG '97, Lecture Notes in Computer Science 1320*, pp. 245–259, Springer Verlag, 1997.
5. M. Bellare. Proof checking and approximation: towards tight results. In: *Complexity Theory Column 12, SIGACT News.* **27**(1), March 1996.
6. R. Boppana and M. Halldorsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, **32**(2), pp. 180–196, 1992.
7. M. Burmester, Y. Desmedt, and G. Kabatianski. Trust and security: a new look at the Byzantine general problems. In: R. N. Wright and P. G. Neumann, Eds, *Network Threats, DIMACS, Series in Discrete Mathematics and Theoretical Computer Science,* AMS, Vol. 38, 1997.
8. R. Canetti, E. Kushilevitz, R. Ostrovsky, and A. Rosen. Randomness vs. fault-tolerance. In: *Proceedings of the PODC '97*, pp. 35–44, Springer Verlag, 1997.
9. D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communication of the ACM*, Vol. 24, pp. 84–88, 1981.
10. D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, **3**, pp. 14–30, 1982.
11. D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. of the ACM*, **40**(1), pp. 17–47, 1993.
12. S. Dolev and R. Ostrovsky. Efficient anonymous multicast and reception. In: Advances in Cryptology, Crypto'97, pp. 395–409, Springer Verlag, 1997.
13. U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost **NP**-complete. In: *Proceedings of 32nd IEEE Symposium on Foundations of Computer Science*, pp. 2–11, 1991.
14. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of* **NP***-Completeness.* W. H. Freeman and Company, San Francisco, 1979.
15. V. Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations.* PhD thesis, Harvard University, Cambridge, MA, 1984.
16. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *J. of the ACM*, **32**(2), pp. 374–382, 1982.
17. N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980.
18. C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, **43**, 425–440, 1991.
19. C. Rackoff and S. Simon. Cryptographic defense against traffic analysis. In: *Proceedings of the STOC 93*, pp. 672–681.

20. S. Sahni and T. Gonzalez. **P**-complete approximation problems. *J. of the ACM*, **23**, pp. 555–565, 1976.
21. M. Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems.* PhD. thesis, U. C. Berkeley, 1992.
22. Y. Wang, Y. Desmedt, and M. Burmester. Hardness of dependable computation with multiple inputs. *Submitted.*

Complexity of Sequential Pattern Matching Algorithms*

Mireille Régnier[1] and Wojciech Szpankowski[2]

[1] INRIA, Rocquencourt, 78153 Le Chesnay Cedex, FRANCE
[2] Dept. Computer Science, Purdue University, W. Lafayette, IN 47907, USA

Abstract. We formally define a class of sequential pattern matching algorithms that includes all variations of Morris-Pratt algorithm. For the last twenty years it was known that the complexity of such algorithms is bounded by a linear function of the text length. Recently, substantial progress has been made in identifying lower bounds. We now prove there exists asymptotically a *linearity constant* for the worst and the average cases. We use *Subadditive Ergodic Theorem* and prove an almost sure convergence. Our results hold for any given pattern and text and for stationary ergodic pattern and text. In the course of the proof, we establish some structural property, namely, the existence of "unavoidable positions" where the algorithm must stop to compare. This property seems to be uniquely reserved for Morris-Pratt type algorithms (e.g., Boyer and Moore algorithm does not possess this property).

1 Introduction

The complexity of string searching algorithms has been discussed in various papers (cf. [1, 6, 7, 8, 9, 12, 18]). It is well known that most pattern matching algorithms perform linearly in the worst case as well as "on average". Several attempts have been made to provide tight bounds on the so-called "linearity constant". Nevertheless, the existence of such a constant has never been proved. The only exception known to us is the average case of Morris-Pratt-like algorithms [18] (cf. [17]) for the symmetric Bernoulli model (independent generation of symbols with each symbol occurring with the same probability) where the constant was also explicitly computed.

In this paper we investigate a fairly general class of algorithms, called *sequential algorithms*, for which the existence of the linearity constant (in an asymptotic sense) is proved for the worst and the average case. Sequential algorithms include the naive one and several variants of Morris-Pratt algorithm [16]. These algorithms never go backward, work by comparisons, and are easy to implement. They perform better than Boyer-Moore like algorithms in numerous cases, e.g., for binary alphabet [2], when character distributions are strongly biased, and when the pattern and text distributions are correlated. Thus, even from a practical point of view these algorithms are worth studying.

* The project was supported by NATO Collaborative Grant CRG.950060, the ESPRIT III Program No. 7141 ALCOM II, and NSF Grants NCR-9415491, NCR-9804760.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 187-199, 1998.

In this paper we analyze sequential algorithms under a general probabilistic model that only assumes stationarity and ergodicity of the text and pattern sequences. We show that asymptotic complexity grows linearly with the text length for all but finitely many strings (i.e., in almost sure sense). The proof relies on the *Subadditive Ergodic Theorem* [11].

The literature on worst case as well average case on Knuth-Morris-Pratt type algorithms is rather scanty. For almost twenty years the upper bound was known [16], and no progress has been reported on a lower bound or a tight bound. This was partially rectified by Colussi *et al.* [8] and Cole *et al.* [7] who established several lower bounds for the so called "on-line" sequential algorithms. However, the existence of the linearity constant was not established yet, at least for the "average complexity" under general probabilistic model assumed in this paper. In the course of proving our main result, we construct the so called *unavoidable positions* where the algorithm must stop to compare. The existence of these positions is crucial for establishing the subadditivity of the complexity function for the Morris-Pratt type algorithms, and hence their linearity. This property seems to be restricted to Morris-Pratt type algorithms (e.g., the Boyer-Moore algorithm does not possess any unavoidable position).

The paper is organized as follows. In the next section we present a general definition of sequential algorithms, and formulate our main results. Section 3 contains all proofs. In concluding remarks we apply Azuma's inequality to show that the complexity is well concentrated around its most likely value (even if the value of the linearity constant is still unknown).

2 Sequential Algorithms

In this section, we first present a general definition of sequential algorithms (i.e., algorithms that work like Morris-Pratt). Then, we formulate our main results and discuss some consequences.

2.1 Basic Definitions

Throughout we write $\mathbf{p}$ and $\mathbf{t}$ for the pattern and the text which are of lengths m and n, respectively. The ith character of the pattern $\mathbf{p}$ (text $\mathbf{t}$) is denoted as $\mathbf{p}[i]$ ($\mathbf{t}[i]$), and by $\mathbf{t}_i^j$ we define the substring of $\mathbf{t}$ starting at position i and ending at position j, that is $\mathbf{t}_i^j = \mathbf{t}[i]\mathbf{t}[i+1]\cdots\mathbf{t}[j]$. We also assume that for a given pattern $\mathbf{p}$ its length m does not vary with the text length n.

Our prime goal is to investigate complexity of string matching algorithms that work by comparisons (i.e., the so called comparison model).

Definition 1. (i) For any string matching algorithm that runs on a given text $\mathbf{t}$ and a given pattern $\mathbf{p}$, let $M(l,k) = 1$ if the lth symbol $\mathbf{t}[l]$ of the text is *compared* by the algorithm to the kth symbol $\mathbf{p}[k]$ of the pattern; and $M(l,k) = 0$ otherwise. We assume in the following that this comparison is performed at most once.

(ii) For a given pattern matching algorithm partial complexity function $c_{r,n}$ is defined as

$$c_{r,s}(\mathbf{t},\mathbf{p}) = \sum_{l\in[r,s],k\in[1,m]} M[l,k] \tag{1}$$

where $1 \le r < s \le n$. For $r = 1$ and $s = n$ the function $c_{1,n} := c_n$ is simply called the **complexity** of the algorithm. If either the pattern or the text is a realization of a random sequence, then we denote the complexity by a capital letter, that is, we write C_n instead of c_n.

Our goal is to find an asymptotic expression for c_n and C_n for large n under deterministic and stochastic assumptions regarding the strings $\mathbf{p}$ and $\mathbf{t}$. (For simplicity of notation we often write c_n instead of $c_n(\mathbf{t},\mathbf{p})$.) We need some further definitions that will lead to a formal description of sequential algorithms.

We start with a definition of an *alignment position*.

Definition 2. Given a string searching algorithm, a text $\mathbf{t}$ and a pattern $\mathbf{p}$, a position AP in the text $\mathbf{t}$ satisfying for some k $(1 \le k \le m)$

$$M[AP + (k-1), k] = 1$$

is said to be an *alignment position*.

Intuitively, at some step of the algorithm, an alignment of pattern $\mathbf{p}$ at position AP is considered, and a comparison is made with character $\mathbf{p}[k]$ of the pattern.

Finally, we are ready to define *sequential algorithms*. Sequentiality refers to a special structure of a sequence of positions that pattern and text visit during a string matching algorithm. Throughout, we shall denote these sequences as (l_i, k_i) where l_i refers to a position visited during the ith comparison by the text while k_i refers to a position of the pattern when the pattern is aligned at position $l_i - k_i + 1$.

Definition 3. A string searching algorithm is said to be:

(i) **semi-sequential** if the text is scanned from left to right;

(ii) **strongly semi-sequential** if the order of text-pattern comparisons actually performed by the algorithm defines a non-decreasing sequence of text positions (l_i) and if the sequence of alignment positions is non-decreasing.

(iii) **sequential** (respectively **strongly sequential**) if they satisfy (i) (respectively (ii)) and if, additionally, for any $k > 1$

$$M[l,k] = 1 \quad \Rightarrow \quad \mathbf{t}_{l-(k-1)}^{l-1} = \mathbf{p}_1^{k-1} \tag{2}$$

In passing, we point out that condition (i) means that the text is read from left to right. Note that our assumptions on non-decreasing text positions in (ii)

implies (i). Furthermore, non-decreasing alignment positions implies that all occurrences of the pattern before this alignment position were detected before this choice. Nevertheless, these constraints on the sequence of text-pattern comparisons (l_i, k_i) are not enough to prevent the algorithm to "fool around", and to guarantee a general tight bound on the complexity. Although (2) is not a logical consequence of semi-sequentiality, it represents a natural way of using the available information for semi-sequential algorithms. In that case, subpattern $\mathbf{t}_{l-(k-1)}^{l-1}$ is known when $\mathbf{t}[l]$ is read. There is no need to compare $\mathbf{p}[k]$ with $\mathbf{t}[l]$ if $\mathbf{t}_{l-(k-1)}^{l-1}$ is not a prefix of $\mathbf{p}$ of size $k-1$, i.e if $AP = l-(k-1)$ has already been disregarded.

We now illustrate our definition on several examples.

Example 1: *Naive or brute force algorithm*

The simplest string searching algorithm is the naive one. All text positions are alignment positions. For a given one, say AP, text is scanned until the pattern is found or a mismatch occurs. Then, $AP+1$ is chosen as the next alignment position and the process is repeated.

This algorithm is sequential (hence semi-sequential) but not strongly sequential. Condition in (ii) is violated after any mismatch on an alignment position l with parameter $k \geq 3$, as comparison $(l+1, 1)$ occurs after $(l+1, 2)$ and $(l+2, 3)$.

Example 2: *Morris-Pratt-like algorithms* [16, 19].

It was already noted in [16] that after a mismatch occurs when comparing $\mathbf{t}[l]$ with $\mathbf{p}[k]$, some alignment positions in $[l+1, \ldots, l+k-1]$ can be disregarded without further text-pattern comparisons. Namely, the ones that satisfy $\mathbf{t}_{l+i}^{l+k-1} \neq \mathbf{p}_1^{k-i}$. Or, equivalently, $\mathbf{p}_{1+i}^{k} \neq \mathbf{p}_1^{k-i}$, and the set of such i can be known by a preprocessing of $\mathbf{p}$. Other i define the "surviving candidates", and chosing the next alignment position among the surviving candidates is enough to *ensure* that condition (ii) in Definition 3 holds. Different choices lead to different variants of the classic Morris-Pratt algorithm [16]. They differ by the use of the information obtained from the mismatching position. We formally define three main variants, and provide an example. One defines a shift function S to be used after any mismatch as:

Morris-Pratt variant:

$$S = \min\{k-1;\ \min\{s > 0 :\ \mathbf{p}_{1+s}^{k-1} = \mathbf{p}_1^{k-1-s}\}\}\ ;$$

Knuth-Morris-Pratt variant:

$$S = \min\{k;\ \min\{s :\ \mathbf{p}_{1+s}^{k-1} = \mathbf{p}_1^{k-1-s} \textbf{ and } \mathbf{p}_k^{k} \neq \mathbf{p}_{k-s}^{k-s}\}\}\ ;$$

Simon variant:

$$K = \max\{k : \ M(l,k) = 1\} \ ;$$
$$B = \{s : \ p_{1+s}^{K-1} = p_1^{K-1-s} \textbf{ and } 0 \leq s \leq K-k\} \ ;$$
$$S = \min\{d > 0 : \mathbf{p}_{1+d}^{k-1} = \mathbf{p}_1^{k-1-d} \textbf{ and } (\mathbf{p}_{k-d}^{k-d} \neq p_{K-s}^{K-s}, s \in B)\}$$

Example 3: *Illustration to Definition 3.*

Let $\mathbf{p} = abacabacabab$ and $\mathbf{t} = abacabacabaaa$. The first mismatch occurs for $M(12,12)$. The comparisons performed from that point are:

- **Morris-Pratt variant:**

  $$(12,12); (12,8); (12,4); (12,2); (12,1); (13,2); (13,1) \ ,$$

  where the text character is compared with pattern characters (b,c,c,b,a,b,a) with the alignment positions $(1,5,9,11,12,12,13)$.

- **Knuth-Morris-Pratt variant:**

  $$(12,12); (12,8); (12,2); (12,1); (13,2); (13,1) \ ,$$

  where the text character is compared with pattern characters (b,c,b,a,b,a) with the alignment positions $(1,5,11,12,12,13)$.

- **Simon variant:**

  $$(12,12); (12,8); (12,1); (13,2); (13,1) \ ,$$

  where the text character is compared in turn with pattern characters (b,c,a,b,a) with the alignment positions $(1,5,12,12,13)$.

Some observations are in sequel: Morris-Pratt variant considers one alignment position at a time, while the optimal sequential algorithm, that of Simon, considers several alignment positions at the same time, and may disregard several of them simultaneously (e.g., in Example 3 positions 1 and 9 at the first step and 5 and 11 at the second step). It is interesting to observe that the subset $\{1,5,12\}$ of alignments positions appears in all variants. We will see that they share a common property of "unavoidability" explored below.

Our definition of semi-sequentiality is very close to the definition of sequentiality given in [13]. We do not use the "on-line" concept of [6]. The on-line algorithms are very close to our strongly sequential ones. Also, while condition (2) is a natural optimization for semi-sequential algorithms, it seems not to be true for other efficient algorithms discussed in [8].

Finally, in the course of proving our main result we discover an interesting structural property of sequential algorithms that we already observed in Ex. 3. Namely, when the algorithm is run on a substring of the text, say $\mathbf{t}_r^n$, then there are some positions $i \geq r$ that are *unavoidable* alignment positions, that is, the algorithm *must* align at this positions at some step (e.g., see positions $\{1,5,12\}$ in Ex. 3). More formally:

Definition 4. For a given pattern $\mathbf{p}$, a position i in the text $\mathbf{t}_1^n$ is an **unavoidable alignment position** for an algorithm if for any r, l such that $r \leq i$ and $l \geq i + m$, the position i is an alignment position when the algorithm is run on $\mathbf{t}_r^l$.

Having in mind the above definitions we can describe our last class of sequential algorithms (containing all variants of KMP-like algorithms) for which we formulate our main results.

Definition 5. An algorithm is said to be ℓ-convergent if, for any text $\mathbf{t}$ and pattern $\mathbf{p}$, there exists an increasing sequence $\{U_i\}_{i=1}^n$ of unavoidable alignment positions satisfying $U_{i+1} - U_i \leq \ell$ where $U_0 = 0$ and $n - \max_i U_i \leq \ell$.

In passing we note that the naive pattern matching algorithm (cf. Ex. 1) is 1-convergent. We prove below that all strongly sequential algorithms (i.e., all Morris-Pratt-like algorithms) are m-convergent which will further imply several interesting and useful properties of these algorithms (e.g., linear complexity).

2.2 Main Results

In this section we formulate our main results. Before, however, we must describe modeling assumptions concerning the strings. We adopt one of the following assumptions:

(A) Worst-Case (Deterministic) Model

Both strings $\mathbf{p}$ and $\mathbf{t}$ are non random (deterministic) and $\mathbf{p}$ is given.

(B) Semi-Random model

The text string $\mathbf{t}$ is a realization of a stationary and ergodic sequence while the pattern string $\mathbf{p}$ is given.

(C) Stationary Model

Strings $\mathbf{t}$ and $\mathbf{p}$ are realizations of a *stationary* and *ergodic* sequence (cf. [3]). (Roughly speaking, a sequence, say $\mathbf{t}_1^n$, is stationary if the probability distribution is the same for all substrings of equal sizes, say $\mathbf{t}_i^{i+k}$ and $\mathbf{t}_j^{j+k}$ for $1 \leq i < j \leq n$.)

Formulation of our results depends on the model we work with. In the deterministic model we interpret the complexity $c_n(\mathbf{t}, \mathbf{p})$ as the worst case complexity (i.e., we maximize the complexity over all texts). Under assumption (B) we consider *almost sure* (a.s.) convergence of C_n. More formally, we write $C_n/a_n \to \alpha$ (a.s.) where a_n is a deterministic sequence and α is a constant if $\lim_{n\to\infty} \Pr\{\sup_{k\geq n} |C_k/a_k - \alpha| > \varepsilon\} = 0$ for any $\varepsilon > 0$ (cf. [3]). Finally, in the stationary model (C) we use standard average case complexity, that is, EC_n.

Now we are ready to formulate our main results.

Theorem 6. *Consider an $\ell \leq m$ convergent sequential string matching algorithm. Let $\mathbf{p}$ be a given pattern of length m.*

(i) *Under assumption (A) the following holds*

$$\lim_{n\to\infty} \frac{\max_{\mathbf{t}} c_n(\mathbf{t},\mathbf{p})}{n} = \alpha_1(\mathbf{p}) \tag{3}$$

where $\alpha_1(\mathbf{p}) \geq 1$ is a constant.

(ii) *Under assumption (B) one finds*

$$\frac{C_n(\mathbf{p})}{n} \to \alpha_2(\mathbf{p}) \quad \text{a.s.} \tag{4}$$

where $\alpha_2(\mathbf{p}) \geq 1$ is a constant. If $E_{\mathbf{t}}$ denotes the the average cost over all text strings, the following also holds:

$$\lim_{n\to\infty} \frac{E_{\mathbf{t}} C_n(\mathbf{p})}{n} = \alpha_2(\mathbf{p}) \tag{5}$$

Theorem 7. *Consider an ℓ-convergent sequential string matching algorithm. Under assumption (C) we have*

$$\lim_{n\to\infty} \frac{E_{\mathbf{t},\mathbf{p}} C_n}{n} = \alpha_3 \tag{6}$$

provided $m = o(\sqrt{n})$, where $\alpha_3 \geq 1$ is a constant and $E_{\mathbf{t},\mathbf{p}}$ denotes the average over all text strings of size n and patterns of size m.

Finally, with respect to our main class of algorithms, namely, Morris-Pratt like (i.e., sequential) we shall prove in the next section the following results concerning the existence of unavoidable positions.

Theorem 8. *Given a pattern $\mathbf{p}$ and a text $\mathbf{t}$, all strongly sequential algorithms have the same set of unavoidable alignment positions $U = \bigcup_{l=1}^{n} \{U_l\}$, where*

$$U_l = \min\{\min_{1\leq k\leq l} \{\mathbf{t}_k^l \preceq \mathbf{p}\},\ l+1\} \tag{7}$$

and $\mathbf{t}_k^l \preceq \mathbf{p}$ means that the substring $\mathbf{t}_k^l$ is a prefix of the pattern $\mathbf{p}$.

Theorem 9. *Strongly sequential algorithms (e.g., Morris-Pratt like algorithms) are m-convergent and (3)-(6) hold.*

In summary, the above says that there exists a constant α such that $c_n = \alpha n + o(n)$ and/or $EC_n = \alpha n + o(n)$. All previous results have been able only to show that $c_n = \Theta(n)$ but they did not excluded some bounded fluctuation of the coefficient at n. We should point out that in the analysis of algorithms on words such a fluctuation can occur in some problems involving suffix trees (cf. [4, 14, 20]). But, in this paper we prove that such a fluctuation cannot take place for the complexity function of the strongly sequential pattern matching algorithms. For example, in the worst case we prove here that for any given pattern $\mathbf{p}$, any $\epsilon > 0$ and any $n \geq n_\epsilon$, one can find a text $\mathbf{t}_1^n$ such that $|\frac{c_{1,n}}{n} - \alpha_1(p)| \leq \epsilon$.

3 Analysis

In this section we prove Theorems 6–9. The idea of the proof is quite simple. We shall show that a function of the complexity (i.e., $c'_n = c_n + f(m)$ where $f(m)$ is a function of the length m of the pattern $\mathbf{p}$) is subadditive. In the "average case analysis" we indicate that under assumption (C) the average complexity C_n is a stationary and ergodic sequence. Then, direct application of an extension of Kingman's *Subadditive Ergodic Theorem* due to Derriennic [10] will do the job of proving our results. In passing, we point out that the most challenging is establishing the subadditivity property to which most of this section is devoted.

For the reader's convenience we start this section with a brief review of the subadditive ergodic theorem (cf. [11, 15].

Theorem 10. (Subadditive Sequence). *(i) Let for a (deterministic) nonnegative sequence* $\{x_n\}_{n=0}^{\infty}$ *the following property, called subadditivity, holds*

$$x_{m+n} \le x_n + x_m \ . \tag{8}$$

Then

$$\lim_{n\to\infty} \frac{x_n}{n} = \inf_{m\ge 1} \frac{x_m}{m} = \alpha \tag{9}$$

for some constant α.

(ii) (Subadditive Ergodic Theorem [15]). *Let* $X_{m,n}$ $(m < n)$ *be a sequence of nonnegative random variables satisfying the following three properties*

(a) $X_{0,n} \le X_{0,m} + X_{m,n}$ *(subadditivity);*

(b) $X_{m,n}$ *is stationary (i.e., the joint distributions of* $X_{m,n}$ *are the same as* $X_{m+1,n+1}$*) and ergodic (cf. [3]);*

(c) $EX_{0,1} < \infty$.

Then,

$$\lim_{n\to\infty} \frac{EX_{0,n}}{n} = \gamma \qquad \text{and} \qquad \lim_{n\to\infty} \frac{X_{0,n}}{n} = \gamma \quad \text{(a.s.)} \tag{10}$$

for some constant γ.

(iii) (Almost Subadditive Ergodic Theorem [10]). *If the subadditivity inequality is replaced by*

$$X_{0,n} \le X_{0,m} + X_{m,n} + A_n \tag{11}$$

such that $\lim_{n\to\infty} EA_n/n = 0$, *then (10) holds, too.*

Thus, to prove our main results we need to establish the subadditivity property for the complexity $c_n(\mathbf{t}, \mathbf{p})$ (for all texts $\mathbf{t}$ and patterns $\mathbf{p}$). The next lemma proves such a result for ℓ-convergent sequential algorithms.

Lemma 11. *An ℓ-convergent semi-sequential (or strongly semi-sequential) algorithm satisfies the basic inequality for all r such that $1 \leq r \leq n$:*

$$|c_{1,n} - (c_{1,r} + c_{r,n})| \leq m^2 + \ell m \ , \tag{12}$$

provided any comparison is done only once.

Proof. Let U_r be the smallest unavoidable position greater than r. We evaluate in turn $c_{1,n} - (c_{1,r} + c_{U_r,n})$ and $c_{r,n} - c_{U_r,n}$ (cf. Figure 1) . We start our analysis by considering $c_{1,n} - (c_{1,r} + c_{U_r,n})$. This part involves the following contributions:

- Those comparisons that are performed after position r but with alignment positions before r. We call this contribution S_1. Observe that those comparisons contribute to $c_{1,n}$ but not to $c_{1,r}$. To avoid counting the last character r twice, we must subtract one comparison. Thus

$$S_1 = \sum_{AP<r} \sum_{i \geq r} M(i, i - AP + 1) - 1.$$

- This contribution, which we call S_2, accounts for alignments AP satisfying $r \leq AP \leq U_r$ that only contribute to $c_{1,n}$, that is,

$$S_2 = \sum_{AP=r}^{U_r - 1} \sum_{i \leq m} M(AP + (i-1), i) \ .$$

- Finally, since the alignment positions after U_r on the text $\mathbf{t}_{U_r}^n$ and $\mathbf{t}_1^n$ are the same, the only difference in contribution may come from the amount of information saved from previous comparisons done on $\mathbf{t}_1^r$. This is clearly bound by

$$|c_{1,n} - (c_{1,r} + c_{U_r,n} + S_1 + S_2)| \leq m \ .$$

Now, we evaluate $c_{r,n} - c_{U_r,n}$ (see second part of Figure 1). We assume that the algorithm runs on $\mathbf{t}_r^n$ and let AP be any alignment position satisfying $r \leq AP < U_r$. The following contributions must be considered:

- The contribution S_3

$$S_3 = \sum_{AP=r}^{U_r - 1} \sum_{i} M(AP + (i-1), i)$$

  counts for the number of comparisons associated positions $r \leq AP < U_r$. This sum is the same as S_2 but the associated alignment positions and searched text positions $AP + k - 1$ may be different.

- Additional contribution may come from the alignment at position U_r. But, no more than m comparisons can be saved from previous comparisons, hence

$$|c_{r,n} - c_{U_r,n} - S_3| \leq m.$$

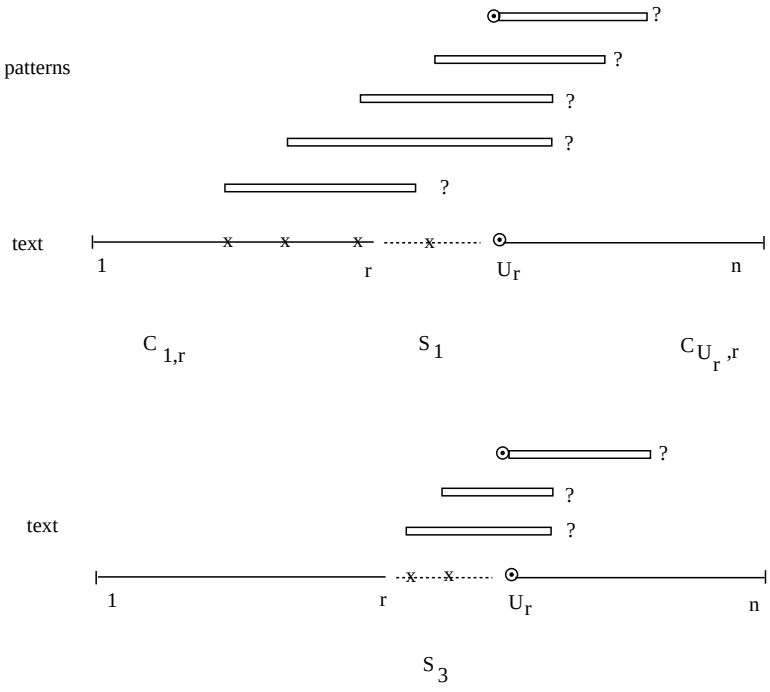


Fig. 1. Illustration to the proof of Lemma 11

To complete the proof, it remains to find upper bounds on S_1, S_2, and S_3. For $\ell \geq U_r - r$ we easily see that S_2 and S_3 are smaller than ℓm. So are their difference. With respect to S_1, for a given alignment position AP, we have $|i - AP| \leq m$. This implies that $|r - AP| \leq m$, and for any AP the index i has at most m different values. Thus, $S_1 \leq m^2$, as desired. ■

Now we are ready to prove ℓ-convergence for strongly sequential algorithms, i.e. Theorem 9. It relies on Theorem 8 so we present the **proof of Theorem 8** first. Let l be a text position such that $1 \leq l \leq n$, and r be any text position satisfying $r \leq U_l$. Let $\{A_i\}$ be the set of alignment positions defined by a strongly sequential algorithm that runs on $\mathbf{t}_r^n$. As it contains r, we may define (cf. Figure 2).

$$A_J = \max\{A_i : \ A_i < U_l\}.$$

Hence, we have $A_{J+1} \geq U_l$. Using an adversary argument, we shall prove that $A_{J+1} > U_l$ cannot be true, thus showing that $A_{J+1} = U_l$. Let $y = \max\{k : M(A_J + (k-1), k) = 1\}$, that is, y is the rightest point we can do a comparison starting from A_J We observe that we have $y \leq l$. Otherwise, according to the algorithm rule, we would have $\mathbf{t}_{A_J}^l \preceq \mathbf{p}$, which contradicts the definition of U_l. Also, since the algorithm is sequential, then $A_{J+1} \leq y + 1 \leq l + 1$. Hence $U_l = l + 1$ contradicts the assumption $A_{J+1} > U_l$ and we may assume $U_l \leq l$.

In that case, $\mathbf{p}^l_{U_l} \preceq \mathbf{p}$ and an occurrence of $\mathbf{p}$ at position U_l is consistent with the available information. Let the adversary assumes that $\mathbf{p}$ does occur. As sequence $\{A_i\}$ is non-decreasing and A_{J+1} has been chosen greater than U_l, this occurrence will not be detected by the algorithm which leads to a contradiction. Thus $A_{J+1} = U_l$, as desired. This completes the **proof of Theorem 8**.

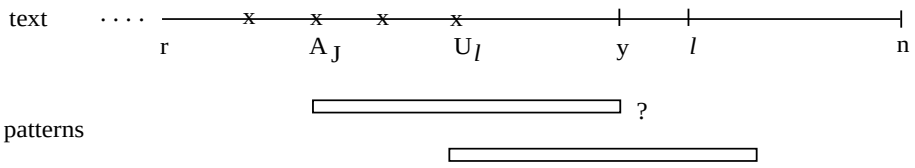


Fig. 2. Illustration to the proof of Theorem 8

Finally, we turn to the **proof of Theorem 9**. Let AP be an alignment position and define $l = AP + m$. As $|\mathbf{p}| = m$, one has $l - (m-1) \leq U_l \leq l$. Hence, $U_l - AP \leq m$ which establishes the m-convergence.

We now apply Theorem 10 to **prove Theorems 6 and 7**. After substituting $x_{1,n} = c_{1,n} + 1.5m^2 + \ell m$, we get subadditivity for any given $\mathbf{p}$ and deterministic $\mathbf{t}$. The worst case complexity results follow since

$$\max_{|\mathbf{t}|=n} c_{1,n} \leq \max_{|\mathbf{t}|=r} c_{r,n} + \max_{|\mathbf{t}|=n-r} c_{r,n} \ .$$

Now, let $\mathbf{t}_1^n$ range over the set of texts of size n, $\mathbf{t}_1^r$ and $\mathbf{t}_r^n$ range over the sets of texts of size r and $n-r$. Then, as the text distribution is stationary, the subadditivity holds in case (B). Also, the cost $C_{r,n}$ is stationary when the text distribution is. Applying Subadditive Ergodic Theorem yields (4) and (5).

We turn now to the average complexity. The uniform bound [16] on the linearity constant, allows to define $E_{\mathbf{p}}(E_{\mathbf{t}}(c_{r,n}))$, when $\mathbf{p}$ ranges over a random (possibly infinite) set of patterns. The subadditivity property transfers to $E_{\mathbf{t},\mathbf{p}}(C_n)$ and (6) follows. This completes the **proof of Theorems 6 and 7**.

4 Concluding Remarks

We consider here sequential algorithms that are variants of classical Morris-Pratt algorithms. We provided a formal definition, but the main property we use is the existence of the so called unavoidable positions in any window of fixed size (here the length of the searched pattern $\mathbf{p}$). Hence, the result extends to any algorithm that satisfies such a property [6, 13].

Nevertheless, in order to speed up the search, Boyer and Moore introduced in [5] a quite different algorithm. Given an alignment position AP, matching against $\mathbf{p}$ are checked from right to left; i.e. k is decreasing. Several variants have been proposed that differ by the amount of information saved to compute the next alignment position.

We point out here that Boyer-Moore like algorithms do not satisfy unavoidability property. We provide an example for the Horspool variant: given an alignment position AP, the next alignment position is computed by aligning the text character $\mathbf{t}[AP+m]$ with $\mathbf{t}[AP+j]$ where

$$m - j = \min\{\max\{k : \ \mathbf{p}[k] = \mathbf{t}[AP+m]\}, \ m\}$$

Let us now consider as an example $\mathbf{p} = x^4ax^2bx^2a$, $x \neq a, b$. When $\mathbf{t}[AP+m]$ is a (resp. b or x) the next alignment position is chosen to be $AP+6$ (resp. $AP+3$ or $AP+1$). When $\mathbf{t}[AP+m] \notin \{a, b, x\}$, one shifts the alignment position by m. Assume now that $\mathbf{t} = y^{10}az^4(bazbz^2)^n$ with $y \neq x$ and natural n. If the Boyer-Moore-Horspool algorithm starts with $AP = 1$, a mismatch occurs on the second comparison between $\mathbf{t}[10]$ and $\mathbf{p}[10]$ with AP shifted by 6. The same event occurs then and we eventually get the sequence $AP_i = 1 + 6i$. Assume now that we split the text at $r = 6$. As $\mathbf{t}[16]$ is b, one shifts by 3 and b is found again. Finally, one gets sequence $AP'_i = 6 + 3i$. As $\gcd(6, 3)$ does not divide 5, these two sequences are disjoint and there is no unavoidable position.

It follows that unavoidability cannot be used to prove linearity of Boyer-Moore algorithms. Nevertheless, it is clear that we assumed a very strong (and unlikely) structure on both text and patterns. In a recent paper [17], the existence of renewal points almost surely allowed to prove the existence of a linearity constant.

It is worth noticing that the Subadditive Ergodic Theorem proves the existence of the linearity constant under quite general probabilistic assumptions. The computation of the constant is difficult and only limited success was achieved so far (cf. [13, 18, 17]). However even if we cannot compute the constant, we can prove that C_n is well concentrated around its most probably value $\alpha_2 n$. Using Azuma's inequality (cf. [21]) we conclude the following.

Theorem 12. *Let the text* $\mathbf{t}$ *be generated by a memoryless source (i.e.,* $\mathbf{t}$ *is an i.i.d sequence). The number of comparisons* C_n *made by the Knuth–Morris–Pratt algorithm is concentrated around its mean* $EC_n = \alpha_2 n(1 + o(1))$*, that is,*

$$\Pr\{|C_n - \alpha_2 n| \geq \varepsilon n\} \leq 2\exp\left(-\frac{1}{2}\varepsilon^2 \frac{n}{m^2}(1 + o(1))\right)$$

for any $\varepsilon > 0$.

References

1. A. Apostolico and R. Giancarlo, The Boyer-Moore-Galil String Searching Strategies Revisited, *SIAM J. Compt.*, **15**, 98-105, 1986.
2. R. Baeza-Yates and M. Régnier, Average Running Time of Boyer-Moore-Horspool Algorithm, *Theoretical Computer Science*, **92**, 19-31, 1992.
3. P. Billingsley, *Convergence of Probability Measures*, John Wiley & Sons, New York, 1968.

4. A. Blumer, A. Ehrenfeucht and D. Haussler, Average Size of Suffix Trees and DAWGS, *Discrete Applied Mathematics*, **24**, 37-45, 1989.
5. R. Boyer and J. Moore, A fast String Searching Algorithm, *Comm. of the ACM*, **20**, 762-772, 1977.
6. D. Breslauer, L. Colussi, and L. Toniolo, Tight Comparison Bounds for the String Prefix-Matching Problem, *Proc. 4-th Symposium on Combinatorial Pattern Matching*, Padova, Italy, 11–19. Springer-Verlag, 1993.
7. R. Cole, R. Hariharan, M. Paterson, and U. Zwick, Tighter Lower Bounds on the Exact Complexity of String Matching, *SIAM J. Comp.*, **24**, 30-45, 1995.
8. L. Colussi, Z. Galil, and R. Giancarlo, On the Exact Complexity of String Matching, *Proc. 31-st Annual IEEE Symposium on the Foundations of Computer Science*, 135–143. IEEE, 1990.
9. M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, New York 1995.
10. Y. Derriennic, Une Théorème Ergodique Presque Sous Additif, *Ann. Probab.*, **11**, 669-677, 1983.
11. R. Durrett, *Probability: Theory and Examples*, Wadsworth & Brooks/Cole Books, Pacific Grove, California, 1991.
12. L. Guibas and A. Odlyzko, A New Proof of the Linearity of the Boyer-Moore String Matching Algorithm, *SIAM J. Compt.*, **9**, 672-682, 1980.
13. C. Hancart, *Analyse Exacte et en Moyenne d'Algorithmes de Recherche d'un Motif dans un Texte*, These, l'Universite Paris 7, 1993.
14. P. Jacquet and W. Szpankowski, Autocorrelation on Words and Its Applications. Analysis of Suffix Tree by String-Ruler Approach, *J. Combinatorial Theory. Ser. A*, **66**, 237-269, 1994.
15. J.F.C. Kingman, *Subadditive Processes*, in Ecole d'Eté de Probabilités de Saint-Flour V-1975, Lecture Notes in Mathematics, **539**, Springer-Verlag, Berlin 1976.
16. D.E. Knuth, J. Morris and V. Pratt, Fast Pattern Matching in Strings, *SIAM J. Compt.*, **6**, 189-195, 1977.
17. H. Mahmoud, M. Régnier and R. Smythe, Analysis of Boyer-Moore-Horspool String Matching Heuristic, *in Random Structures and Algorithms*, **10**, 169–186, 1996.
18. M. Régnier, Knuth-Morris-Pratt Algorithm: An Analysis, *Proc. Mathematical Foundations for Computer Science 89*, Porubka, Poland, *Lecture Notes in Computer Science*, **379**, 431–444. Springer-Verlag, 1989.
19. I. Simon, String Matching Algorithms and Automata, *First South-American Workshop on String Processing 93*, Belo Horizonte, Brazil, *R. Baeza-Yates and N. Ziviani,ed*, 151–157, 1993.
20. W. Szpankowski, Asymptotic Properties of Data Compression and Suffix Trees, *IEEE Trans. Information Theory*, **39**, 1647-1659, 1993.
21. M. Waterman, *Introduction to Computational Biology*, Chapman & Hall, London 1995.

This article was processed using the LaTeX macro package with LLNCS style

A Random Server Model for Private Information Retrieval

or

How to Achieve Information Theoretic PIR Avoiding Database Replication[⋆]

Yael Gertner[1], Shafi Goldwasser[2], and Tal Malkin[2]

[1] Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA 19104, USA,
ygertner@saul.cis.upenn.edu

[2] Laboratory for Computer Science, Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139, USA
{shafi,tal}@theory.lcs.mit.edu

Abstract. Private information retrieval (PIR) schemes enable users to obtain information from databases while keeping their queries secret from the database managers. We propose a new model for PIR, utilizing auxiliary random servers to provide privacy services for database access. In this model, prior to any on-line communication where users request queries, the database engages in an initial preprocessing setup stage with the random servers. Using this model we achieve the first PIR information theoretic solution in which the database does not need to give away its data to be replicated, and with minimal on-line computation cost for the database. This solves privacy and efficiency problems inherent to all previous solutions.

In particular, all previous information theoretic PIR schemes required multiple replications of the database into separate entities which are not allowed to communicate with each other; and in all previous schemes (including ones which do not achieve information theoretic security), the amount of computation performed by the database on-line for every query is at least linear in the size of the database. In contrast, in our solutions the database does not give away its contents to any other entity; and after the initial setup stage which costs at most $O(n \log n)$ in computation, the database needs to perform only O(1) amount of computation to answer questions of users on-line. All the extra on-line computation is done by the auxiliary random servers.

1 Introduction

Private Information Retrieval (PIR) schemes provide a user with information from a database in a private manner. In this model, the database is viewed as an n-bit string x out of which the user retrieves the i-th bit x_i, while giving the database no information about his query i. The notion of PIR was introduced in [10], where it was shown that if there is only one copy of the database available then $\Omega(n)$ bits of communication are needed (for information theoretic user privacy). However, if there are $k \geq 2$ non-communicating copies of the database, then there are solutions with much better (sub-linear) communication complexity. *Symmetrically Private Information Retrieval (SPIR)* [11] addresses the database's privacy as well by adding the requirement that the user, on

⋆An earlier version of this work appears as MIT technical report MIT-LCS-TR-715. This work was done with the support of DARPA grant DABT63-96-C-0018.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 200-217, 1998.

the other hand, cannot obtain any information about the database in a single query except for a single physical value.

Two major problems arise with all existing solutions: firstly, in order to achieve information theoretic privacy all previous solutions call for replicating the database into several non-communicating copies, which constitutes a serious privacy problem (as discussed below); and secondly, even though the communication complexity is sublinear, the amount of computation that the database engages in is *linear* in the size of the database for *every* query of the user. It seems unreasonable to expect a commercial database to distribute copies of its data to non-communicating entities and to perform linear amount of computation per single query solely for the purpose of the user's privacy.

In this paper, we introduce a new model for PIR (or SPIR), which allows us to achieve significant improvements both in terms of security (circumventing the replication problem) and in terms of computational complexity.

The first enhancement to the PIR model is the use of *auxiliary random servers*, whose contents are independent of the contents of the database. This separates the task of information retrieval from the task of providing privacy. We use only a single copy of the original data (the database owner itself), who does not engage in any complex privacy protocols, while all the privacy requirements are achieved utilizing the random servers, who do the work instead of the database. The random servers do not gain any information about the database or the user in the process (this is in contrast to the old model, where a database who wants to hire an agent to do all the privacy work for it must give away all its information to that agent).

The second enhancement to the model, is that we divide the PIR computation into two stages: the setup stage, which takes place ahead of query time and does not involve the user, and the on-line stage, during which the user performs his various queries. The purpose of this split of computation is to allow much of the computation to be done once ahead of time, so that during the on-line stage the database is required to engage in minimal computation and communication.

Using this model, we construct straightforward and efficient protocols for solving the two problems described above. We achieve information theoretic privacy without data replication, and we minimize the on-line computation required from the database.

Below we describe these problems and their solutions in more detail.

1.1 Problems With The Previous PIR Model

Protocols for PIR and SPIR schemes, guaranteeing information theoretic privacy, appeared in [10, 3, 15, 11]. These solutions are based on the idea of using multiple copies of the database that are not allowed to communicate with each other. This allows the user to ask different questions from different copies of the database and combine their responses to get the answer to his query, without revealing his original query to any single database (or a coalition). The recent PIR scheme of [14] uses a single database, but guarantees only computational privacy under the assumption that distinguishing quadratic residues from non-residues modulo composites is intractable. In fact, it was shown in [10] that using a single database makes it impossible to achieve information theoretic privacy with sublinear communication complexity.

Unfortunately, the common paradigm behind all the solutions that guarantee information theoretic privacy — the replication of the database in multiple separated loca-

tions — introduces a serious privacy problem to the database, *the data replication problem*. Namely, the database owner is required to distribute its data among multiple foreign entities, each of which could be broken into, or could use the data and sell it to users behind the legitimate owner's back. This is particularly problematic since the database cannot communicate with any of the other copies. Since this replication is used to protect the user's interest, it is doubtful that real world commercial databases would agree to distribute their data to completely separated holders which they cannot communicate with. Viewed from the user's standpoint, it may be doubtful that users interested in privacy of their queries would trust copies of the same database not to communicate with each other.

Secondly, the paradigm used in all existing PIR schemes requires the database to actively participate in a complex protocol in order to achieve privacy. The protocol is complex both in terms of the *computation* necessary for the database to perform in order to answer every question of the user, and in the less quantifiable *lack of simplicity*, compared to the classical lookup-the-query-and-answer approach.

In particular, in all of the existing solutions (whether using a multiple number of databases or a single one), each database performs a computation which is at least *linear* in the size of the database in order to compute the necessary answer for *each* question of the user. This is in contrast to the user's computation and the communication complexity, which are at most sublinear per query. In the single database case (computational privacy) the complexity of the database computation is a function of both the size n of the database and the size of the security parameter underlying the cryptographic assumption made to ensure privacy. Specifically, in the single database solution of [14], the computation takes a linear number of multiplications in a group whose size depends on the security parameter chosen for the quadratic residuosity problem.[1] Again, the overhead in computational complexity and lack of simplicity of existing schemes make it unlikely to be embraced as a solution by databases in practice.

1.2 New Approach: The Random Server Model for PIR

We introduce a new model for PIR, which allows for information theoretic privacy while eliminating the problems discussed above. Since it is not possible to use a single database and achieve sublinear communication complexity information theoretic results ([10]), we must still use a multiple database model. The crucial difference is that the multiple databases are *not* copies of the original database. Rather, they hold auxiliary random strings provided by, say, WWW servers for this purpose. These auxiliary servers contain strings each of which cannot be used on its own[2] to obtain any information about the original data. Thus, an auxiliary server cannot obtain information about the data, sell it to others, or use it in any other way. Instead, they may be viewed as servers who are selling security services to ordinary current day databases.

The database owner, after engaging the services of some servers for the purpose of offering private and secure access to users, performs an initial setup computation with

[1] For example, to achieve communication complexity $O(n^{\epsilon})$ the security parameter is of size $O(n^{\epsilon^2})$ and the number of multiplications is $O(\frac{1}{\epsilon}n)$.

[2] or in extended solutions, in coalition with others (the number if which is a parameter determined by the database)

the auxiliary servers. The servers are then ready to assist users in retrieving information from the database owner efficiently and privately during the on-line stage. Periodic re-initialization (setup stage) may be required in some frequency specified by the protocol. Typically this frequency will be once in a large number of queries (e.g. sublinear), or, if no periodic re-setup is required, then only when the database needs to be updated.[3]

We differentiate between two kinds of random servers: universal and tailored. Universal random servers are servers whose contents may be determined in advance, even before the setup stage, without any connection to a particular database. Tailored random servers are those who store some random string specific to the database they are serving, namely those whose content is determined during the setup stage.

One of the parameters of a PIR scheme is how many servers of each kind are required. Clearly, universal servers are preferable, since they can be prepared in advance and therefore are more efficient and more secure. Moreover, since they do not need to store any data specific to the database they are serving, they could potentially be used for multiple databases at the same time. Indeed, our strongest definition of privacy (total independence, below) requires that *all* servers involved are universal.

We define two new kinds of privacy for the database in this setting (formal definitions are in the next section), *independence*, and *total independence*.

Independence informally means that no server can get any information about the original data of the database owner. Thus, the real data is distributed among all the servers in a private way, so that no single one gets any information about it (this can be generalized to t-independence, for any coalition of upto t servers).

Total independence informally means that even *all* the auxiliary servers *jointly* do not contain any information about the original data (namely all servers are universal).

Clearly, total independence implies independence. Indeed the solutions we propose to address the latter are simpler than the ones to address the former.

1.3 Our Results

We provide general reductions, starting from any PIR scheme, to schemes that achieve independence or total independence and low database computation complexity, while maintaining the other privacy properties of the underlying starting scheme (namely user privacy and database privacy). The database computation complexity on-line is reduced to a simple O(1) look-up-the-query computation, or for some of our schemes to no computation at all. Instead, the servers assume responsibility for all computations required for privacy in the starting scheme. The user computation complexity stays the same as in the starting scheme, and (using existing solutions) it is already bounded by the communication complexity (sublinear). Therefore, we concentrate on reducing the database's computation, which in all previous schemes has been at least linear.

Let us describe our results.

Let S be a PIR scheme which requires k copies of the database,[4] and has communication complexity of C_S. We provide two sets of schemes (all terms used below are defined in section 2).

[3] Note that also in the old replication model, reinitialization is required when the database changes.

[4] Note that creating k copies of the database may be viewed as a setup stage of complexity $O(n)$.

Schemes Achieving Independence We state the result both for the interesting special case of $t = 1$ (i.e. independence), and the most general case for any t.

- A scheme achieving independence and maintaining the other privacy properties of S. The scheme uses k tailored and k universal servers, with communication complexity $O(C_S)$, and no database participation in the on-line stage (i.e. no computation).
- A scheme achieving t-independence (for any $t \geq 1$) and maintaining the other privacy properties of S. The scheme uses k tailored and tk universal servers, with communication complexity $(t+1)C_S$, and no database participation in the on-line stage.

Setup stage: The complexity of the setup stage is $O(n)$. The number of tailored servers, who need to obtain some string during setup stage, is k (the same as in the starting scheme S).

Schemes Achieving Total Independence There are two variants here.

- A (basic) scheme achieving total independence and database privacy, and maintaining user privacy up to equality between repeated queries.[5] The scheme uses $\max(k, 2)$ universal servers and the database owner, with at most $O(C_S \log n)$ communication complexity, and $O(1)$ database computation complexity.
- A scheme achieving total independence and maintaining the other privacy properties of S (in particular complete user privacy). The scheme uses $\max(k, 2)$ universal servers and the database owner, with at most $O((m + C_S) \log n)$ communication complexity, where the servers and the database need to engage in a re-setup after every m queries. The database computation is $O(1)$.

Setup stage : The complexity of the setup stage is $O(n \log n)$. Note that all servers are universal, namely they could be prepared ahead of time, and do not change during setup stage.

Tradeoff between the two versions: In the basic version the database can detect repeated queries, but cannot gain any other information about the user's queries. This is dealt with in the final scheme, where total independence is achieved preserving complete user privacy. The price for eliminating detection of repeated queries is that re-setup has to be performed every m queries. The value of m, the frequency of reinitialization, is a parameter chosen to optimally trade off the frequency and the communication complexity. A suitable choice for existing schemes is to choose $m = C_S = n^{\epsilon}$ the size of the communication complexity for a single query, so that the over all communication complexity does not increase by more than a logarithmic factor, and yet a sublinear number of queries can be made before reinitialization. Choosing between the two versions should depend on the specific application: preventing the database from detecting equal questions, or avoiding reinitialization. Note also that the first version adds database privacy even if the underlying S was not database private.

Main Idea: Note that total independence guarantees that all the auxiliary servers jointly do not contain any information about the data. So how can they assist the database

[5] namely, the only information that the database can compute is whether this query has been made before.

at all? The idea is that during the setup stage, a setup protocol is run amongst the database and the universal random servers, at the end of which the *database* is the one which changes appropriately to ensure the privacy and correctness properties for the on-line stage. During the on-line stage, as before, the user communicates with the random servers and with the database to privately extract the answer to his query.

1.4 Related Work

PIR was originally introduced by [10], who were only concerned with protecting the user's (information theoretic) privacy. In particular, for a constant number k of database copies, [10] with further improvement in [3] (for the case $k > 2$), achieve information theoretic user security with communication complexity of $n^{\frac{1}{2k-1}}$, where n is the length of the data (in bits).

Recently, [11] extended PIR to SPIR (*Symmetrically private information retrieval*), where the privacy of the data (with respect to the user) is considered as well. They use a model where the multiple databases may use some shared randomness, to achieve reductions from PIR to SPIR, paying a multiplicative logarithmic factor in communication complexity.

The work in [9] considers computational privacy for the user, and achieves a 2 database scheme with communication complexity of n^{ϵ} for any $\epsilon > 0$, based on the existence of one way functions. As mentioned earlier [14] relies on a stronger computational assumption – the quadratic residuosity problem – to achieve a 1-database PIR scheme with computational privacy and communication complexity of n^{ϵ} for any $\epsilon > 0$.

The work in [15] generalizes PIR for private information storage, where a user can privately read and write into the database. This model differs from ours, since in our model users do not have write access into the database. Still, some connection between the two models can be made, since one might consider a storage model where the first n operations are restricted to be private write (performed by the database owner), and all operations thereafter are restricted to be private reads (by users). This results in a model compatible to our model of independence (although this approach cannot lead to total independence). We note that [15] independently[6] use a basic scheme which is essentially the same as our basic RDB scheme of section 3.1. However, they use the scheme in their modified model (where users have write access), and with a different goal in mind, namely that of allowing users to privately read and write into the databases.

None of the above PIR and SPIR works consider the data replication problem.

Recently, independently from our work, [7] had suggested the commodity based model for cryptographic applications, which relies on servers to provide security, but not to be involved in the client computations. Although this model is related to ours we stress here some important differences. First, their model engages the servers only in the task of sending one message (commodity) to each client, without any interaction. In contrast, our model stresses the interaction of the servers with the clients for the purpose of reducing the computational complexity. Second, our model, unlike theirs, is designed specifically for PIR, and solves problems which were not previously addressed.

[6] our results of section 3 were actually done previously to the publication of [15]

Organization Section 2 introduces the relevant definitions and notation used. In section 3 we describe schemes achieving independence, and in section 4 we describe schemes achieving total independence.

2 Notation and Definitions

The Information Retrieval Model: The *data string* is a string of n bits denoted by $x = x_1, \ldots, x_n$. The user's *query* is an index $i \in \{1, \ldots, n\}$, which is the location of the bit the user is trying to retrieve. The *database* (also referred to as the original database, or the database owner) is denoted by D.

An *information retrieval* scheme is a protocol consisting of a *setup stage* and an *on-line* stage. During the setup stage, the auxiliary random servers are chosen, and possibly some other setup computation is performed. During the on-line stage, a user interacts with the servers and possibly also with the original database in order to obtain his query. At the end of the interaction, the user should have the bit x_i. In all our schemes, the on-line stage consists of a single round.

The Random Servers: There are two kinds of auxiliary servers: *Universal* and *Tailored*. The universal servers contain completely random data that can be prepared ahead of time independently of the particular database in mind. The tailored servers on the other hand are each independent of the database, but their content should be prepared for a particular database (during the setup stage), since the combination of all servers together is dependent on the specific database. One of the parameters for an information retrieval scheme is how many servers of each kind are required.

We require that all servers are separate, in the sense that they are not allowed to communicate with each other. We also address the case where up to t of the servers are faulty and *do* communicate with each other.

Notions of Privacy: We define the following privacy properties for an information retrieval scheme.

User privacy [10]: No single database (or server) can get any information about the user's query i from the on-line stage. That is, all the communication seen by a single database is identically distributed for every possible query. This definition can be extended to *user l-privacy*, where all communication seen by any coalition of up to l databases (servers) is identically distributed for every possible query.

Database privacy [11]: The user cannot get any information about the data string other than its value in a single location. That is, all the communication seen by the user in the on-line stage is dependent on a single physical bit x_i (so it is identically distributed for any string x' s.t. $x_i = x'_i$).

Independence: No auxiliary server has any information about the data string x. That is, the content of the auxiliary server is identically distributed for any data string x. This definition can be extended to *t-independence*, where no coalition of up to t servers has any information about x (thus, independence is the special case of 1-independence).

Total independence: All the auxiliary servers jointly have no information about the data string x, or equivalently all the servers are universal. That is, they are completely independent of the original data, and thus may all be chosen in advance.

In all the above definitions, information theoretic privacy may be relaxed to *computational* privacy, requiring indistinguishablity instead of identical distribution.

Protocols: A *private information retrieval (PIR)* scheme is one that achieves user privacy, and a *symmetrically private information retrieval (SPIR)* scheme is one that achieves user privacy and database privacy. These two protocols were defined in [10, 11], respectively. In this paper we will show how to incorporate the independence or total independence properties into PIR and SPIR schemes.

Complexity: We define the *communication complexity* of an information retrieval protocol to be the total number of bits sent between the user, the database, and the servers. The *computation complexity* (of a user/database/server during setup/on-line stage) is the amount of computation that needs to be performed by the appropriate party before the required communication can be sent. Note that we count sending bits from a specific location towards communication complexity, rather than computation complexity. Communication and computation complexity during the on-line stage refer to the complexity per each (single) query.

3 Achieving Independence: The RDB Scheme

In this section we describe a simple and efficient scheme, which takes advantage of the random server model to achieve t-independence and no database participation in the on-line stage. Specifically, we prove the following theorem.

Theorem 1. *Given any information retrieval scheme S which requires k copies of the database and communication complexity C_S, and for every $t \geq 1$, there exists an information retrieval scheme achieving t-independence and maintaining the other privacy properties (user privacy and data privacy) of S. The t-independent scheme requires $(t+1)C_S$ communication complexity and $(t+1)k$ servers, out of which only k are tailored. The setup complexity is $O(n)$ and the database is not required to participate in the on-line stage.*

An immediate useful corollary follows, setting $t = 1$:

Corollary 1. *Given any information retrieval scheme S which requires k copies of the database, there exists an information retrieval scheme achieving independence and maintaining the other privacy properties of S, which requires a factor of 2 in communication complexity, and uses k tailored servers and k universal ones. The setup complexity is $O(n)$ and the database is not required to participate in the on-line stage.*

The basic version of our reduction (the RDB scheme) is described in section 3.1. In section 3.2 we present another version, possessing some appealing extra properties for security and simplicity. We note however, that the starting point for the second reduction is any information retrieval scheme which has a linear reconstruction function. This is usually the case in existing PIR schemes (cf. [10, 3]). Finally, in section 3.3 we prove that the RDB construction satisfies theorem 1.

Another benefit of our scheme is that it does not require the participation of the database owner D after the setup stage. Instead, the servers deal with all the on-line queries and computations. Even though D is not there for the on-line stage, he is guaranteed that none of the servers who are talking to users on his behalf has any information about his data x.

3.1 The Basic RDB Scheme

In the basic RDB (random data base) scheme, instead of replicating the original database as in the underlying scheme, every copy is replaced by $t+1$ random servers whose contents xor to the contents of the database. The idea behind this replacement is that if these $t+1$ databases are chosen uniformly at random with this property, then any coalition of t of them are simply a random string, independent of the actual original data string. Therefore, t-independence is achieved . We proceed with the details of the basic reduction. The communication complexity and privacy properties of this scheme will be proved in section 3.3.

Let the underlying scheme S be a PIR scheme with k copies of the database.

Setup Stage The database owner D chooses uniformly at random $t+1$ random servers $R_1, \ldots, R_{t+1}$ in $\{0,1\}^n$, such that for every $1 \leq j \leq n$, $R_1(j) \oplus \ldots \oplus R_{t+1}(j) = D(j) = x_j$ i.e., the xor of all the servers is the original data string x. This is done by choosing k universal servers, and computing the content of another tailored server in an appropriate way. A protocol to do that is described in the appendix.

Each of these servers is then replicated k times, for a total of $k(t+1)$ servers.

Thus, at the end of the setup stage, the random servers are

$$R_1^1, \ldots, R_1^k, \ \ldots \ , R_{t+1}^1, \ldots, R_{t+1}^k$$

where $R_s^1 = R_s^2 = \ldots = R_s^k$ for every s, and where $R_1^r \oplus R_2^r \oplus \ldots \oplus R_{t+1}^r = x$ for every r.

On-Line Stage During the on-line stage, the user executes the underlying scheme S $t+1$ times, each time with a different set of k databases. The first execution is with the k copies of R_1, which results in the user obtaining $R_1(i)$. The second execution is with the k copies of R_2, resulting in the retrieval of $R_2(i)$, and so on. Finally, the user xors all the $t+1$ values he retrieved, $R_1(i) \oplus \ldots \oplus R_{t+1}(i) = D(i) = x_i$ in order to obtain his desired value x_i.

Note that the user can perform all these $t+1$ executions of S in parallel. Also, the user may either perform all these parallel executions independently, or simply use exactly the same questions in all of them. Our proofs will cover both these variants, but we prefer the latter since it simplifies the protocol of user-privacy against coalitions. However, in the most general case, if S is a multi round scheme with adaptive questions, we must use the first strategy of independent executions.

Remarks Note that out of the $k(t+1)$ servers, all but k are universal servers which can be prepared ahead of time, whereas the other k (copies of R_{t+1}) are tailored.

Another thing to note is the fact that our scheme uses replication of the random servers. At first glance, this may seem to contradict our goal of solving the data replication problem. However, in contrast to replicating the original database, replicating random servers does not pose any threat to the original data string which we are trying to protect. Thus, we manage to separate the user privacy, which requires replication, from the database privacy, which requires not to replicate the data. Still, in the next section we describe a version in which there is no replication, not even of the auxiliary servers, and which subsequently provides a higher level of privacy, as discussed below.

3.2 The RDB Scheme : Improved Variant

While the basic scheme does achieve t-independence (as no coalition of t servers has any information about x), some of the servers there are replications of each other.

Here, we propose an improvement to the basic scheme, in which a higher level of independence among the random servers is achieved, allowing for more flexibility in choosing the random servers from different providers. Specifically, we achieve t-independence among the servers, namely every combination of t servers are independent of each other (in particular, there is no replication of the servers).[7] Another benefit of this scheme over the basic one is that, while t is still the maximal size of coalition that the database is secure against, it is also secure against many other specific combinations of larger coalitions. This protocol works provided that the underlying PIR scheme has a linear reconstruction function (see 3.3), a quite general requirement that is satisfied by currently known PIR schemes.

Setup Stage Recall that in the basic version, we created $t+1$ servers and replicated each of them k times, thereby creating $t+1$ sets, each of which consist of k identical servers. In this protocol, the k servers in every set will be independent random strings, instead of replications. Specifically, the database owner D chooses *uniformly at random* $k(t+1)$ servers $R_1^1, \ldots, R_{t+1}^1, \ldots, R_1^k, \ldots, R_{t+1}^k$ with the property that $R_1^r \oplus \ldots \oplus R_{t+1}^r = x$ for every $1 \leq r \leq k$.

As in the basic scheme, kt of these servers are universal, and k are tailored. The contents of the tailored servers is computed by D using the same protocol as in the basic scheme (see appendix).

On-Line Stage During the on-line stage, the user sends his queries (according to the underlying S) to each of the servers, where $\{R_1^r, \ldots, R_{t+1}^r\}$ correspond to the r-th copy of the database in the underlying scheme S. After receiving the answers from all the $k(t+1)$ servers, the user xors the answers of $R_1^r, \ldots, R_{t+1}^r$ for each r to obtain the answer of the r-th copy in S, and combines these answers as in S to obtain his final value x_i.

The difference between this version and the basic version, is the following. In the basic scheme, the user first runs S to completion with each of the $t+1$ sets of servers (for example one set is $R_1^1, \ldots R_1^k$) giving the user $t+1$ values that enable him to xor them all together and obtain the value of the primary database. In contrast, here the user first combines answers by xoring values he received (for example from $R_1^1, \ldots R_{t+1}^1$) in the middle of the S protocol, which gives the user the intended answer of each copy of the database, and only then combines the answers as needed in S.

Thus, to succeed in this version, the underlying S must have the following closeness property under xor: *If $f_r(x,q)$ is the function used by the r-th copy of the database in S to answer the user's query q with the data string x, and given $y_1, \ldots, y_m$, then* $f_r(y_1,q) \oplus \ldots \oplus f_r(y_m,q) = f_r(y_1 \oplus \ldots \oplus y_m, q)$. This may be generalized to any underlying scheme with a linear reconstruction function. This requirement is very general,

[7] Moreover, if we assume that the original data x is randomly distributed, then the servers are $2t+1$ independent.

and is usually satisfied by existing PIR protocols (for example, protocols based on xoring subsets of locations in the data string, such as [10, 3], the best PIR schemes known to date).

3.3 Analysis of the RDB Scheme: Proof of Theorem 1

We now analyze the RDB scheme in terms of complexity, correctness, and privacy, to show that it satisfies the bounds given in theorem 1.

The RDB scheme requires a multiplicative factor of $(t+1)$ in communication complexity over the underlying scheme S, since S is simply executed $t+1$ times. Typically, t is a constant $t \geq 1$, which means the communication complexity of RDB is $O(C_S)$, where C_S is the communication complexity of S. The number of tailored servers required is the same as the number of databases required in S, since all the tuples $R_1, \ldots, R_t$ can be prepared in advance, and then they can be xored with the original data to produce R_{t+1}. Thus, one tailored server is needed per one copy of the database in the underlying S.

It is not hard to check that the scheme gives the user the correct value x_i, because of the way the servers were chosen, and from the correctness of S.

User privacy properties carry from S, namely if S was user-l-private (i.e. user private against coalitions of up to l databases), then so is the corresponding RDB scheme (where user privacy is protected from any coalition of l servers). This is clear for coalitions involving servers from the same set $R_s^1, \ldots, R_s^k$ for some s, since the user simply runs S with the set. This argument immediately extends to arbitrary coalitions if the user sends exactly the same questions in all sets (i.e. in every execution of S).[8] In the case of parallel independent executions and a multi round adaptive S, a little more care is needed to show that the view of any coalition is independent of i, using the l-user-privacy of S inside sets, and the independence of the executions across sets.

Database privacy of S also implies database privacy of the corresponding RDB scheme, as follows. If S is database private (SPIR), then in the r-th parallel execution of S the user gets at most one bit, and altogether the user gets at most $(t+1)$ bits. Since these are chosen uniformly at random among all strings that xor to x, it follows that if the $(t+1)$ bits are from the same location i in all servers, they are distributed uniformly over all $(t+1)$-tuples that xor to x_i, and otherwise the $(t+1)$ bits are distributed randomly among all possible tuples. In any case, the user's view depends on at most one physical bit of x, and database privacy is maintained.

Finally, the RDB scheme achieves t-independence since any coalition of up to t servers contains only t or less of the servers in $R_1^r, \ldots, R_{t+1}^r$, and thus (from the way the auxiliary databases were defined), the coalition consists of a string uniformly distributed over all strings of appropriate length, independent of x.

4 Achieving Total Independence: The Oblivious Data Scheme

In this section we present a scheme for total independence PIR (or total independence SPIR), where *all* auxiliary servers are universal, i.e. jointly independent of the database. This scheme also achieves O(1) computation complexity for the database.

[8] This strategy is always possible unless S is a multi round adaptive scheme.

Overview of Results We first describe a basic version of our scheme, which achieves total independence, as well as database privacy, but maintains the user privacy with one exception: in repeated executions of the basic scheme, the database can tell whether the questions in different executions correspond to the same index or not. We prove that no other information about the content of the queries or the relations among them is revealed to the database. We call this *user privacy up to equality between repeated queries*. Thus, we prove the following theorem.

Theorem 2. *Given any PIR scheme S which requires k copies of the database and communication complexity C_S, there exists a total independence SPIR scheme, private for the database and private for the user up to equality between repeated queries, which uses $\max(k, 2)$ universal servers, and requires communication complexity of at most $O(C_S \log n)$. The setup complexity is $O(n \log n)$, and the on-line computation complexity of the database is $O(1)$.*

The scheme is described in section 4.1, and in section 4.2 we prove that it satisfies the theorem.

Since the information of whether users are asking the same question or not may in some applications be an important information that violates the user privacy, we present a generalized version of our scheme in section 4.3, which completely hides all information about the user queries, even after multiple executions. This scheme maintains the privacy properties of the underlying scheme, namely it transforms a PIR scheme into a total independence PIR scheme, and a SPIR scheme into a total independence SPIR scheme. The price we pay for eliminating the equality leakage, is that the setup stage needs to be repeated every m queries, and an additive factor of $m \log n$ is added to the communication complexity, where m is a parameter to the scheme (see 4.3 for how to choose m). Thus, we prove the following theorem.

Theorem 3. *Given any information retrieval scheme S which requires k copies of the database and communication complexity C_S, there exists a total independence information retrieval scheme, maintaining the privacy properties (user privacy and database privacy) of S, which uses $\max(k, 2)$ universal servers, and requires communication complexity of at most $O((m + C_S) \log n)$, where m is the number of queries allowed before the system needs to be reinitialized. The setup complexity is $O(n \log n)$, and the on-line computation complexity of the database is $O(1)$.*

The following corollary is obtained by setting $m = n^\epsilon$ in the above theorem, where n^ϵ is some polynomial equal to the communication complexity of the underlying PIR scheme (it is conjectured in [10] that all information theoretic PIR schemes must have communication complexity of at least $\Omega(n^\epsilon)$ for some ϵ).

Corollary 2. *Given any information retrieval scheme S which requires k copies of the database and has communication complexity $O(n^\epsilon)$, there exists a total independence information retrieval scheme, maintaining the privacy properties of S, which uses $\max(k, 2)$ universal servers, requires communication complexity of $O(n^\epsilon \log n)$, and has to be reinitialized after every $O(n^\epsilon)$ number of queries.*

It is not clear which of the two schemes – the one achieving privacy up to equality (plus database privacy), or the one achieving full privacy but with periodic setups – is better. This depends on the particular needs of the application.

The Main Idea: Oblivious Data Recall that in order to achieve information theoretic PIR a number of multiple servers is required. On the other hand in order to achieve total independence PIR, all auxiliary servers must be (jointly) independent of the data. To accommodate these two seemingly conflicting requirements we use the following idea. During the setup stage, the database and the auxiliary servers create a new "oblivious" string y which depends on the content of all of them. This string must be held by the database D (since all others cannot hold any data dependent on x). Thus, we let the database change during the setup stage, rather than the servers. Later, during the on-line stage, the user interacts with the servers to obtain information about the relation between y and x. Knowing this information the user can simply ask D for the value of y in an appropriate location, whose relation to x he knows from communication with the servers, which enables him to compute x_i. We call y an *oblivious* data string, since it should be related to the data string x, yet in a way which is oblivious to its holder D, so that D cannot relate the user's query in y to any query in x, and therefore learns nothing about the user's interests from the user's query in y. Note that all the database's work is in the setup stage (which amounts to only a logarithmic factor over the work that needs to be done to replicate itself in the old model). During the on-line stage, however, all D needs to do is to reply with a bit from the oblivious string which requires no computation.

4.1 Basic Scheme

Let the underlying scheme S be a PIR scheme with k copies of the database.

Setup Stage The (universal) auxiliary servers are k servers each containing a random string $r \in \{0,1\}^n$, and a random permutation $\pi : [1..n] \to [1..n]$ (represented by $n \log n$ bits in the natural way). D and two of the servers R_1, R_2 engage in a specific multi party computation, described below, at the end of which D obtains the oblivious data string

$$y = \pi(x \oplus r)$$

but no other information about r, π. Each server does not obtain any new information about x.

Naturally, by the general multi-party theorems of [6, 8], such setup stage protocol exist, but are very expensive. Instead, we design a special purpose one-round efficient protocol for this purpose.

The multi party computation is done as follows: D chooses uniformly at random two strings x^1 and x^2 such that $x^1 \oplus x^2 = x$. Similarly, R_1 chooses uniformly at random r^1, r^2 such that $r^1 \oplus r^2 = r$. R_2 chooses uniformly at random π^1, π^2 such that $\pi^1 \circ \pi^2 = \pi$, where $\circ$ is the composition operator (that is, $\pi^2(\pi^1(\cdot)) = \pi(\cdot)$). The following information is then sent between the parties on secure channels:

$$\begin{array}{l|l|l} R_2 \to R_1 : \pi^1 & R_1 \to R_2 : r^2 & R_1 \to D : v = \pi^1(r^1 \oplus x^1) \\ D \to R_1 : \ x^1 & D \to R_2 : \ x^2 & R_2 \to D : pi^2, u = \pi(r^2 \oplus x^2) \end{array}$$

D can now compute $y = \pi^2(v) \oplus u = \pi(r^1 \oplus x^1) \oplus \pi(r^2 \oplus x^2) = \pi(r \oplus x)$ ("the oblivious string"). R_1 and R_2 discard all communication sent to them during the setup stage, and need to maintain only their original independent content.

At the end of the setup stage the database D has two strings: x which is the original data string, and also y which is the oblivious data. The auxiliary servers contain the same strings as before the setup stage, and do not change or add to their content.

On-Line Stage In the on-line stage the user first runs S (the underlying PIR scheme) with the servers to retrieve the block $(j := \pi(i)\ ,\ r_i)$, as specified below (recall that r_i is the random bit with which the user's desired data bit was masked, and that j is the location of the masked bit in the oblivious data string). Then the user queries D for the value at the j-th location y_j. This is done by simply sending j to D on the clear, and receiving the corresponding bit y_j back. To reconstruct his desired bit, the user computes $y_j \oplus r_i = [\pi(x \oplus r)]_j \oplus r_i = (x \oplus r)_i \oplus r_i = x_i$.

Since S is a PIR scheme for retrieving a single bit, we need to specify how to retrieve the required block. The most naive way is to apply S $\log n+1$ times, each time retrieving a single bit out of the $n(\log n + 1)$ bits. However this way does not necessarily maintain database privacy, and is not the best in terms of communication complexity. This can be improved by noticing that each of the $\log n + 1$ bits required belongs to a different set of n bits. Thus, the online stage can be performed by $\log n + 1$ parallel applications of S for one bit out of n. Further improvements are possible when methods for block retrieval which are more efficient than bit by bit are available (cf. [10]).

Note that the computation complexity for the database here is minimal – $O(1)$. In fact, the only thing required from D is to send to the user a single bit from the specified location.

Remarks Two questions may arise from our setup stage. First, can the setup stage be achieved using only a single server and the database? This would change the required number of servers in the scheme to k instead of $\max(k, 2)$. Second, and more important, note that during our setup stage if R_1 and R_2 collude, together they can find out the data. So, to guarantee total independence, they should discard the communication sent to them during the setup stage.[9] Can we construct a different protocol for the setup stage which avoids this problem? The following lemma helps us answer the above questions, by showing that it is impossible to achieve our setup stage with only two parties.

Before stating the lemma, let us informally describe what it means for a 2-argument function to be privately computable, in the information theoretic model, and with honest players (for formal definitions and treatment see [17, 6, 8, 13]). A function $f([x_1], [x_2]) = ([y_1], [y_2])$ is *privately computable* if there exist a protocol for two players P_1, P_2, as follows. At the beginning of the protocol P_1 holds x_1 and P_2 holds x_2. during the protocol the players may flip coins and alternately send messages to each other. At the end of the protocol, each player P_i $(i = 1, 2)$ can use the communication and his input x_i to reconstruct his output y_i (*correctness*), but cannot obtain any other information about the other party's input, that does not follow from his own input and output (*privacy*).

Lemma 1. *The two-argument function* $f([\pi, r], [x]) = ([\emptyset], [\pi(x \oplus r)])$ *is* not *privately computable (in the information theoretic model).*

Proof. See appendix. □

From the lemma, it is clear that our setup stage cannot be achieved with a single server and the database, since a multi-party computation is needed (rather than a two-party computation). This is not a real problem though, since if we want information theoretic privacy, we must have $k \geq 2$ in the PIR scheme to begin with, and thus $\max(k, 2) =$

[9] Note that if the servers do not discard the communication, the privacy and independence are not compromised, but the total independence is replaced by simple independence (and may be extended to t-independence).

k is optimal number of servers. If however we are willing to settle for computational privacy, then we can achieve the setup stage with a single server.

As for the second question, note that the lemma implies that there cannot exist a protocol (even with arbitrary number of servers), such that the database obtains $\pi(r \oplus x)$ and the servers *jointly* obtain no information about x. This is because if we consider the information obtained by a coalition of all servers, this is reduced to a two party protocol. Thus, our setup stage cannot be improved in this sense, with respect to our function $\pi(r \oplus x)$.

4.2 Analysis of the Basic Oblivious Scheme: Proof of Theorem 2

We now analyze the oblivious scheme in terms of complexity, privacy, and correctness, to show that it satisfies the bounds given in theorem 2.

It is not hard to verify that the setup stage computation is correct, namely that indeed $y = \pi(x \oplus r)$. Now the **correctness** of the scheme follows from the correctness of the underlying S: since the user uses S to obtain r_i and $j = \pi(i)$, it follows that $y_j = x_i \oplus r_i$ and thus $x_i = y_j \oplus r_i$.

The **communication complexity** of the scheme is at most $(\log n + 1)C_S(1, n) + \log n + 1$, where $C_S(l, n)$ is the communication complexity that the underlying scheme S requires to retrieve a block of l bits out of n bits. This expression is based on a bit by bit retrieval, as discussed above. Alternatively, any other method for retrieving blocks can be used, yielding communication complexity $C_S(\log n + 1, n(\log n + 1)) + \log n + 1$, which may be lower than the general expression. The **computation complexity** of the database is $O(1)$ during the on-line stage because it needs to send only one bit of information to the user. During the **setup stage** the computation of the database involves linear computation complexity which is similar to the amount of work it needs to do in order to replicate itself in the original PIR model. The communication complexity of the setup stage is $O(n \log n)$, which is a factor of $\log n$ over the $O(n)$ of existing PIR algorithms, where the database has to be replicated.

Total independence is clearly achieved, since the auxiliary servers may all be predetermined in advance, and do not change their content after setup stage.

Database Privacy is also guaranteed by our scheme, even if the underlying S is not database private. This is because, no matter what information the user obtains about π and r, this information is completely independent of the data x. The user gets only a single bit of information which is related to x, and this is the bit y_j at a certain location j of the user's choice. Note that since $y = \pi(x \oplus r)$, the bit y_j depends only on a single physical bit of x.

User Privacy with respect to the servers follows directly from the user privacy of the underlying scheme, and user privacy with respect to D is maintained in a single execution of the scheme, and in multiple executions up to equality between queries, as we prove below. However, if in multiple executions two users are interested in the same query i, the database will receive the same query $j = \pi(i)$, and will thus know that the two queries are the same. This will be dealt with in section 4.3. We proceed in proving user privacy up to equality with respect to the database D.

Consider an arbitrary sequence $(i_1, \ldots, i_m)$ of query indices which are all distinct. We will prove that the distribution of D's view after the setup stage and m execution of the on-line stage with these indices is independent of the values $i_1, \ldots, i_m$.

Lemma 2. *Let V_{setup} be the view of D after performing the setup stage. For every permutation $\hat{\pi} : [1..n] \to [1..n]$, $Prob[\pi = \hat{\pi} \mid V_{\text{setup}}] = \frac{1}{n!}$ where probability is taken over all the random setup choices π, π^1, r, r^1. In particular, D does not get any information about the permutation π from the setup stage.*

Proof. D's view consists of $V_{\text{setup}} = [x^1, x^2, \pi^2, v = \pi^1(r^1 \oplus x^1), u = \pi(r^2 \oplus x^2)]$. Given this view, every choice for a permutation π fixes the choices of π^1, r, r^1. That is, every $\hat{\pi}$ corresponds to a single choice $(\hat{\pi}, \hat{\pi}^1, \hat{r}, \hat{r}^1)$ which generates the given view. Since all these random choices of the setup stage are done uniformly and independently of each other, each such choice is equally likely. Thus, the probability of a particular $\bar{\pi}$ is $\frac{1}{n!}$. □

Lemma 3. (User Privacy) *Let $(i_1, \ldots, i_m)$ be a tuple of distinct indices in $[1..n]$. Let V_{setup} be the view of D after the setup stage, and $V(i_1, \ldots, i_m)$ be the view of D for m executions of the on-line stage with queries $i_1, \ldots, i_m$. Then for every tuple $(j_1, \ldots, j_m)$ of distinct indices, and for every setup view V_{setup},*

$$Prob[V(i_1, \ldots, i_m) = (j_1, \ldots, j_m) \mid V_{\text{setup}}] = \frac{(n-m)!}{n!}$$

where probability is taken over all the random choices π, π^1, r, r^1. In particular, the view is independent of the user queries.

Proof. Since after the setup stage D did not get any information about π, as proved in proposition 2, every π is equally likely, and thus the given tuple $(j_1, \ldots, j_m)$ may correspond to any original queries tuple $(i_1, \ldots, i_m)$ with equal probability. A formal derivation follows. Denote by $\Pi = \{\pi \mid \pi(i_1, \ldots, i_m) = (j_1, \ldots, j_m)\}$.
$Prob[V(i_1, \ldots, i_m) = (j_1, \ldots, j_m) \mid V_{\text{setup}}] = \sum_\pi Prob[\pi \mid V_{\text{setup}}] Prob[V(i_1, \ldots, i_m) = (j_1, \ldots, j_m) \mid V_{\text{setup}}, \pi] = \sum_{\pi \in \Pi} Prob[\pi \mid V_{\text{setup}}] = \sum_{\pi \in \Pi} \frac{1}{n!} = \frac{(n-m)!}{n!}$ □

We proved that any two tuples of distinct queries $(i_1, \ldots, i_m)$ and $(i'_1, \ldots, i'_m)$ induce the same distribution over the communication $(j_1, \ldots, j_m)$ sent to D. Therefore, the basic scheme is user private up to equality.

4.3 Eliminating Detection of Repeated Queries: Proof of Theorem 3

In order for the oblivious database scheme to be complete we need to generalize the basic scheme so that it guarantees user privacy completely and not only up to equality between repeated executions. To extend it to full privacy we need to ensure that no two executions will ever ask for the same location j. To achieve this, we use a buffer of some size m, in which all (question,answer) pairs (j, y_j) that have been queried are recorded. The on-line stage is changed as follows: the user who is interested in index $\hat{i}$ first obtains the corresponding $r_{\hat{i}}, \hat{j}$ from the servers similarly to the basic version. He does that by running S to obtain the bit $r_{\hat{i}}$, and (in parallel) using the most efficient way available to obtain the block $\hat{j}$ (again, a possible way to do it is by running $S(1, n) \log n$ times).[10] Then the user scans the buffer. If the pair $(\hat{j}, y_{\hat{j}})$ is not there, the user asks D for $y_{\hat{j}}$ (as in the basic scheme). If the desired pair is there, the user asks D for y_j in some random location j not appearing in the buffer so far. In any case, the pair (j, y_j) which was asked from D is added to the buffer.

Clearly, a buffer size m results in an additive factor of $m \log n$ in the communication complexity over the basic scheme. On the other hand, after m executions the buffer is full, and the system has to be reinitialized (i.e. the setup stage is repeated, with new r, π). Thus, we want to choose m as big as possible without increasing the communication complexity much. A suitable choice for existing schemes will therefore be $m = n^\epsilon$ the

[10] so far we are doing the same as in the basic scheme, except we insist that r_i is retrieved separately from $\hat{j}$. This is done in order to maintain database privacy in case the underlying S is database private, as proved below, and it does not change the communication complexity.

same as the communication complexity of the underlying S. This only increases communication complexity by a constant factor, and still allows for polynomial number of executions before reinitialization is needed. We note that in many practical situations, reinitialization after m steps is likely to be needed anyway, as the database itself changes and needs to be updated.

The database privacy in this case depends on the underlying scheme S: If S is database private (a SPIR scheme), then so is our scheme. This is because, when running S, the user gets only a single physical bit r_i out of r. Now, no matter how much information the user obtains about π or y (either from direct queries or from scanning the buffer), the data x is masked by r (namely $y = \pi(x \oplus r)$), and thus the user may only obtain information depending on a single physical bit of x.

The other privacy and correctness properties can be verified similarly to the basic scheme proofs of section 4.2.

References

1. M. Abadi, J. Feigenbaum, J. Kilian. On Hiding Information from an Oracle. JCSS, 39:1, 1989.
2. N. Adam, J. Wortmann. Security Control Methods for Statistical Databases: a comparative study, ACM Computing Surveys, 21(1989).
3. A. Ambainis. Upper bound on the communication complexity of private information retrieval. ICALP 97.
4. D. Beaver, J. Feigenbaum. Hiding instances in Multioracle Queries. STACS,1990.
5. D. Beaver, J. Feigenbaum, J. Kilian, P. Rogaway. Security with Low Communication Overhead. CRYPTO 1990.
6. M. Ben-Or, S. Goldwasser, A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. STOC 1988.
7. D. Beaver. Commodity Based Cryptography. STOC 1997.
8. D. Chaum, C. Crepeau, I. Damgaard. Multiparty Unconditionally Secure Protocols. STOC 1988.
9. B. Chor, N. Gilboa. Computationally Private Information Retrieval. STOC 1997.
10. B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan. Private Information Retrieval. FOCS 1995.
11. Y. Gertner, Y. Ishai, E. Kushilevitz, T. Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. STOC 1998.
12. O. Goldreich, S. Micali, A. Wigderson. How to Solve any Protocol Problem. STOC 1987.
13. E. Kushilevitz. Privacy and Communication Complexity. FOCS 1989.
14. E. Kushilevitz, R. Ostrovsky. Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. FOCS 1997
15. R. Ostrovksy, V. Shoup. Private Information Storage. STOC 1997.
16. P. Tendick, and N. Natloff. A modified Random Perturbation Method for Database Security. ACM Transactions on Database Systems, 19:1, pp.47-63, 1994.
17. A. C. Yao. Protocols for Secure Computations. FOCS 1982.

A Setup Stage of RDB

For the basic RDB scheme, the database owner D chooses uniformly at random $t+1$ random servers $R_1, \ldots, R_{t+1}$ in $\{0,1\}^n$, such that for every $1 \leq j \leq n$, $R_1(j) \oplus \ldots \oplus R_{t+1}(j) = D(j) = x_j$ i.e., the xor of all the servers is the original data string x. This is done by first choosing $R_1, \ldots, R_t$ containing completely random strings (universal servers), and then using the following protocol which allows D to prepare an appropriate content for the tailored server R_{t+1}. Since we do not allow the servers to gain any information about each other, the result of this computation should only go to the database. One possible way would be to let the database read the content of all universal servers, but this would give the database much more information than it needs, which may be

a source for future security problems.[11] Thus, we use a simple multi-party protocol for computing the xor, at the end of which D learns R_{t+1} but no other information, and the servers do not learn any new information.

Computing $R_{t+1} = R_1 \oplus \ldots \oplus R_t \oplus x$: Each of the servers R_s ($1 \leq s \leq t$) first shares its content among all others and D, by choosing uniformly at random $t+1$ shares $a_{s1}, \ldots, a_{st}, a'_s$ that xor to R_s. Each a_{sj} is sent to R_j, and a'_s is sent to D. Next, every server xors all shares sent to it from all other servers, and sends the result to D, who now xors all the messages and x, to obtain the desired content for R_{t+1}.

For the RDB scheme of section 3.2, the same summing protocol is performed k times (computing R^r_{t+1} for every $1 \leq r \leq k$).

B Proof of Lemma 1

Assume towards contradiction that there is a protocol that privately computes the function $f([\pi, r], [x]) = ([\emptyset], [\pi(x \oplus r)])$. That is, before the protocol starts, player P_1 holds (π, r) and player P_2 holds x. During the protocol P_1, P_2 may flip sequences of random coins, denoted c_1, c_2 respectively. Denote the communication exchanged between the players by $\mathtt{comm} = \mathtt{comm}(\pi, r, c_1, x, c_2)$. At the end of the protocol, P_1 gets no information about x; P_2 may apply a reconstruction function $g(\mathtt{comm}, x, c_2) = \pi(r \oplus x)$ to obtain his output; and no other information about (π, r) is revealed to P_2. Now, when r is chosen uniformly $(x, \pi(r \oplus x))$ gives no information about π, and thus P_2 obtains no information about π. In particular, this means that given P_2's view $(x, c_2, \mathtt{comm})$, any permutation could have generated the given communication $\mathtt{comm}$, or more formally,

$$\forall x, c_2, \forall c_1, r, \pi, \pi', \exists r', c'_1 \quad \mathtt{comm}(\pi, r, c_1, x, c_2) = \mathtt{comm}(\pi', r', c'_1, x, c_2) \qquad (1)$$

We will show that this implies that P_1 can obtain information about x from the communication, which contradicts the privacy of the protocol. Let P_2 conduct the following mental experiment on his view $(\pi, r, c_1, \mathtt{comm})$. P_2 sets π' to be the identity permutation, and finds r', c'_1 that yield the same communication $\mathtt{comm}$ (such r', c'_1 exist by (1)). Now, since $\pi(r \oplus x) = g(\mathtt{comm}(\pi, r, c_1, x, c_2), x, c_2) = g(\mathtt{comm}(\pi', r', c'_1, x, c_2), x, c_2) = \pi'(r' \oplus x) = (r' \oplus x)$, P_2 can find out that x satisfies the equation

$$x = r' \oplus \pi(r \oplus x). \qquad (2)$$

Since(2) cannot hold for all x (unless $n = 1$), we have shown that P_2 obtains some information about x, which concludes the proof. □

[11] e.g. in a setting where the same universal random servers may be used by multiple applications.

Almost Optimal (on the average) Combinatorial Algorithms for Boolean Matrix Product Witnesses, Computing the Diameter (Extended Abstract)

C.P. Schnorr[1] and C.R. Subramanian[2]

[1] Fachbereich Mathematik/Informatik, Universität Frankfurt, Germany.
email: schnorr@research.bell-labs.com
[2] Max-Planck Institute für Informatik, Saarbrücken, 66123, GERMANY.
email: crs@mpi-sb.mpg.de

Abstract. We describe almost optimal (on the average) combinatorial algorithms for the following algorithmic problems : **(i)** computing the boolean matrix product, **(ii)** finding witnesses for boolean matrix multiplication and **(iii)** computing the diameter and all-pairs-shortest-paths of a given (unweighted) graph/digraph. For each of these problems, we assume that the input instances are drawn from suitable distributions. A random boolean matrix (graph/digraph) is one in which each entry (edge/arc) is set to 1 or 0 (included) independently with probability p. Even though fast algorithms have been proposed earlier, they are based on algebraic approaches which are complex and difficult to implement. Our algorithms are purely combinatorial in nature and are much simpler and easier to implement. They are based on a simple combinatorial approach to multiply boolean matrices. Using this approach, we design fast algorithms for **(a)** computing product and witnesses when A and B both are random boolean matrices or when A is random and B is arbitrary but fixed (or vice versa) and (b) computing diameter, distances and shortest paths between all pairs in the given random graph/digraph. Our algorithms run in $O(n^2(\log n))$ time with $O(n^{-3})$ failure probability thereby yielding algorithms with expected running times within the same bounds. Our algorithms work for all values of p.

1 Introduction

Given two boolean matrices A and B of dimension nxn each, their product C is defined as $c_{i,j} = \vee_k (a_{i,k} \wedge b_{k,j})$. If $c_{i,j} = 1$, then a *witness* for this fact is any index k such that $a_{i,k} = b_{k,j} = 1$. The *witness problem* is to find one witness for each $c_{i,j}$ which has a witness. Computing boolean matrix multiplication and the associated witness problem naturally arises in solving many path problems like shortest paths [2, 3, 10, 16], verifying the diameter of a given graph [6].

The only known approach to solving the boolean matrix multiplication (*BMM*) problem is by reducing to matrix multiplication over integers by treating each 0,1-entry as an integer. $c_{i,j} = 1$ as per *BMM* if and only if $c_{i,j} > 0$ as

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 218-231, 1998.

per integer matrix multiplication. The currently best algorithm for multiplying matrices over arbitrary rings is due to Coppersmith and Winograd [9] and runs in $O(n^{\omega})$ time where $\omega = 2.376...$

Recently, Alon and Naor [3] and independently Galil and Margalit [10] have shown how by repeatedly using matrix multiplication, one can solve the boolean matrix witness problem (*BMWP*) in $O(n^{\omega}(\log n)^{O(1)})$ time. The approach used in [3] is derandomizing a simple randomized algorithm and is based on some recent results on small size almost c-wise independent probability spaces.

The previous approaches to multiplying arbitrary matrices naturally use algebraic techniques and are quite complex and very difficult to implement. But, the *BMM, BMWP* problems are basically combinatorial in nature. For each i, j, one has to select a proper index k (if it exists) such that $a_{i,k} = b_{k,j} = 1$. There is no forcing argument which suggests that a combinatorial approach to these problems would not work efficiently. In view of the complex nature of the algebraic approach, it is desirable to have a combinatorial approach for solving these problems.

Basch, Khanna and Motwanit [6] present a combinatorial approach that solves these problems in $O(n^3/(\log n)^2)$ time. This works by dividing the columns of A and the rows of B into $\log n$ sized groups and treating each $\log n$-length 0,1-vector as an integer between 0 and n. This helps them to obtain a $O(\log n)$ improvement over the $n^3/(\log n)$ algorithm by Arlazarov, et. al. [5]. Other than these, there is no subcubic algorithm (based on combinatorial approach) for these two problems. As of now, it is not even clear if *BMM, BMWP* problems have the same complexity.

In this paper, we present an alternative approach[1] to solve *BMM, BMWP* problems. So far, we have not been able to provide a worst-case guarantee of $O(n^{3-\epsilon})$ time. However, using our approach, we design almost optimal $O(n^2(\log n))$ time algorithms for multiplying two random boolean matrices and also for finding witnesses. We can also achieve the same within the same time bounds, when one of the matrices is random and the other is an arbitrary but fixed matrix. The failure probability of our algorithms is $O(n^{-3})$ leading to $O(n^2(\log n))$ expected time algorithms for both problems. It should be noted that the two results (on both matrices being random and only one is random) are independent and do not imply one another. These results provide strong evidence that boolean matrix problems are probably efficiently solvable by our or other combinatorial approaches.

One important feature of our approach is that it takes into account the density of the 1s in the input matrices whereas the algebraic approach seems indifferent to the density of 1s. By treating $O(\log n)$-length 0,1-vectors as small integers

[1] The first author observed [15] this alternative approach and applied it to derive an $O(m+n(\log n))$ average time algorithm to compute Ab where A is a random boolean matrix and b is a *fixed* column vector. Here m refers to the expected number of 1's in A. Independently, the second author observed the same approach and used it to derive the results of Sections 2,3,4,5 and 6. Upon knowing each other's work, it was decided to write this joint paper.

as is done in [6], our approach can also be guaranteed to have $O(n^3/(\log n)^2)$ worst-case running time. For some special classes of matrices, we obtain much better running times.

Using our results on boolean matrices, we show how to compute the diameter and all-pairs-shortest paths of a random graph/digraph in $O(n^2(\log n))$ time with probability $1 - o(1)$. Previously, the best-known algorithms (without using matrix multiplication) require $O(n^3/(\log n)^2)$ time to verify if the diameter of a given graph is d or less, for fixed d. A shortest path between a pair i, j is implicitly specified by its first vertex (if any) other than i and j. Most of the previous work on average case analysis of shortest path algorithms have been only over *randomly weighted complete digraphs*, randomness was only in the weights and not in the presence of edges. Given such a graph, the idea is to prove that it is sufficient to consider only $O(n(\log n))$ edges to find all shortest paths and apply standard shortest path algorithms. On the contrary, in our model, we introduce randomness about the presence of edges and difficulties arise only when we have $\Omega(n(\log n))$ edges. The two models are not comparable. To the best of our knowledge, our algorithmic results on finding shortest paths constitute the first work of this nature.

2 *A*-columns-*B*-rows Approach

In this paper, we describe a different approach to studying boolean matrix problems. The usual combinatorial approach can be termed as A-rows-B-columns approach. It works by considering a row of A and a column of B and looking for matching positions in these n-vectors. This follows the definition of *BMM*. On the other hand, in our approach, we consider any index k and consider the k-th column of A (denoted A_{*k}) and the k-th row of B (denoted B_{k*}). For any i, j such that $a_{i,k} = 1 = b_{k,j}$, k is a witness for the pair i, j and hence $c_{i,j} = 1$. That is, every pair of 1s in A_{*k} and B_{k*}, is witnessed by k. This implies, if A_{*k} and B_{k*} each have $\Omega(n)$ 1s, then we get a witness for each of $\Omega(n^2)$ pairs in in constant time per pair. If we repeat this for every k, we get witnesses for all pairs. We call this method *A-columns-B-rows* approach.

One drawback is that any pair will be counted for each of its witnesses. That is, if a pair has several witnesses, then it will be counted several times resulting in redundant work. However, this approach gives faster combinatorial algorithms than previously possible for some speciall classes of boolean matrices.

First, we do a preprocessing during which we group together all 1s in each A_{*k} in a binary search tree A_k. Similarly, we group together all 1s in each B_{k*} in a search tree B_k.[2] The information stored in the search trees include the row and column numbers as well. Then, we compute the sizes $n_A(k)$ of each A_k and

[2] Sometimes, it is useful to store the sets A_k, B_k as doubly-linked lists. For both ways of storage, all pairs (i, j) $(i \in A_k, j \in B_j)$ of elements can be enumerated in $O(|A_k| \cdot |B_k|)$ time. When there is a need for searching the sets, it is useful to store A_k, B_k as binary search trees, even though it takes $O(|A_k| \cdot (\log |A_k|))$ time to build the search trees.

$n_B(k)$ of each B_k. There are exactly $n_A(k) \cdot n_B(k)$ pairs having k as a witness, for each k.

If $\sum_k (n_A(k) \cdot n_B(k)) = m$, then we get an $O(m + n^2 \log n)$ algorithm. If $m = O(n^{2+\epsilon})$, in particular if each pair has at most n^ϵ witnesses, then *BMM, BMWP* both can be solved in $O(n^{2+\epsilon})$ time. This is certainly faster than $O(n^3/(\log n)^2))$ algorithm for such special classes of matrices.

On the other hand, if each pair has either zero or sufficiently large number of witnesses, then a witness for each pair can be found easily. Suppose the pair (i, j) has $\Omega(n^{1-\epsilon})$ witnesses. If we pick up an index value k between 1 and k repeatedly and randomly, and check if it is a witness for (i, j), then in $O(n^\epsilon(\log n))$ steps, we will encounter a witness for (i, j), with probability of at least $1 - n^{-5}$. Thus we can obtain witnesses for all such pairs in $O(n^{2+\epsilon}(\log n)^2)$ time, with probability of at least $1 - n^{-3}$. In fact it is enough to pick a set of $O(n^\epsilon(\log n))$ random values of k only once (as described in the algorithm given below). In particular, for $\epsilon = 0.5$, we can find in $O(n^{2.5}(\log n)^2)$ time, a witness for ech pair. This algorithm is deterministic if each pair has at most $n^{0.5}$ witnesses, or randomized (with failure probability at most n^{-3}) if each pair has either zero or at least $n^{0.5}$ witnesses.

Not knowing whether each pair has at most n^ϵ ($\epsilon \leq 0.5$) or at least $n^{1-\epsilon}$ ($\epsilon \leq 0.5$) witnesses is not a handicap. For the former case, we merely need to check if m (the total number of witnesses) is at most $n^{2+\epsilon}$. For the latter case, let l be the minimum non-zero number of witnesses that any pair has. Then it is enough to pick an index k between 1 and n repeatedly and randomly at most $5(n/l)(\log n)$ times, to ensure that we pick a witness for every pair (with at least one witness), with probability $\geq 1 - n^{-3}$. We still do not know the value of l. However as shown below, we only need to know an approximate value of l. Note that since $l \geq n^{1-\epsilon}$ with $\epsilon \leq 0.5$, w have $l \geq n^{0.5}$.

Suppose we divide the interval $[0, n]$ into $[0], [1, 2], (2, 4], (4, 8], \ldots, (2^j, n]$ for some j. Clearly, there are only $\log n$ intervals and l lies in one of the intervals. For any interval indexed by s, for any $l' \in (2^s, 2^{s+1}]$, the maximum and the minimum values of n/l' differ by a factor of at most 2. Moreover, as l' goes from n down to $n^{0.5}$, n/l' grows from 1 (when $n = l'$) monotonically. Thus, starting with the last interval and going down over successive intervals, we use the lower bound 2^s as an approximate value of l and pick up the required number of random values of k. When we come to the correct interval, we will find witnesses for all pairs with probability of at least $1 - n^{-4}$. If $l \in (2^s, 2^{s+1}]$, the total time is $O\left((\Sigma_{s \leq r \leq j} n/2^r) n^2 (\log n))^2\right)$. But $\Sigma_{s \leq r \leq j} n/2^r \leq 2n/2^s = O(n/l))$. Thus the total running time is $O(n^{2+\epsilon}(\log n)^2)$. and remains the same as when we know apriori the exact value of l.

The algorithm explained so far has formally been described below as algorithm **Bool-Matrix-Witnesses**. Based on the arguments given above we can assert the following: **(i)** If $m = O(n^{2+\epsilon})$ for some $\epsilon \leq 0.5$, then C and W can be computed deterministically in $O(n^2 + m)$ time. **(ii)** If every pair has either zero or at least $n^{1-\epsilon}$ (or more strongly $|N|/n^\epsilon$, N defined in Step 3) witnesses for some $\epsilon \leq 0.5$, then C and W can be computed deterministically in

$O(n^{2+\epsilon}(\log n)^2)$ time with probability at lest $1-n^{-3}$. **(iii)** The algorithm always runs in $O(n^{2.5}(\log n)^2)$ time.

Remark 1. While assertion **(ii)** can be realized by using direct method also, it does not seem possible to realize assertion **(i)** by the direct method.

Algorithm 21 Bool-Matrix-Witnesses
(**Input**: Boolean Matrices A and B of size $n \times n$ each)
(**Output**: Product matrix C and witness matrix W.)

1. **for** each $i, j \in \{1, 2, \ldots, n\}$, set $c_{i,j} = w_{i,j} = 0$.
2. **for** $k = 1$ to n **do**
   (a) Store the 1s in A_{*k} in a search tree A_k and store the 1s in B_{k*} in a search tree B_k.
   (b) Compute $n_A(k) = |A_k|$, and $n_B(k) = |B_k|$.
3. **if** $\Sigma_k n_A(k) n_B(k) = O(n^{2.5}(\log n))$ **then**
   (a) **for** $k = 1$ to n **do**
      i. for all pairs (i, j), where $i \in A_k, j \in B_k$, set $c_{i,j} = 1$ and $w_{i,j} = k$.
   (b) **Exit**.
4. Compute $N = \{k \mid n_A(k) \neq 0, n_B(k) \neq 0\}$, $r_u = \lfloor \log_2 n \rfloor$, $r_l = \lfloor \log_2 n^{0.5} \rfloor$ and set $r = r_u$.
5. **repeat**
   (a) Randomly (with replacement) draw $s = 5(n/2^r)(\log n)$ integers $k_1, k_2, \ldots, k_s$ from N.
   (b) **for** each $k \in \{k_1, k_2, \ldots, k_s\}$ **do** Execute step 3(a)(i).
   (c) Set $r = r - 1$.
   until either $r < r_l$ or there are at most $n^2/2^r$ pairs with no witnesses so far.
6. if there are at most $n^{1.5}$ pairs with no witnesses, find the witnesses for these pairs by a direct method.

3 An $O(n^2(\log n))$ Algorithm for Random Boolean Matrices

In this section, we show that if both input matrices are random instances, we can obtain, using our approach, almost optimal $O(n^2(\log n))$ combinatorial algorithms for finding witnesses for all pairs, with very high probability. We consider the following random model. A and B are both random boolean matrices. Each entry in both of them is set (independently) to 1 with probability $p = p(n)$ and 0 otherwise. Our results hold for *all values* of $p(n)$. For this section, we assume that the sets A_k, B_k of non-zero entries in A_{*k}, B_{k*} respectively are stored in doubly-linked lists. We initially set the product matrix $c_{i,j}$ and the witness matrix $w_{i,j}$ both to be zero for all i and j. This takes $O(n^2)$ time. We will often be using the CH bounds [17, 13, 14] given below.

Chernoff-Hoeffding (CH) bounds [17] : Let $X_1, \ldots, X_m$ be *independent* random variables that take on values in $[0, 1]$, with $E[X_i] = p_i$, $1 \leq i \leq m$. Let $X \doteq \sum_i X_i$ and $\mu \doteq E(X) = \sum_i p_i$. Then,

$$\mathbf{Pr}(X \leq \mu(1-\delta)) \leq e^{-\mu\delta^2/2} \quad \text{for any } \delta \in [0,1]$$
$$\mathbf{Pr}(X \geq \mu(1+\delta)) \leq e^{-\mu\delta^2/3} \quad \text{for any } \delta \in [0,1]$$
$$\mathbf{Pr}(X \geq \mu(1+\delta)) \leq e^{-\mu(1+\delta)(\log_e(1+\delta))/4} \quad \text{for any } \delta > 1$$

Case $p(n) \leq 10(\log n)/n$ **:**

By CH bounds, for any fixed k, the number of non-zero entries in A_{*k} is $O(\log n)$ with probability[3] at least $1 - n^{-4}$. Similar statement holds for the matrix B also. This implies that, with probability at least $1 - O(n^{-3})$, we have

$$n_A(k),\ n_B(k) = O(\log n),\ \text{for all}\ \ k$$

Hence

$$\sum_k (n_A(k) \cdot n_B(k)) = O(n(\log n)^2)$$

Thus we can find witnesses (if there is any) for all pairs in $O(n^2 + n(\log n)^2) = O(n^2)$ time.

Case $10(\log n)/n \leq p(n) \leq 10\sqrt{(\log n)/n}$ **:**

Again, by Chernoff-Hoeffding, for any fixed k, and any positive constant $\delta \geq 1$, the number of non-zero entries in A_{*k} is at most $10(1+\delta)\sqrt{n(\log n)}$ with probability at least $1 - e^{-10(1+\delta)np/4} \geq 1 - O(n^{-4})$. This implies that, with probability at least $1 - O(n^{-3})$,

$$n_A(k),\ n_B(k) \leq 10(1+\delta)\sqrt{n(\log n)},\ \text{for all}\ \ k$$

Hence

$$\sum_k (n_A(k) \cdot n_B(k)) = O(n^2(\log n))$$

and witnesses for all pairs can be found in $O(n^2(\log n))$ time.

Case $10\sqrt{(\log n)/n} \leq p(n) \leq 1$ **:**

First, we compute an approximation q to p as follows : $q = NZ/n^2$ where NZ is the number of non-zero entries in A. Since each entry is set to 1 or 0 with independent probabilities, by CH bounds, we have for any $0 < \delta < 1$,

$$(1-\delta)n^2p \leq\ NZ\ \leq (1+\delta)n^2p$$

with probability at least $1 - 2e^{-n^2p\delta^2/3}$. As long as $p \geq n^{-2+\epsilon}$ and $\delta = (\log n)^{-M}$ for fixed positive ϵ, M, we have $1 - 2e^{-n^2p\delta^2/3} \geq 1 - n^{-5}$, say. This means that $q = p[1 \pm o(1)]$ with polynomially low failure probability.

Now fix a value of i and fix $\delta = 0.8$. As before, we assume that the sets B_{k*} have been found out and stored in doubly linked lists B_k. Let $n_0 = \lceil 25(\log n)/q^2 \rceil$. Due to the lower bound on p in this case, $n_0 \leq n/4$. Consider the

[3] Actually, when $p(n) = o((\log n)/n)$, we can not directly deduce this failure probability from CH bounds because the expected number of 1s in A_{*k} becomes $o(\log n)$. However, we can overcome this problem by choosing a larger value for p and applying CH bounds. This is justified since we are only trying to upper bound the number of 1s.

first n_0 columns of the i-th row, the entries $a_{i,1}, \ldots, a_{i,n_0}$. By CH bounds, with probability at least $1 - 2e^{-\delta^2 n_0 p/3} \geq 1 - O(n^{-5})$, there are at least $(1-\delta)n_0 p$ and at most $(1+\delta)n_0 p$ columns among the first n_0 columns such that there is a 1-entry in those columns in the i-th row.

Let $k_1, \ldots, k_s$ be the corresponding column indices, where $(1-\delta)n_0 p \leq s \leq (1+\delta)n_0 p$, with very high probability. Now consider the rows of B with corresponding indices $k_1, \ldots, k_s$. There are totally ns possible 0,1-entries. Of these, there will be at least $(1-\delta)nsp$ and at most $(1+\delta)nsp$ non-zero entries, with probability $1 - O(n^{-5})$. Substituting the values of n_0 and bounds on q, s, we deduce that there will be at least $(1+\delta^2-2\delta)n(\log n)$ and at most $(1+\delta^2+2\delta)n(\log n)$ non-zero entries in the rows $B_{k_1,*}, \ldots, B_{k_s,*}$.

Also, for any column j,

$$\mathbf{Pr}(b_{k_l,j} = 0,\ \forall\, l = 1, \ldots, s) = (1-p)^s \leq (1-p)^{(1-\delta)n_0 p} = O(n^{-5})$$

Hence, with probability $\geq 1 - O(n^{-4})$, for every j, the pair (i,j) has at least one witness in $\{k_1, \ldots, k_s\}$ and the total number of such witnesses is $O(n(\log n))$.

The indices $k_1, \ldots, k_s$ can be found in $O(n)$ time. The sets $B_{k_1}, \ldots, B_{k_s}$ represent the set of all witnesses within $\{k_1, \ldots, k_s\}$ for each column. Thus the total running time for finding witnesses for all pairs corresponding to the i-th row is $O(n(\log n))$. Thus witnesses for all pairs can be found in $O(n^2(\log n))$ time, with probability at least $1 - O(n^{-3})$.

Remark 2. If we apply the straightforward $O(n^3)$ algorithm upon the failure of our $O(n^2(\log n))$ algorithm, we get an $O(n^2(\log n))$ expected time algorithm for *BMM, BMWP* problems. The above given analysis works when $B = A$ also. Hence computing A^2 and its witnesses can be done in $O(n^2(\log n))$ time, for a random A.

4 An $O(n^2(\log n))$ Algorithm for Random A and Fixed Arbitrary B

When only A is assumed to be a random matrix and B is arbitrary but *fixed*, we can find witnesses in $O(n^2(\log n))$ time with probability at least $1 - n^{-3}$. Let $L = L(n)$ be the positive integer such that $2^L \leq n/10(\log n) < 2^{L+1}$. Let $m = 10(\log n)/n$ and divide the interval $[0,1]$ into $[0,m], [m,2m], \ldots, [2^j m, 2^{j+1}m], \ldots, [2^L m, 1]$. As before, we assume that each entry in A is chosen with probability $p(n)$. Initially, we assume that the algorithm knows the value of p. Also, we maintain the non-zero entries of each column of B in a doubly-linked list. As before, we initially set $c_{i,j} = w_{i,j} = 0$ for all i, j.

Case $p(n) \leq 10(\log n)/n$ **:**

As before, for any fixed k, we have $n_A(k) = O(\log n)$ with probability at least $1 - n^{-4}$. Hence $\sum_k n_A(k) \cdot n_B(k) = O(n^2(\log n))$. Thus, witnesses for all pairs can be found in $O(n^2(\log n))$ time with probability at least $1 - n^{-3}$.

Case $2^l \cdot 10(\log n)/n \leq p(n) \leq 2^{l+1} \cdot 10(\log n)/n$ **for some** l **:**

For any j such that the jth column of B, B_{*j}, has at most $n/2^l$ 1s, we certainly can find witnesses for all pairs (i,j) $(1 \leq i \leq n)$ by scanning all *non-zero* entries of the k-th column of A, for *each* k such that $b_{k,j} = 1$. The number of such non-zero entries is $O(n^2 p/2^l) = O(n(\log n))$ with probability $\geq 1 - n^{-5}$. This takes $O(n(\log n))$ time for each such j.

If j is such that B_{*j} has at least $n/2^l$ 1s, then consider the first $n/2^l$ row positions $k_1, \ldots, k_s$ where $s = n/2^l$, corresponding to non-zero entries in B_{*j}. For any fixed i,

$$\mathbf{Pr}(a_{i,k_r} = 0,\ \forall r = 1, \ldots, s) = (1-p)^{n/2^l} = O(n^{-5})$$

Hence every pair (i,j) has a witness in $\{k_1, \ldots, k_s\}$ and these witnesses can be found by scanning the non-zero entries of $A_{*,k_1}, \ldots, A_{*,k_s}$. But there are only $O(n^2 p/2^l) = O(n(\log n))$ non-zero entries in these columns, with probability $\geq 1 - n^{-5}$. This takes $O(n(\log n))$ time for each such j. Hence witnesses for all pairs can be found in $O(n^2(\log n))$ time with probability at least $1 - n^{-3}$.

Case $2^L \cdot 10(\log n)/n \leq p(n) \leq 1$ **:**

As before, for any j, consider all 1s in B_{*j} if there are only at most $n/2^L$ 1s, otherwise consider the first $n/2^L$ 1s in B_{*j}. Let $k_1, \ldots, k_s$ be the row positions corresponding to these 1s. We have $s \leq n/2^L$ in the former case and $s = n/2^L$ in the latter case and $n/2^L \leq 20(\log n)$. Let us consider the non-zero entries in the columns $A_{*k_1}, \ldots, A_{*k_s}$. In the former case, these 1s give witnesses for all pairs (which have a witness) (i,j) and it takes $O(n(\log n))$ time for each such j.

In the latter case, consider any fixed i. Then,

$$\mathbf{Pr}(a_{i,k_r} = 0,\ \forall r = 1, \ldots, s) = (1-p)^{n/2^l} = O(n^{-5})$$

Hence every pair (i,j) has a witness in $\{k_1, \ldots, k_s\}$ and these witnesses can be found by scanning the non-zero entries of $A_{*,k_1}, \ldots, A_{*,k_s}$. But there can be only $O(n(\log n))$ non-zero entries in these columns. This takes $O(n(\log n))$ time for each such j. Hence witnesses for all pairs can be found in $O(n^2(\log n))$ time with probability at least $1 - n^{-3}$.

Remark 3. The working of the algorithm depends on the value of p. Our analysis assumes that the algorithm knows the values of p. However, as shown in the previous section, we can get an estimate q such that $q = p[1 \pm o(1)]$ with very high probability, as long as $p \geq n^{-2+\epsilon}$ for any positive constant ϵ. Once we have q, we can find the range into which it falls and accordingly the algorithm proceeds. Since q is not the same as p, it may be that q falls into an interval other than the one to which p belongs. But since $q = p[1 \pm o(1)]$, it should only fall into an adjacent interval. So, we modify the algorithm to run for these adjacent intervals also. But this only increases the running time by a constant factor. Thus algorithm runs in $O(n^2(\log n))$ time even when we use q in place of p. When $p \leq n^{-2+\epsilon}$, the number of 1s in A is $O(n^\epsilon)$ with probability $\geq 1 - O(n^{-5})$. But this means that the total number of witnesses for all pairs is only $O(n^{1+\epsilon})$.

Remark 4. It should be noted that the algorithm is the same for all B and the analysis is also the same. However, the analysis holds only when B is assumed to be fixed. The

same result holds when A is arbitrary but fixed and B is a random boolean matrix. This is because the matrix multiplication definition is symmetric with respect to A and B. Computing $A \cdot B$ is equivalent to computing $B^T \cdot A^T = (A \cdot B)^T$. Hence when B is random and A is arbitrary, $B^T \cdot A^T$ can be computed using the algorithm described in this section.

5 Computing the Diameter

Based on the results developed so far, we show how to estimate the diameter of random graphs. For an undirected graph $G = (V, E)$, for $u, v \in V$, we define $d(u, v)$ to be the length of any shortest path between u and v if any exists. Otherwise, define $d(u, v)$ to be ∞. For directed graphs, we consider the directed paths for distance calculation. The diameter of G, $diam(G)$, is defined as the maximum distance between any pair of vertices, $diam(G) = max\{d(u, v) \mid u, v \in V\}$.

If we adopt the algebraic approach, we can find the diameter in $\tilde{O}(n^\omega)$ time using the shortest path algorithms developed in [2, 16]. As pointed out in [1], this seems to be the only known approach for estimating the diameter. Chung [8] had earlier asked whether there exists an $O(n^{3-\epsilon})$ algorithm for diameter computation which does not use fast matrix multiplication. Basch, et. al. [6] and Aingworth, et. al. [1] present combinatorial algorithms. The algorithm of [6] requires $O(n^3/(\log n)^2)$ time to verify if an arbitrary graph has diameter d for any fixed $d \geq 2$. [1] presents an $O(n^{2.5}\sqrt{\log n})$ algorithm for diameter computation with an additive error ≤ 2. The running times are with respect to worst-case measure. On the other hand, results of this section show that there is an $O(n^2(\log n))$ algorithm for computing the diameter exactly if the input is a random graph.

First consider the undirected random graph $G = (V, E)$ drawn from the model $\mathcal{G}(n, p)$. Here each of $\binom{n}{2}$ edges is chosen independently with probability $p = p(n)$. When $p(n) \leq 30(\log n)/n$, by CH-bounds, there are only $O(n(\log n))$ edges in the random graph, with probability at least $1 - O(n^{-3})$. In this case, we can apply the straightforward approach (breadth-first-search starting at every vertex) to find the diameter in $O(n^2(\log n))$ time. Hence we assume that $p(n) \geq 30(\log n)/n$. First, we prove (in Section 6) that with probability at least $1 - O(n^{-3})$, there is a d-path between any pair of vertices in G, where $d = (\log n)/(\log\log n)$ and hence $diam(G) \leq d$. In what follows, we show how to find the exact value of $diam(G)$ in $O(n^2(\log n))$ time.

Notations: For a pair (i, j), a *walk* of length $l \geq 0$ between i and j is a sequence of alternating vertices and edges such that it starts at i, ends at j, and there are l edges in the sequence. A path is a walk in which no edge or vertex appears more than once. For any pair (i, j), there exists a walk of length of length $\leq l$, iff there exists a path of length $\leq l$. For a vertex i, and an integer $l \geq 0$, let $N_l(i)$ (and $n_l(i)$) denote the set (and number) of all vertices j such that $d(i, j) \leq l$. The adjacency matrix A of G defined by $a_{i,j} = 1$, if either $i = j$ or $(i, j) \in E$. This is slightly different from the usual definition in which $a_{i,i} = 0$. The following facts are easy to verify.

Fact 51 **(i)** *A is a random boolean matrix.*[4] **(ii)** *A represents all pairs which are at a distance at most 1.* **(iii)** *In A^2 (boolean multiplication), any pair i, j has a 1-entry if and only if there is a walk (and hence a path) of length ≤ 2 between i and j in G. In general, A^l has a 1-entry in (i,j)-th position if and only if there is a path of length $\leq l$ between i and j.* **(iv)** *$d(i,j)$ is the least value of l such that A^l has a 1 in its (i,j)-th entry.*

Case $p(n) \geq 10\sqrt{\log n/n}$ **:**

By CH bounds, with probability $\geq 1-n^{-3}$, any two vertices share at least one common neighbor and hence the diameter is at most 2. Thus A^2 has a 1-entry everywhere. We compute A^2 and its witnesses by setting $B = A$ and applying the algorithm outlined in Section 3. By Remark 3.1, this algorithm runs in $O(n^2(\log n))$ time and succeeds with required probability. Now, by comparing A and A^2, we can compute the distances and shortest paths for every pair of vertices. We should note that even if diameter is at most 2, it is not easy to verify it quickly in the worst-case.

Case $30(\log n)/n \leq p(n) \leq 10\sqrt{\log n/n}$ **:**

Initially, we set $B = A$ (first iteration). Then we update $B = AB$ for at most d iterations. Each time we compute a product, we also compute its witnesses. We stop the procedure the first time we notice that all entries of B are 1. If this happens after the l-th iteration, then diameter of G is l. Since diameter is at most d, we need at most d iterations. To analyze these multiplications, we can neither directly use the algorithm and its analysis given in Section 3 (since after the second iteration, B ceases to be truly random) nor directly use the analysis of Section 4 (since B is not fixed). However, by carefully estimating the number of 1s in successive B's and using this to decide when the multiplication algorithm should stop, we can achieve all multiplications (without loosing correctness) in a total time of $O(n^2(\log n))$. The details are given below.

For a pair i, j such that $b_{i,j}$ becomes 1 for the *first* time in the l-th iteration, we record the product witness (for this pair in this iteration) as the witness for a shortest path between i and j. We also record that $d(i,j) = l$. Thus, we can not only correctly compute the diameter, but also compute the shortest paths (through witnesses) between all pairs in G. The following lemma helps us in tightly bounding the running time. Its proof is provided in the full version of the paper.

Lemma 1. *With probability $\geq 1 - n^{-4}$, for some $L(1 \leq L \leq d-3)$ and some $s = 0, 1, 2$, the following hold: For all $i \in V$,* **(i)** *$n_{L-1}(i) \leq 1/(p \log n) \leq n_L(i)$ and $n_{L-1}(i) = o(n_L(i))$;* **(ii)** *$n_{L+s-1}(i) \leq 5(\log n)/p \leq n_{L+s}(i)$ and $n_{L+s-1}(i) = o(n_{L+s}(i))$;* **(iii)** *every vertex not in $N_{L+s}(i)$ will be adjacent to at least one vertex of the first $4(\log n)/p$ vertices of $N_{L+s}(i) - N_{L+s-1}(i)$.*

By Fact 5.1, after the l-th iteration, the set of 1s in $B_{*,i}$ and $B_{i,*}$ is precisely $N_l(i)$. So we perform $B = AB$ till $n_l(i) \geq 5(\log n)/p$ for all i. This will happen exactly

[4] The diagonal entries of A are not random, but our algorithms for random *BMM*, *BMWP* work even if the diagonal entries are not random.

after the $(L+s)$-th iteration by Lemma 5.1. It means *till* $(L+s)$-th iteration, we have $|B_i| \leq 5(\log n)/p$. Similarly, *till* L-th iteration, we have $|B_i| \leq 1/p(\log n)$. Till L-th iteration, we compute $B = AB$ by scanning (for each j) the $O(np)$ 1s in $A_{*,k}$ for each k such that $b_{k,j} = 1$ and there are at most $1/p(\log n)$ such k's. The total running time for iterations numbered $1, \ldots, L$ is $O(n^2 L + nL \cdot np/p(\log n)) = O(n^2(\log n))$. For iterations, $L+1, \ldots, L+s$, we multiply in the same way taking $O(n^2 s + ns \cdot np(\log n)/p)$. Since $s \leq 2$, we get this time to be $O(n^2(\log n))$. For the $(L+s+1)$-th iteration, we consider only the first $5(\log n)/p$ non-zero entries in each column of B, for computing AB. By Lemma 5.1 **(iii)**, this will be the last iteration taking $O(n^2(\log n))$ time. So the total time taken is $O(n^2(\log n))$ and success probability is $1 - O(n^{-3})$.

The proof of diameter (given in Section 6) extends to random digraphs also. Hence the algorithm outlined in this section can be used to compute the diameter of random digraphs also. We formally state the algorithmic results obtained in Sections 3,4 and 5 below.

Theorem 1. *Let A and B be two boolean matrices. Let G be a random graph/digraph in which each possible edge/arc is chosen with probability p. The following hold.*

1. *If A and B are both random, then their product and its witnesses can be computed in $O(n^2(\log n))$ time with probability at least $1 - O(n^{-3})$.*
2. *If one of A and B is random and the other is arbitrary but fixed, then their product and its witnesses can be computed in $O(n^2(\log n))$ time with probability at least $1 - O(n^{-3})$.*
3. *The diameter of G, distances and shortest paths between all pairs can be computed in $O(n^2(\log n))$ time. The failure probability is $O(n^{-3})$.*

6 $diam(G(n,p)) \leq (\log n)/(\log\log n),\ p \geq 30(\log n)/n$

Diameter of random graphs have been studied before [7,12]. But these results only state that for certain ranges of $p(n)$, the diameter is sharply concentrated in a few values. But, what we need is a bound on the diameter for a broader range of $p(n)$ so that we can bound the running time. Also these results do not seem to have the required rate of convergence of the failure probability. But we need a guaranteed failure probability of at most n^{-1} so that we can get faster (on the *average*) algorithms. Hassin and Zemel [11] mention a bound of $O(\log n)$ on the diameter. We prove a stronger bound of $\log n/\log\log n$ in the following theorem. The proof uses Janson inequalities. Note that the theorem is actually a stronger statement than specifying an upper bound on the diameter.

Theorem 2. *Let $G(n,p)$ be a random graph/digraph on n vertices obtained by choosing each edge/arc with probability $p \geq 30(\log n)/n$. Then, with probability at least $1 - O(n^{-3})$, there is a d-path $(d = (\log n)/(\log\log n))$ in G between* every *pair of vertices.*

Proof. We prove only for random graphs. For random digraphs, the arguments are similar. Write $p = w/n$ for simplifying the calculations. We have $w \geq 30(\log n)$.[5] Fix two vertices u and v in V. For $d = (\log n)/(\log\log n)$, let $P_1, \ldots, P_m$, $m = (n-2)\ldots(n-d)$, be the collection of all possible d-paths (paths of length d) between u and v in G. For each i, let B_i be the event that P_i is present in G. Also, let $n_d(u,v)$ denote the number of d-paths between u and v in G. We have

$$\mu \doteq E(n_d(u,v)) = (n-2)\ldots(n-d)(w/d)^n \;=\; (w^d/n)[1+o(1)]$$

Since $w \geq 30(\log n)$, we have $\mu = \omega(1)$.

We use **Generalized-Janson-Inequality(GJI)** [4, Chapter 8] to prove that $\mathbf{Pr}$(there is no d-path between u and v) $= O(n^{-5})$. This proves the required bound on the diameter. The inequality is used as follows. For any $i \neq j$, $1 \leq i, j \leq m$, we write iSj if the paths P_i and P_j share at least one edge in common. For any i, define $\Delta_i = \sum_{j:jSi} \mathbf{Pr}(B_i \wedge B_j)$. Define $\Delta = \sum_{1\leq i\leq m} \Delta_i$. Let $\epsilon = \mathbf{Pr}(B_i) = (w/n)^d = o(1)$. **GJI** states that if $\Delta \geq \mu(1-\epsilon)$, then $\mathbf{Pr}(B_i$ does not hold for all $i) \leq e^{-\mu^2(1-\epsilon)/2\Delta}$. We can verify that $\Delta \geq \mu$.

Thus if we can prove that $\Delta \leq 3\mu^2/w$, it implies that $\mathbf{Pr}(\wedge_i \overline{B_i}) \leq e^{-5(\log n)} = O(n^{-5})$.

Fix an i and let P_i be the path $\langle u, u_1, \ldots, u_{d-1}, v\rangle$. Consider any j such that iSj. Then P_j shares at least one vertex (other than u and v) and at least one edge in common with P_j. Let $v(i,j), e(i,j)$ denote respectively the number of common vertices (excluding u and v) and the number of common edges shared by P_i and P_j.

Let $J_i = \{j : jSi\}$. Partition $J_i = J_i^0 \cup J_i^1$ where $J_i^0 = \{j \in J_i : v(i,j) = e(i,j)\}$ and $J_i^1 = \{j \in J_i : v(i,j) \geq e(i,j)+1\}$. Clearly, we can not have $v(i,j) < e(i,j)$. Further partition

$$J_i^0 = \bigcup_{1\leq l_e\leq d-1} J_i^0(l_e) \quad \text{where } J_i^0(l_e) = \{j \in J_i^0 : e(i,j) = l_e\}.$$
$$J_i^1 = \bigcup_{1\leq l_e\leq d-1} J_i^1(l_e) \quad \text{where } J_i^1(l_e) = \{j \in J_i^1 : e(i,j) = l_e\}.$$

Note that $e(i,j)$ always lies between 1 and $d-1$.

Clearly, for any $j \in J_i$, we have $\mathbf{Pr}(B_i \wedge B_j) = (w/n)^{2d-e(i,j)}$. Hence for any l_e,

$$\sum_{j\in J_i^1(l_e)} \mathbf{Pr}(B_i \wedge B_j) = (w/n)^{2d-l_e}|J_i^1(l_e)| \tag{1}$$

Since $v(i,j) \geq e(i,j)+1$ for each $j \in J_i^1$, we have $|J_i^1(l_e)| \leq (n^{d-l_e-2})(d^{l_e})$. In this expression, d^{l_e} bounds the number of different subsets of P_i of size l_e.

[5] We also assume that $w \leq n(\log\log n)/(\log n)$. Otherwise, any two vertices in V share a common neighbor with probability $\geq 1 - O(n^{-3})$ and hence $diam(G) \leq 2$.

Having fixed such a subset, n^{d-l_e-2} bounds the number of js that can be in $J_i^1(l_e)$. Using this in (1), we get

$$\sum_{j \in J_i^1(l_e)} \mathbf{Pr}(B_i \wedge B_j) \leq (w/n)^{2d-l_e} \cdot n^{d-l_e-2} \cdot d^{l_e}$$
$$\leq (w/n)^d \mu/n, \text{ since } d \leq w \tag{2}$$

Hence

$$\sum_{l_e} \left(\sum_{j \in J_i^1(l_e)} \mathbf{Pr}(B_i \wedge B_j) \right) \leq (w/n)^d d\mu/n \leq (w/n)^d \mu/w \tag{3}$$

Now we turn our attention to J_i^0. For any $j \in J_i^0(l_e)$, note that $P_i \cap P_j$ should be the union of *first* l'_e edges and the *last* l''_e edges on P_i for some $l'_e + l''_e = l_e$. Otherwise, we can not have $v(i,j) = e(i,j)$. For a given l_e, there are only $l_e + 1$ such possibilities. Also, for a given subset of size l_e, there are at most n^{d-l_e-1} such js that can be in $J_i^0(l_e)$. Using these, we get for each $l_e \geq 1$,

$$\sum_{j \in J_i^0(l_e)} \mathbf{Pr}(B_i \wedge B_j) \leq (w/n)^{2d-l_e} \cdot n^{d-l_e-1} \cdot (l_e + 1)$$
$$\leq (w/n)^d \mu (l_e + 1)/w^{l_e}$$
$$\text{Hence} \quad \sum_{j \in J_i^0} \mathbf{Pr}(B_i \wedge B_j) \leq (w/n)^d \mu \left(\sum_{l_e} (l_e + 1)/w^{l_e} \right) \tag{4}$$

Now using the fact $l_e \leq d - 1 = o(w)$, we deduce that $\sum_{l_e} (l_e + 1)/w^{l_e} \leq 2/w$ ignoring the $[1 + o(1)]$ factor. Combining (3) and (4), we get $\Delta_i \leq (w/n)^d 3\mu/w$. Hence $\Delta = \sum_i \Delta_i \leq 3\mu^2/w$.

7 Conclusions

In this paper, we introduced an alternative approach to boolean matrix multiplication and showed that it leads to almost optimal (on the average) combinatorial algorithms for solving *BMM, BMWP* problems, computing the diameter, all-pairs-shortest-paths in a graph/digraph. The constant 30 stated in Theorem 6.1 can be decreased to 18 with a corresponding increase in failure probability to $O(n^{-1})$. It may be possible to bring it below 18. Some interesting questions these results lead to are given below :

1. Does there exist an implementation of the A-columns-B-rows approach which finds witnesses in $o(n^\omega)$ time (randomized or deterministic) in the worst-case ? Or more generally, is it possible to maintain a collection of subsets of a universe so that we can compute their intersections efficiently.
2. In Section 4, can the requirement that B is fixed be removed ? In other words, can one design a fast algorithm for witnesses when A is random and B is arbitrary.

3. Can we remove the $\log n$ factor from the algorithms of Sections 3 and 4 ?
4. Can all-pairs-shortest-paths problem be solved in $O(n^{3-\epsilon})$ time without using matrix multiplication ?

Acknowledgement : The second author thanks Volker Priebe for bringing to his attention the work of the first author and also the work of [11].

References

1. D. Aingworth, C. Chekuri, P. Indyk and R. Motwani, "Fast Estimation of Diameter and Shortest Paths (without Matrix Multiplication)", Proceedings of the *Seventh Annual ACM Symposium on Discrete Algorithms*, pp. 547-554, 1996.
2. N. Alon, Z. Galil, O. Margalit and M. Naor, "Witnesses for Boolean Matrix Multiplication and Shortest Paths", Proceedings of the *33rd IEEE Symposium on Foundations of Computer Science*, pp.417-426, October 1992.
3. N. Alon and M. Naor, "Derandomization, Witnesses for Boolean Matrix Multiplication and Construction of Perfect hash functions", *Algorithmica*, 16:434-449, 1996.
4. N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley & Sons, 1992.
5. V.L. Arlazarov, E.A. Dinic, M.A. Kronrod and L.A. Faradzev, "On economical construction of the transitive closure of a directed graph", *Doklady Acad. Nauk SSSR*, 194:487-488, 1970 (in Russian).
6. J. Basch, S. Khanna and R. Motwani, "On Diameter Verification and Boolean Matrix Multiplication", Technical Report, Stanford University CS department, 1995.
7. B. Bollobas, *Random Graphs*, Academic Press (London), 1985.
8. F.R.K. Chung, "Diameters of Graphs: Old Problems and New Results", *Congressus Numerantium*, 60:295-317, 1987.
9. D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions", *Journal of Symbolic Computation*, 9(3):251-280, 1990.
10. Z. Galil and O. Margalit, "Witnesses for Boolean Matrix Multiplication and Shortest Paths", *Journal of Complexity*, pp. 417-426, 1993.
11. R. Hassin and E. Zemel, "On Shortest Paths in Graphs with Random Weights", *Mathematics of Operations Research*, Vol.10, No.4, 1985, 557-564.
12. M. Karonski, "Random Graphs", Chapter 6, *Handbook of Combinatorics*, Vol. I, ed. Graham, Grötschel, Lovászi, North-Holland, pp.351–380, 1995.
13. C.J.H. McDiarmid, "On the method of bounded differences", *Surveys in Combinatorics*, Edited by J. Siemons, London Mathematical Society Lecture Notes Series 141, pp.148-188, 1989.
14. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
15. C.P. Schnorr, "Computation of the Boolean Matrix-Vector AND/OR-Product in Average Time $O(m + n(\log n))$", Informatik-Festschrift zum 60. Geburtstag von Günter Hotz, (eds. Buchmann/Ganzinger/Paul), Teubner-Texte zur Informatik Band 1, 1992, pp.359-362.
16. R. Seidel, "On the All-Pairs-Shortest-Path Problem", Proceedings of the *24th ACM Symposium on Theory of Computing*, 745-749, 1992.
17. A. Srinivasan, "Scheduling and load-balancing via randomization", Proceedings of the *FST&TCS'97 Pre-conference Workshop on Randomized Algorithms*, Kharagpur, India, December, 1997.

Randomized Lower Bounds for Online Path Coloring *

Stefano Leonardi[1]** **Andrea Vitaletti**[1]

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza",
Via Salaria 113, 00198 Roma, Italy.
Email: {leon, vitale}@dis.uniroma1.it

Abstract. We study the power of randomization in the design of online graph coloring algorithms. No specific network topology for which randomized online algorithms perform substantially better than deterministic algorithms is known until now. We present randomized lower bounds for online coloring of some well studied network topologies.

We show that no randomized algorithm for online coloring of interval graphs achieves a competitive ratio strictly better than the best known deterministic algorithm [KT81].

We also present a first lower bound on the competitive ratio of randomized algorithms for path coloring on tree networks, then answering an open question posed in [BEY98]. We prove an $\Omega(\log \Delta)$ lower bound for trees of diameter $\Delta = O(\log n)$ that compares with the known $O(\Delta)$-competitive deterministic algorithm for the problem, then still leaving open the question if randomization helps for this specific topology.

1 Introduction

In this paper we present randomized lower bounds for a class of online graph coloring problems. The input instance to an online graph coloring problem is a sequence $\sigma = \{v_1, ..., v_{|\sigma|}\}$ of vertices of a graph. The algorithm must color the vertices of the graph following the order of the sequence. When the color is assigned to vertex v_i, the algorithm can only see the graph induced by vertices $\{v_1, ..., v_i\}$. The goal of a graph coloring algorithm is to use as few colors as possible under the constraint that adjacent vertices receive different colors.

Online graph coloring problems have been studied by several authors [KT91,HS92,LST89,Vis90]. The study of online graph coloring has actually been started even before the notion of competitive analysis of online algorithms was

* This work is partly supported by EU ESPRIT Long Term Research Project ALCOM-IT under contract n 20244, and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative".

** This work was partly done while the author was visiting the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 232-247, 1998.

introduced [ST85]. Kierstead and Trotter [KT81] in 1981 considered the online coloring problem for interval graphs. Every vertex of an interval graph is associated with an interval of the line. Two vertices are adjacent if the two corresponding intervals are intersecting. Since interval graphs are perfect graphs [Gol80], they have chromatic number χ equal to the maximum clique size ω, i.e. the maximum number of intervals overlapping at the same point of the line.

In [KT81] it is a presented a deterministic online algorithm that colors an interval graph of chromatic number ω with $3\omega - 2$ colors. They also prove that the $3\omega - 2$ bound is tight: for every deterministic algorithm there exists an input sequence where the algorithm uses at least $3\omega - 2$ colors.

The online interval graph coloring problem has a natural extension to trees. Every vertex of the graph, called the *intersection graph*, is in this case associated with a path on a tree network. Two vertices are adjacent in the graph if the two corresponding paths are intersecting. This problem has recently received a growing attention due to its application to wavelength assignment in optical networks [RU94,BL97,GSR96].

Several authors show an $O(\Delta)$ competitive deterministic algorithm for the problem of coloring online paths on a tree network (see for instance [BL97,GSR96]), where Δ is the diameter of the graph. Bartal and Leonardi [BL97] also show an almost matching $\Omega(\Delta/\log\Delta)$ deterministic lower bound on a tree of diameter $\Delta = O(\log n)$, where n is the number of vertices of the graph.

In this paper we present the first randomized lower bounds on the competitive ratio of randomized algorithms for online interval graph coloring and online coloring of paths on tree networks.

Randomized algorithms for online problems [BDBK$^+$90] have often been proved to achieve competitive ratios that are strictly better than deterministic online algorithms. The competitive ratio of a randomized algorithm against an oblivious adversary is defined as the maximum over all the input sequences of the ratio between the expected online cost and the optimal offline cost. The input sequence for a given algorithm is generated by the oblivious adversary without knowledge of the random choices of the algorithm.

However, for no network topology it is known a randomized online coloring algorithm that achieves a substantially better competitive ratio that the best deterministic algorithm for the problem. The first result we present in the paper is also along this direction.

We present the first randomized lower bound, up to our best knowledge, for online coloring of interval graphs. We show that any randomized algorithm uses an expected number of colors equal to $3\omega - 2 - o(1/\omega)$ for an interval graph of maximum clique size equal to ω, thus proving that randomization does not basically improve upon the best deterministic algorithm of [KT81].

Our second result is the first randomized $\Omega(\log\Delta)$ lower bound for online coloring of paths on a tree network of diameter $\Delta = O(\log n)$, then answering an open question of Borodin and El-Yaniv [BEY98]. There is still a substantial gap

between the presented lower bound and the $O(\Delta)$ deterministic upper bound known for the problem [BL97,GSR96].

The current status of the online path coloring problem on trees can be compared with the known results for the dual problem of selecting online a maximum number of edge-disjoint paths on a tree network, i.e. a maximum independent set in the corresponding intersection graph. An $O(\log \Delta)$-competitive randomized algorithm is possible for the online edge-disjoint path problem on trees [AGLR94,LMSPR98], against an $\Omega(\Delta)$ deterministic lower bound obtained even on a line network of diameter $\Delta = n$ [AAP93]. Our result still leaves open the question if an $O(\log \Delta)$-competitive randomized algorithm is possible for the online path coloring problem on tree networks.

Irani [Ira90] studies the problem of coloring online inductive graphs. A graph is d-inductive if the vertices of the graph can be associated with numbers 1 through n in a way that each vertex is connected to at most d vertices with higher numbers. Irani shows that any d-inductive graph can be colored online with $O(d \log n)$ colors and presents a matching $\Omega(\log n)$ deterministic lower bound. The graph obtained from the intersection of paths on a tree network has been independently observed to be a $(2\omega - 1)$ inductive graph by [BL97] and by Kleinberg and Molloy as reported in [BEY98]. Our lower bound for online path coloring on trees then implies a first $\Omega(\log \log n)$ lower bound on the competitive ratio of randomized algorithms for online coloring of inductive graphs.

We conclude this section mentioning the previous work on randomized online coloring algorithms for general graphs. Vishwanathan [Vis90] gives an $O(n/\log n)$ competitive randomized algorithm, improving over the $O(n/\log^* n)$ deterministic bound of Lovász, Saks and Trotter [LST89]. Halldórson and Szegedy [HS92] give an $\Omega(n/\log^2 n)$ randomized lower bound for the problem. Bartal Fiat and Leonardi [BFL96] study the model in which a graph G is known in advance to the online algorithm. The sequence σ may contain only a subset of the vertices of G. The algorithm must color the subgraph of G induced by the vertices of σ. The authors show that even under this model an $\Omega(n^\epsilon)$ randomized lower bound, for a fixed $\epsilon > 0$, is possible.

The paper is structured as follows. Section 2 presents the lower bound on online coloring of interval graphs. Section 3 presents the lower bound for path coloring on tree networks. Conclusions and open problems are in Section 4.

2 A lower bound for online interval graph coloring

In this section we present a lower bound on the competitive ratio of randomized algorithms for online interval graph coloring.

The input instance to the online interval graph coloring problem is given by a sequence of intervals on a line graph. Every interval is denoted by two

endpoints of the line. The algorithm must color the intervals one by one, in the order in which they appear in the sequence. The goal is to use as few colors as possible under the constraint that any two overlapping intervals are assigned with different colors.

The competitive ratio of an online algorithm for the interval graph coloring problem is given by the maximum over all the input sequences of the ratio between the expected number of colors used by the algorithm and the chromatic number of the interval graph, i.e. the maximum number of intervals ω overlapping at the same point of the line.

A lower bound for randomized algorithms against an oblivious adversary is established using the application of Yao's Lemma [Yao77] to online algorithms [BEY98,BFL96]. A lower bound over the competitive ratio of randomized algorithms is obtained proving a lower bound on the competitive ratio of deterministic online algorithms for a specific probability distribution over the input sequences for the problem.

We first give some notation. We will denote by P_ω the specific probability distribution over input sequences with chromatic number ω we use to prove the lower bound. Probability distribution P_ω will be described by a set of input sequences with chromatic number ω, *with every input sequence presented with equal probability.*

We denote by $\sigma \in P$ the generic input sequence of probability distribution P. Input instance σ is formed by a sequence of intervals $\{I_1, ..., I_{|\sigma|}\}$.

Probability distributions P and Q are said *independent* if for any $I^1 \subset \sigma^1 \in P$, $I^2 \subset \sigma^2 \in Q$, I^1 and I^2 are disjoint intervals. The set of sequences of probability distribution $P \cup Q$ is obtained by the concatenation of every input sequence of P with every input sequence of Q.

2.1 The probability distribution

The probability distribution P_ω used for proving the lower bound is defined recursively. We will resort to a pictorial help to describe the sequence.

Probability distribution P_1 is formed by a single input sequence containing a single interval. P_ω is the union of λ *independent* probability distributions $\overline{P}^1_\omega \cup .... \cup \overline{P}^\lambda_\omega$, as described in Figure 1. The value of λ will be fixed later.

$\overline{P}^j_\omega$ is obtained from four independent distributions $P^1_{\omega-1}, P^2_{\omega-1}, P^3_{\omega-1}, P^4_{\omega-1}$. The set of input sequences of $\overline{P}^j_\omega$ is obtained by the concatenation of every input sequence of $P^1_{\omega-1} \cup P^2_{\omega-1} \cup P^3_{\omega-1} \cup P^4_{\omega-1}$ with every of 10 distinct subsequences $T_1, ..., T_{10}$, called *configurations,* of at most 4 intervals as described in Figure 2. The intervals of every of the 10 different configurations are numbered in Figure 2 following the order in which they appear in the sequence. Every probability distribution $P^i_{\omega-1}$ is generated applying the present definition for $\omega - 1$.

Observe that every input sequence of P_ω has chromatic number ω. This can easily be seen with an inductive argument. Probability distribution P_1 contains

a single sequence with chromatic number 1. By induction every input sequence from $P^i_{\omega-1}$, $i = 1, .., 4$, has chromatic number $\omega - 1$. Every input sequence $\sigma \in P^1_{\omega-1} \cup P^2_{\omega-1} \cup P^3_{\omega-1} \cup P^4_{\omega-1}$ has also chromatic number $\omega - 1$. One can check from Figure 2 that the concatenation of σ with every of the 10 configurations increases the chromatic number by 1. Since P_ω is the union of λ independent probability distributions $\overline{P}^1_\omega, ..., \overline{P}^\lambda_\omega$, every sequence of P_ω has chromatic number ω.

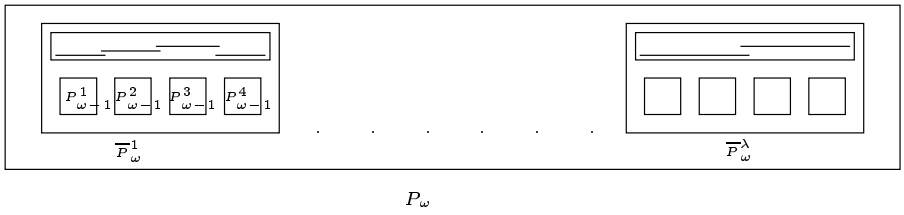


Figure 1. The definition of the probability distribution.

2.2 The proof of the lower bound

The proof of the lower bound is based on the following lemma:

Lemma 1. *Any deterministic online algorithm uses at least* $3\omega - 2$ *colors with probability at least* $1 - e^{-c}$ *on an input sequence drawn from probability distribution* P_ω*, if* $\lambda \geq \frac{10c}{(1-e^{-c})^4}$.

From the above lemma, choosing constant c large enough, say $c = \ln 3\omega^2$, we obtain the following Theorem:

Theorem 1. *For any randomized algorithm for online interval graph coloring, there exists a input sequence of chromatic number* ω *where the expected number of colors used by the algorithm is at least* $3\omega - 2 - o(1/\omega)$.

The remaining part of this section is then devoted to the proof of Lemma 1. The proof is by induction.

The claim of Lemma 1 holds for $\omega = 1$, since a deterministic algorithm uses one color for the sequence from probability distribution P_1 containing one single interval. Assume the claim holds for a probability distribution $P_{\omega-1}$, i.e. with probability at least $1 - e^{-c}$, the deterministic algorithm uses $3(\omega - 1) - 2$ colors for an input sequence drawn from a probability distribution $P_{\omega-1}$.

As an intermediate step of the proof we will prove a claim that holds for any probability distribution $\overline{P}^j_\omega$, $j = 1, .., \lambda$. We denote in the following by $\overline{P}_\omega$ the generic probability distribution $\overline{P}^j_\omega$.

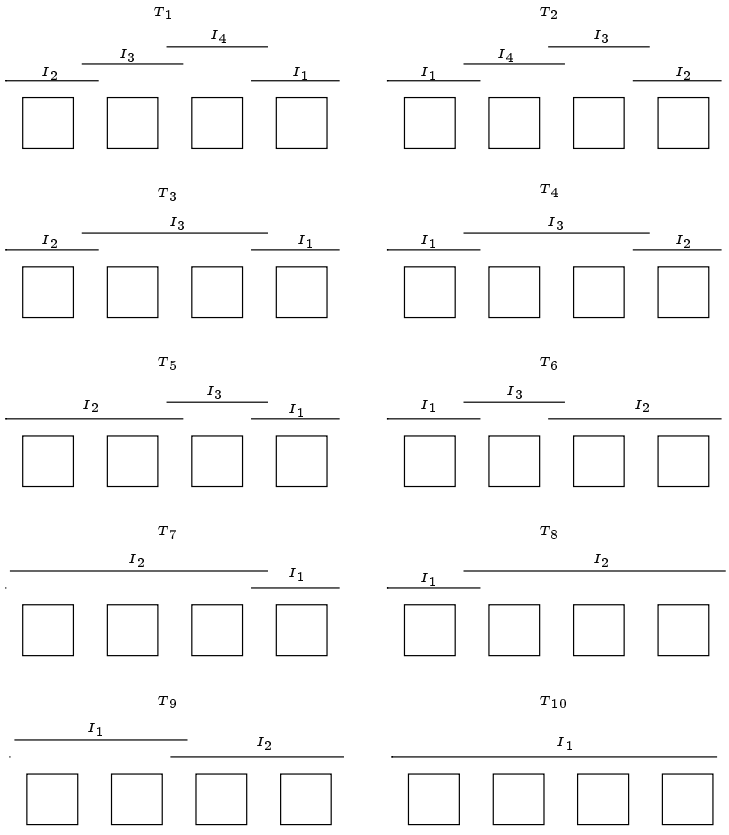


Figure 2. The 10 configurations used to form the probability distribution.

Lemma 2. *Consider a probability distribution* $\overline{P}_\omega$. *Assume the deterministic algorithm uses at least* $3(\omega-1)-2$ *colors for every input sequence* σ^i *drawn from probability distribution* $P^i_{\omega-1}$, $i = 1, ..., 4$. *With probability at least* $1/10$, *the deterministic algorithm uses at least* $3\omega-2$ *colors for an input sequence drawn from probability distribution* $\overline{P}_\omega$.

Proof. Denote by C^i the set of colors used for a generic input sequence σ^i drawn from probability distribution $P^i_{\omega-1}$, and $C_{\omega-1} = \cup_{i=1}^4 C^i$. Let $\overline{C}_\omega$ be the set of colors used for an input sequence from probability distribution $\overline{P}_\omega$. We will prove that $|\overline{C}_\omega| \geq 3\omega-2$ with probability at least $1/10$.

We distinguish four cases on the basis of value $c = |\cup_{i=1}^4 C^i| - (3(\omega-1)-2)$, the number of colors exceeding $3(\omega-1)-2$ used by the algorithm for the four sequences σ^i, $i = 1, ..., 4$. We will separately consider the cases of $c = 0, 1, 2$ and $c \geq 3$.

c=0. In this case the deterministic algorithm uses the same set of colors for every sequence σ^i, $i = 1, .., 4$. The new intervals presented in any of the 10 configurations must be assigned with colors not in $C_{\omega-1}$. With probability $2/5$, one of configurations T_1, T_2, T_3 or T_4 is presented. In all these configurations, one among intervals I_1 and I_2 that contains all the intervals of σ^4, the other all the intervals of σ^1. We furtherly distinguish two cases: *a.)* Intervals I^1 and I^2 have assigned the same color, say c_1; *b.)* Intervals I^1 and I^2 have assigned different colors, say c_1 and c_2.

a. With probability $1/2$, the sequence is completed by intervals I^3 and I^4 of configuration T_1 or T_2. Interval I^4 must be assigned with a color different from c_1, say c_2, since it is overlapping interval I^1. Interval I^3 must be assigned with a color different from c_1 and c_2, say c_3, since it is overlapping intervals I_4 and I_2. Then, with probability $1/5$, 3 more colors are used, and the claim is proved.

b. In this case, with probability $1/2$, the sequence is completed by interval I_3 of configuration T_3 or T_4. Interval I^3 is assigned with a color different from c_1 and c_2, say c_3, since it overlaps with both intervals I^1 and I^2. Also in this case, with probability $1/5$, 3 more colors are used, thus proving the claim.

c=1. We prove that with probability at least $1/10$, 2 colors not in $C_{\omega-1}$ are used by the deterministic algorithm. The difficulty of this case is given to the fact that a sequence σ^i may use only a subset of colors of $C_{\omega-1}$. A color of $C_{\omega-1}$ not used for an interval of a sequence σ^i may be "re-used" for an interval overlapping the intervals of sequence σ^i.

However, since any C^i contains at least $|C_{\omega-1}| - 1$ colors, we can make use of the following simple fact:

Claim. For any two sequences σ^i, σ^j, $i \neq j$, if $C^i \neq C^j$ then $C^i \cup C^j = C_{\omega-1}$.

The proof separately considers 4 different cases distinguished on the basis of the maximum cardinality s of a subset of $\{\sigma^1, \sigma^2, \sigma^3, \sigma^4\}$, formed by sequences assigned with same set of colors, that is every color is either assigned to an interval of all the sequences of the subset, or not assigned at all to an interval of any sequence of the subset. We have four different cases, for $s = 1, 2, 3, 4$. (Every subcase also includes its symmetric, obtained by replacing σ^1 with σ^4 and σ^2 with σ^3.)

s=1. In this case, for every two sequences have assigned a different set of colors. Then, we have $C^1 \cup C^2 = C^3 \cup C^4 = C_{\omega-1}$. With probability $1/10$ configuration T_9 is given. Interval I_1 is assigned with a color $c_1 \notin C_{\omega-1}$, since for any color of $C_{\omega-1}$, interval I_1 overlaps an interval

assigned with that color. For the same reason, a color $c_2 \notin C_{\omega-1}$ is assigned to I_2. Color c_2 must be distinct from c_1 since interval I_2 intersects interval I_1. The claim is then proved.

s=2. The case in which at most two sequences are assigned with same set of colors is broken in three subcases: A.) σ^1 and σ^4 *or* σ^2 and σ^3 are assigned with same set of colors; B.) Both σ^1 and σ^2 *and* σ^3 and σ^4 are assigned with same set of colors; C.) σ^1 and σ^2 are assigned with different set of colors, while σ^3 and σ^4 are assigned with same set of colors.

A.) Since $s = 2$, we have $C^1 \neq C^2$ and $C^3 \neq C^4$. The same argument of *s=1* applies here to prove the claim.

B.) In this case we know that $C^1 = C^2 \neq C^3 = C^4$. With probability $2/5$, corresponding to configurations T_1, T_3, T_5 and T_7, interval I_1 includes all the intervals of σ^4. Assume I_1 is assigned with a color $c_1 \notin C_{\omega-1}$. With probability $1/4$ configuration T_7 is given. For every color of $C^2 \cup C^3 = C_{\omega-1}$, interval I_2 of configuration T_7 overlaps an interval assigned with that color. Since interval I_2 intersects I_1, it is assigned with a color $c_2 \notin C_{\omega-1}$ distinct from c_1, then proving the claim.
Otherwise, consider the case in which interval I_1 of configuration T_1, T_3, T_5 or T_7 is assigned with a color of $C_{\omega-1}$, say c_1^o. With probability $1/2$, corresponding to configurations T_1 and T_3, interval I_2 includes sequence σ^1. Consider the case in which I_2 is assigned with a color $c_1 \notin C_{\omega-1}$. With probability $1/2$, corresponding to the selection of configuration T_3, interval I_3 overlaps all the intervals of σ^2 and σ^3. For every color of $C^2 \cup C^3 = C_{\omega-1}$, interval I_3 overlaps an interval assigned with that color. I_3 also intersects interval I_2 assigned with a color $c_1 \notin C_{\omega-1}$. Interval I_3 is then assigned with a color $c_2 \notin C_{\omega-1}$ thus proving the claim.
We are left to consider the case of I_1 and I_2 both assigned with a color of $C_{\omega-1}$, say c_1^o and c_2^o. With probability $1/2$, corresponding to configuration T_1, interval I_3 includes sequence σ^2 and interval I_4 includes sequence σ^3. Since $C^1 = C^2$ and $C^3 = C^4$, we have $C^2 \cup c_1^o = C_{\omega-1}$ and $C^3 \cup c_2^o = C_{\omega-1}$. Interval I_3 that overlaps σ^2 and intersects interval I_2, must be assigned with a color $c_1 \notin C_{\omega-1}$. Interval I_4, that includes sequence σ^3, and intersects interval I_1 and I_3, must be assigned with a color $c_2 \notin C_{\omega-1}$, distinct from c_1. The claim is then proved.

C.) In this case $C^1 \neq C^2$ and $C^3 = C^4$. Since $s = 2$, $C^1 \neq C^3$ and $C^2 \neq C^3$. With probability $2/5$, corresponding to configurations T_1, T_3, T_5 or T_7, interval I_1 that includes sequence σ^4 is presented. Consider the case in which interval I_1 is assigned with a color $c_1 \notin C_{\omega-1}$. With probability $1/4$, corresponding to config-

uration T_7, interval I_2, that includes σ^1, σ^2 and σ^3, is presented. For any color of $C_{\omega-1}$, I_2 includes an interval assigned with that color, and intersects interval I_1 assigned with color c_1. I_2 is thus assigned with a color $c_2 \notin C_{\omega-1}$ different from c_1, thus proving the claim.
We finally consider the case of I_1 assigned with a color $c_1^o \in C_{\omega-1}$. With probability $1/4$ configuration T_5 is presented. Interval I_2 overlaps sequences σ^1 and σ^2. Since $C^1 \cup C^2 = C_{\omega-1}$, I_2 is assigned with a new color, say c_1. Since $C^3 = C^4$, $C^3 \cup c_1^o = C_{\omega-1}$. For every color of C^3, interval I_3 overlaps an interval of σ^3 assigned with that color. I_3 also intersects I_1 assigned with color c_1^o and I_2 assigned with color c_1. Interval I_3 is then assigned with a color $c_2 \notin C_{\omega-1}$, thus proving the claim.

s=3 If $C^3 = C^4$, we have that either $C^1 = C^3$ or $C^2 = C^3$, but $C^1 \neq C^2$. Under these assumptions, the same argument used in case C.) of *s=2* allows to prove the claim.

s=4 In case $C^1 = C^2 = C^3 = C^4$, all the sequences are assigned with same set of colors. The same analysis of case $c = 0$ allows to prove the claim.

c=2. In this case $|C_{\omega-1}| = 3\omega - 3$. To prove the claim, a new color must be used with probability at least $1/10$. With probability $1/10$, configuration T_{10} is presented. For any color of $C_{\omega-1}$, interval I_1 overlaps an interval assigned with that color. I_1 is thus assigned with a new color $c_1 \notin C_{\omega-1}$.

c≥3. In this case $|C_{\omega-1}| \geq 3\omega - 2$, the claim is then proved.

■

We finally present the proof of Lemma 1. Let $p(\omega)$ be the probability that a deterministic algorithm uses $3\omega-2$ colors on an input sequence from probability distribution P_ω. Consider a probability distribution $\overline{P}_\omega^j$ formed by probability distributions $P_{\omega-1}^i$, $i = 1, .., 4$. By induction, we assume that the algorithm uses at least $3(\omega-1)-2$ colors with probability $p(\omega-1) \geq (1-e^{-c})$ on an input sequence σ^i drawn from probability distribution $P_{\omega-1}^i$.

With probability $p(\omega-1)^4$, a deterministic algorithm uses at least $3(\omega-1)-2$ colors for all the input sequences σ^i drawn from probability distributions $P_{\omega-1}^i$, $i = 1, .., 4$. With probability $p(\omega-1)^4$ we are then under the assumptions of Lemma 2. We obtain from Lemma 2 that with probability at least $\frac{1}{10}p(\omega-1)^4$, the algorithm uses at least $3\omega-2$ colors for an input sequence from $\overline{P}_\omega^j$, $j = 1, .., \lambda$.

Since all the $\overline{P}_\omega^j$ that form P_ω are mutually independent, the probability that a deterministic algorithm uses less than $3\omega-2$ colors on all the input sequences

drawn from probability distributions $\overline{P}^j_\omega$ is then upper bounded by

$$\left[1-\frac{1}{10}p(\omega-1)^4\right]^\lambda \leq \left[1-\frac{1}{10}(1-e^{-c})^4\right]^\lambda \approx e^{-\lambda\frac{(1-e^{-c})^4}{10}}.$$

If $\lambda \geq \frac{10c}{(1-e^{-c})^4}$, the given expression has value less than e^{-c}.

We have then proved that with probability at least $(1-e^{-c})$, a deterministic algorithm uses at least $3\omega-2$ colors for a sequence from a probability distribution $\overline{P}^j_\omega$, thus implying the claim of Lemma 1.

3 A lower bound for online path coloring on trees

We prove that any randomized algorithm for online path coloring on trees of diameter $\Delta = O(\log n)$ has competitive ratio $\Omega(\log \Delta)$.

We establish the lower bound using Yao's Lemma [Yao77]. We prove a lower bound on the competitive ratio of any deterministic algorithm for a given probability distribution on the input sequences for the problem.

The tree network we use for generating the input sequence is a complete binary tree of $L \geq 4$ levels. The root of the tree is at level 0, the leaves of the tree are at level $L-1$. The 2^l vertices of level l are denoted by r^l_j, $j = 0, ..., 2^l-1$. The direct ancestor of vertex u is denoted by $p(u)$. We will indicate by $[a, b]$ the path in the tree from vertex a to vertex b.

The input sequence for the lower bound is generated in $\rho = \Omega(\log L)$ stages. We will prove that at stage $i = 0.., \rho$, with high probability, the number of colors used by deterministic algorithm is i. An optimal algorithm is shown to be able to color all the paths of the sequence with only 2 colors, thus proving the lower bound.

At stage i of the input sequence, we concentrate on a specific level $l_i = l_{i-1} - \lceil (3^i \log \frac{\rho}{1-1/n} + i \log \rho + \log 4 \log n) \rceil$ of the tree, with $l_0 = L-1$. It turns out that at least $\left(\frac{\rho}{1-\frac{1}{n}}\right)^{3^i} 4\rho^i \log n$ vertices of level l_{i-1} are contained in the subtree rooted at every vertex of level l_i. To simplify notation, the jth vertex of level l_i, $r^{l_i}_j$, is denoted by r^i_j.

We define at stage i a set of pairs $\mathcal{I}_i = \{(u^i_j, v^i_j), j = 0, ..., 2^{l_i}-1\}$, where u^i_j, v^i_j are two leaves of the subtree rooted at vertex r^i_j of level l_i.

Set of pairs $\mathcal{I}_0 = \{(r^0_j, r^0_j), j = 1, ..., 2^L-1\}$ is composed by one degenerated pair for every leaf of the tree.

The set of pairs at stage i is formed selecting at random for every vertex r^i_j, two pairs of stage $i-1$ in the subtree rooted at r^i_j. The pair associated with vertex r^i_j is formed by the two second vertices of the two selected pairs. More formally:

1. For every vertex r_j^i of level l_i, $j = 0, ..., 2^{l_i} - 1$:

   Select uniformly at random two vertices $r_{k_j^1}^{i-1}, r_{k_j^2}^{i-1}$ of level l_{i-1} in the subtree rooted at vertex r_j^i. Let $(u_{k_j^1}^{i-1}, v_{k_j^1}^{i-1}), (u_{k_j^2}^{i-1}, v_{k_j^2}^{i-1}) \in \mathcal{I}_{i-1}$ be the two pairs associated with vertices $r_{k_j^1}^{i-1}$ and $r_{k_j^2}^{i-1}$.

2. $\mathcal{I}_i = \{(v_{k_j^1}^{i-1}, v_{k_j^2}^{i-1}) : j = 0, ..., 2^{l_i} - 1\}$ is the set of pairs at stage i.

The input sequence at stage i is formed for every pair r_j^i of $\mathcal{I}_i$, by a path from the first vertex of the pair to the direct ancestor of vertex r_j^i:

$\mathcal{P}_i = \{[u_j^i, p(r_j^i)] : j = 1, ..., 2^{l_i} - 1\}$.

We prove in the following that any optimal algorithm serves the input sequence with two colors. We first observe:

Lemma 3. *Every edge of the tree is included in at most two paths of* $\cup_{i \geq 0} \mathcal{I}_i$.

Proof. For every vertex r_j^i, for every stage i, denote by E_j^i the set of edges in the subtree rooted at r_j^i with endpoints between level $l_{i-1} - 1$ and level l_i, plus the edge $(r_j^i, p(r_j^i))$. (For a leaf vertex r_j^0, E_j^0 includes the only edge $(r_j^0, p(r_j^0))$). Since all the paths of the input sequence are directed from a leaf vertex to one of its ancestors in the tree, it is sufficient to prove separately for every r_j^i that every edge of E_j^i is included in at most 2 paths of the input sequence.

Edges of E_j^i are not included in any path $\mathcal{P}_{i'}$, $i' < i$. Every leaf vertex is the endpoint of at most one path of the sequence. By the construction of the input sequence, vertices u_j^i and v_j^i are the only leaf vertices in the subtree rooted at r_j^i that may be endpoints of paths in a set $\mathcal{I}_{i'}$, $i' \geq i$. The claim is then proved. ■

The following lemma bounds the size of the optimal solution.

Lemma 4. *The optimal number of colors for any input instance from the probability distribution is 2.*

Proof. We prove the claim for any input instance on a binary tree with: (i.) Every path of the input instance directed from a leaf to an ancestor of the leaf; (ii.) Every edge of the tree included in at most two paths. The claim is proved showing a coloring of all the paths of the input sequence that uses only two colors. We proceed from the top to the bottom of the tree. Consider an internal vertex v (initially the root), and let v_1 and v_2 be the two children of v. Consider edge (v_1, v). (A similar argument holds for edge (v_2, v)).

If no path of the input sequence includes both (v_1, v) and $(v, p(v))$ (assume this is the case if v is the root), edge (v_1, v) is crossed by at most 2 paths, say

p_1 and p_2, that end at v. Paths p_1 and p_2 are assigned with the two available colors. If only one path, say p_1, includes both (v_1, v) and $(v, p(v))$, there is at most one path, say p_2, including edge (v_1, v) that ends at v. Path p_2 is assigned with the color not given to p_1. If there are two paths, p_1 and p_2, including both (v_1, v) and $(v, p(v))$, these have already received colors. The coloring procedure then moves to consider vertex v_1. ■

In the reminder of the section we show that the expected number of colors used by any deterministic online algorithm is $\Omega(\log L)$, thus implying the lower bound.

The following lemma will be used to prove our result.

Lemma 5. *For every pair $(u_j^i, v_j^i) \in \mathcal{I}_i$, path $[v_j^i, p(r_j^i)]$ intersects a single path in every set $\mathcal{P}_j$, $j \leq i$.*

Proof. We prove the claim by induction on the number of stages. The claim is true for pairs of stage 0. Path $[v_j^i, p(r_j^i)]$ is formed by the union of paths $[v_{k_j^2}^{i-1}, p(r_{k_j^2}^i)]$ and $[p(r_{k_j^2}^i), p(r_j^i)]$.

If the claim holds at stage $i-1$, path $[v_{k_j^2}^{i-1}, p(r_{k_j^2}^i)]$ intersects one single path for every $\mathcal{P}_j$, $j \leq i-1$.

Path $[p(r_{k_j^2}^i), p(r_j^i)]$ includes only edges of level lower than $l_{i-1} - 1$. Since no path of a set $\mathcal{P}_j$, $j \leq i-1$, includes edges of level lower than $l_{i-1} - 1$, path $[u_j^i, p(r_j^i)]$ can intersect only paths of level $\mathcal{P}_i$. It certainly intersects path $[u_j^i, p(r_j^i)] \in \mathcal{P}_i$ on edge $(r_j^i, p(r_j^i))$. This is also the single path of $\mathcal{P}_i$ in the subtree rooted at vertex r_j^i, thus showing the claim. ■

We introduce some more notation. Given a pair $(u_j^i, v_j^i) \in \mathcal{I}_i$, let $C_j^i = \{c_0, ..., c_i\}$ be a set of $i+1$ colors. Color c_j is defined to be the color assigned to the single path of $\mathcal{P}_j$ intersecting $[v_j^i, p(r_j^i)]$.

Pair (u_j^i, v_j^i) is a *good pair* if C_j^i is formed by $i+1$ distinct colors. We denote by p_j^i the probability that pair r_j^i is a good pair. We will prove that with high probability, for any stage $i = 0, .., \rho$, there exists at least one good pair of level i. The existence of a good pair of level i gives the evidence that at least $i+1$ colors have been used by the deterministic algorithm, thus proving the claim.

The following claim gives a sufficient condition for a pair of level i to be a good pair.

Lemma 6. *Pair $(u_j^i, v_j^i) \in \mathcal{I}_i$ of level i is a good pair if obtained from selecting two good pairs $(u_{k_j^1}^{i-1}, v_{k_j^1}^{i-1})$, $(u_{k_j^2}^{i-1}, v_{k_j^2}^{i-1})$ with $C_{k_j^1}^{i-1} = C_{k_j^2}^{i-1}$.*

Proof. For every color $c \in C_{k_j^1}^{i-1}$, by Lemma 5, path $[u_j^i, p(r_j^i)]$ intersects a single path assigned with color c. Path $[u_j^i, p(r_j^i)]$ is then assigned with a color $c_i \notin C_{k_j^1}^{i-1}$. For every color $c \in C_{k_j^1}^{i-1} = C_{k_j^2}^{i-1}$ path $[v_j^i, p(r_j^i)]$ intersects a path

assigned with color c. Path $[v_j^i, p(r_j^i)]$ also intersects path $[u_j^i, p(r_j^i)]$ assigned with color c_i on edge $(r_j^i, p(r_j^i))$. Pair (u_j^i, v_j^i) is then a good pair with set of colors $C_{k_j^2}^{i-1} \cup \{c_i\}$. ■

The following lemma bounds the probability that a pair is a good pair:

Lemma 7. *For every pair r_j^i of level l_i, $p_j^i \geq \left(\frac{1-\frac{1}{n}}{\rho}\right)^{3^{i+1}}$.*

Proof. The proof is by induction on the number of stages. The claim is true for $i = 0$. Assume it is true for any pair of $\mathcal{I}_{i-1}$.

Pair $(u_j^i, v_j^i) \in \mathcal{I}_i$ of level i is obtained by selecting two good pairs $i_1 = (u_{k_j^1}^{i-1}, v_{k_j^1}^{i-1})$, $i_2 = (u_{k_j^2}^{i-1}, v_{k_j^2}^{i-1})$ with colors $C_1 = C_{k_j^1}^{i-1}$ and $C_2 = C_{k_j^2}^{i-1}$.

By Lemma 6, the probability that (u_j^i, v_j^i) is a good pair is:

$$\begin{aligned} p_j^i \geq & \Pr[\ i_1 \text{ is a good pair}] \times \Pr[\ i_2 \text{ is a good pair}] \\ & \times \Pr[\ C_1 = C_2 \mid i_1 \text{ and } i_2 \text{ are good pairs}]. \end{aligned}$$

The probabilities that i_1 and i_2 are good pairs are denoted in the following by p_1^{i-1} and p_2^{i-1}. By the inductive hypothesis, $p_1^{i-1}, p_2^{i-1} \geq \left(\frac{1-\frac{1}{n}}{\rho}\right)^{3^i}$.

We are left to determine $\Pr[\ C_1 = C_2 \mid i_1 \text{ and } i_2 \text{ are good pairs}]$.

The event "$C_1 = C_2 \mid i_1$ and i_2 are good pairs" contains the event "there exist two good pairs i_1, i_2, with $C_1 = C_2$ in the subtree rooted at r_j^i" $\cap$ "the two selected good pairs i_1, i_2 have $C_1 = C_2$".

Since the deterministic algorithm uses at most ρ colors, there are at most $\frac{\rho!}{(\rho-i)!i!} \leq \rho^i$ distinct possible set of colors for a good pair at stage $i-1$. The probability that there are at least two good pairs with same set of color is then lower bounded by the probability that there are at least $\rho^i + 1$ good pairs of level $i-1$ in the subtree rooted at r_j^i. Such bound is established by the following claim:

Lemma 8. *The probability that there are at least $\rho^i + 1$ good pairs at stage $i-1$ in the subtree rooted at a vertex r_j^i is at least $1 - \frac{1}{n}$.*

Proof. To establish the claim, we use Chernoff's bounds. We associate to every of at least $\left(\frac{\rho}{1-1/n}\right)^{3^i} 4\rho^i \log n$ pairs of level $i-1$ in the subtree rooted at r_j^i a $\{0,1\}$ random variable X_k. We indicate with $X_k = 1$ that pair (u_k^{i-1}, v_k^{i-1}) is good pair, $X_k = 0$ otherwise. Random variables X_k are independent. This follows from the following fact. Consider any two pairs of level $i-1$ in the subtree rooted at r_j^i, associated with vertices r_1^{i-1} and r_2^{i-1}. Vertices r_1^{i-1} and r_2^{i-1} are the roots of two different subtrees, and all the paths presented until stage $i-1$ with edges in the subtree rooted at r_1^i are colored independently from the paths presented until stage $i-1$ with edges in the subtree rooted at r_2^i.

Random variable X_k has value 1 with probability p_k^{i-1}, value 0 with probability $1 - p_k^{i-1}$.

Under these conditions, we can use Chernoff's bounds to estimate $\Pr[x < \rho^i + 1]$: Let $x = \sum_k X_k$, $\mu = E[x] = \sum_{k=1}^{k} p_k^{i-1}$, $\delta \in (0, 1]$:

$$\Pr[x < (1 - \delta)\mu] < e^{(-\mu\delta^2/2)}.$$

A lower bound over the expected number of good pairs at level $i - 1$ in the subtree rooted at r_j^i is

$$E[x] \geq \overline{\mu} = \left(\frac{1 - \frac{1}{n}}{\rho}\right)^{3^i} \left(\frac{\rho}{1 - \frac{1}{n}}\right)^{3^i} 4\rho^i \log n = 4\rho^i \log n,$$

obtained multiplying the number of pairs at level $i - 1$ in the subtree rooted at r_j^i times a lower bound over the probability that a pair of level $i - 1$ is a good pair. The following expression is easily obtained from the expression of Chernoff's bounds:

$$\Pr[x < (1 - \delta)\overline{\mu}] \leq \Pr[x < (1 - \delta)\mu] < e^{(-\mu\delta^2/2)} \leq e^{(-\overline{\mu}\delta^2/2)}.$$

Setting $\delta = 1 - \frac{1}{4 \log n}$ we obtain:

$$\Pr[x < \rho^i + 1] \leq exp(-2\rho^i \log n(1 - \frac{1}{\log n})^2) \leq \frac{1}{n}.$$

Since $L \geq 4$, the claim follows from $n \geq 15$. ■

We have shown that with probability at least $(1 - \frac{1}{n})$ there exist two good pairs i_1, i_2 with colors $C_1 = C_2 = C$. The two selected pairs of level $i - 1$ are chosen at random between all the pairs in the subtree rooted at r_j^i. Since there are at most ρ^i distinct possible set of colors for a good pair of level $i - 1$, the probability that two good pairs have assigned the same set of colors C is at least $\left(\frac{1}{\rho^i}\right)^2$.

It then follows that $\Pr[\ C_1 = C_2 \mid i_1 \text{ and } i_2 \text{ are good pairs}] \geq \frac{1 - \frac{1}{n}}{\rho^{2i}}$.

The probability that r_j^i is a good pair is then bounded by

$$\begin{aligned} p_j^i &\geq p_1^{i-1} p_2^{i-1} \Pr[\ C_1 = C_2 \mid i_1 \text{ and } i_2 \text{ are good pairs}] \\ &\geq \left(\frac{1 - \frac{1}{n}}{\rho}\right)^{2\ 3^i} \frac{1 - \frac{1}{n}}{\rho^{2i}} \geq \left(\frac{1 - \frac{1}{n}}{\rho}\right)^{2\ 3^i + 2i} \geq \left(\frac{1 - \frac{1}{n}}{\rho}\right)^{3^{i+1}} \end{aligned}$$

■

The construction of the input sequence is repeated until stage $\overline{i} = \rho$ such that $|\mathcal{I}_{\overline{i}}| = 2^{l_{\overline{i}}} \geq \left(\frac{\rho}{1 - \frac{1}{n}}\right)^{3^i} 4\rho^i \log n$. Easy computation shows $\rho = \Omega(\log L)$.

Lemma 8 shows that under these assumptions with probability at least $(1 - \frac{1}{n})$, there exist more than ρ^{ρ} good pairs of level $l_{\rho-1}$. Then, with probability at least $(1 - \frac{1}{n})$, a set C_ρ of ρ distinct colors is used by the deterministic algorithm.

Since $n > 2$, the expected number of colors used by any deterministic algorithm is at least $\rho/2$. Lemma 4 states that the optimal solution uses two colors on any of these input sequences. Thus, the lower bound over the competitive ratio of randomized algorithms is given by $\rho/4$. We then conclude with the following theorem:

Theorem 2. *There exists a $\Omega(\log \Delta)$ lower bound on the competitive ratio of randomized algorithms for online path coloring on a tree of diameter $\Delta = O(\log n)$.*

4 Conclusions

In this paper we have presented the first randomized lower bounds for online interval graph coloring and online path coloring on tree networks. This line of research is aimed to establish if there exists a specific network topology where randomized online algorithms obtain substantially better competitive ratios than deterministic algorithms.

A first open problem is to close the gap between the randomized lower bound for online path coloring on trees and the best deterministic upper bound known for the problem.

The lower bound for path coloring on trees is actually obtained on a 2-colorable graph. This does not preclude the existence of an algorithm that uses $\chi + O(\log \Delta)$ colors for the problem. A second open problem, posed in [BEY98], is to establish a multiplicative lower bound rather than an additive lower bound, i.e. a lower bound on a graph of arbitrary large chromatic number.

References

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.

[AGLR94] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 412–423, 1994.

[BDBK+90] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Widgerson. On the power of randomization in on-line algorithms. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990.

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[BFL96] Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 531–540, 1996.

[BL97] Y. Bartal and S. Leonardi. On-line routing in all-optical networks. In *Proceedings of the 24th International Colloqium on Automata, Languages and Programming*, LNCS 1256, pages 516–526. Springer-Verlag, 1997.

[Gol80] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, NY, 1980.

[GSR96] O. Gerstel, G.H. Sasaki, and R. Ramaswami. Dynamic channel assignment for WDM optical networks with little or no wavelength conversion. In *Proceedings of the 34th Allerton Conference on Communication, Control, and Computing*, 1996.

[HS92] M.M. Halldórsson and M. Szegedi. Lower bounds for on-line graph coloring. In *Proc. of 2th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 211–216, 1992.

[Ira90] S. Irani. Coloring inductive graphs on-line. In *Proceedings of the of 31th Annual IEEE Symposium on Foudations of Computer Science*, pages 470–479, 1990.

[KT81] H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

[KT91] H. A. Kierstead and W. T. Trotter. On-line graph coloring. In Lyle A. McGeoch and Daniel D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 85–92. AMS/ACM, February 1991.

[LMSPR98] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosèn. On-line randomized call-control revisited. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 323–332, 1998.

[LST89] L. Lovász, M. Saks, and W.T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75:319–325, 1989.

[RU94] P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 133–143, 1994.

[ST85] D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of ACM*, 28:202–208, 1985.

[Vis90] S. Vishwanathan. Randomized on-line graph coloring. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 121–130, 1990.

[Yao77] A. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

Parallel Random Search and Tabu Search for the Minimal Consistent Subset Selection Problem

Vicente Cerverón, Ariadna Fuertes

Institut de Robòtica - Departament d'Informàtica i Electrònica - Universitat de València
C/ Dr. Moliner, 50 - 46100 Burjassot (Valencia) - SPAIN
Vicente.Cerveron@uv.es

Abstract. The Minimal Consistent Subset Selection (MCSS) problem is a discrete optimization problem whose resolution for large scale instances requires a prohibitive processing time. Prior algorithms addressing this problem are presented. Randomization and approximation techniques are suitable to face the problem, then random search and meta-heuristics are proposed and discussed. Specifically, Tabu Search emerges as a promising technique, consequently Tabu Search strategies are applied and evaluated. Parallel computing helps to reduce processing time and/or produce better results; different approaches for designing parallel tabu search are analyzed.

1 Introduction

Nearest Neighbor based decision systems used in pattern classification have the *Nearest Prototype Classifier* as the simplest and most widely used classifier. Let $S=\{p_1,...,p_t\} \subset R^d$ be a labeled data set (the reference set), with each $p_i \in S$ (called a prototype) labeled as one of the c classes (when $t \geq c > 1$). *One-Nearest-Neighbor rule* assigns any unlabeled object in Rd to the class of its nearest prototype, according to a specified metric in R^d (usually but not necessarily Euclidean metric) [1]. Despite its simplicity, practical use of NPC is limited by its high computational demands in the operational phase (when classifying unlabeled objects by using a reference set).

In order to reduce the computational demands, the goal is to design a good prototype set of minimal cardinality that will ideally allow for the lowest possible error rate of the classifier. There are two options: *selection*, when we retain a subset (formed by *S-prototypes*) from the original reference set, and *replacement*, when replacing the original data set by a number of labeled prototypes (referred to as *R-prototypes*) that do not necessarily coincide with any original prototype [2]. Selection techniques that find subsets SS guaranteeing zero errors when used to classify the original reference set S are called *condensation techniques*, and the produced subset is said to be consistent with S. Computational efficiency of the operational phase is increased in a ratio given by cardinality(S) / cardinality(SS), so efforts spent in the condensation phase are well worth in the operational phase, when real-time constraints appear.

The aim is to design a method to find a Minimal Consistent Subset. Several papers on the topic presented algorithms condensing or reducing the given reference set ensuring that the selected subset is consistent with the original data set, but none of

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp.248-259, 1998.

them realize the goal of minimal cardinality. Related algorithms tradeoff accuracy for prototype number reduction, i.e., try to select subsets of reduced cardinality (as low as one prototype per class) being their goal the lowest resubstitution error rate. The work reported here is focused on consistent subset selection only. Exact algorithms to ensure a minimal consistent subset (like Branch and Bound) requires exhaustive search and prohibitive processing time.

From a different point of view, the *Minimal Consistent Subset Selection* (MCSS) problem can be considered as a discrete optimization problem (with binary variables set to 1 when a sample is selected and 0 otherwise), suitable for randomization algorithms and meta-heuristics, as they are genetic algorithms, simulated annealing and tabu search [3], that have shown to be successful in a wide range of problems, resulting in good-quality solutions in reasonable times. Specifically, tabu search techniques applied to MCSS problem solving are implemented, analyzed and compared with results produced by other techniques.

Finally, parallel computing offers the advantage of reducing the execution time and its use can also improve the quality of the final solution. In the literature various approaches have been suggested for designing parallel search [4]; some of them are implemented and evaluated in our research.

The remainder of this paper is organized as follows. Prior developments in condensation techniques are presented in Section 2. Randomization and meta-heuristics applied to the MCSS problem are discussed in Section 3, followed by description of Tabu Search strategies and options for the MCSS problem solving in Section 4. Different approaches incorporating parallelism into the search are also discussed in Section 5. Experimental results are given in Section 6, and the last section presents some concluding comments.

2 Selection Algorithms (Condensation Techniques)

2.1 Hart's Algorithm

One of the earliest algorithms for prototype selection was presented as the *"Condensed Nearest Neighbor Rule"* [5] by Hart (named *Hart's Algorithm* in this paper). The testing process in the algorithm ensures that the subset is indeed consistent, but as admitted by the author the goal of minimal subset is not realized. The procedure ends up with a relatively large consistent subset, being very sensitive to the randomly picked initial selection and to the order of consideration of the data.

Hart's elegant method has been used as a basis for many subsequent modifications that, unlike the original procedure, permitted both addition and deletion of samples to and from the condensed subset. For instance, Gates [6] presented an algorithm where the reduced set is derived by iteratively contracting the given set, provided for the possibility of reinsertion of dropped samples until stability is reached. However, this and other proposals, while obtaining smaller subset than Hart's algorithm at a higher computational cost, do not realize the goal of minimality.

Considering the previous reasons we will retain Hart's algorithm as the fastest method to produce consistent subsets, and we will use it later at several points in this paper.

2.2 Dasarathy's Algorithm

In 1994 (Hart presented his method in famous 1968) a study of Dasarathy presented an algorithm [7] (named *Dasarathy's Algorithm* in this paper) for selecting an optimal consistent subset based on his Nearest Unlike Neighbor concept [1]. His approach claimed for benefits as compared to prior approaches, including that the derived subset is aimed to be minimal in size.

Dasarathy's results are basically independent of the order of presentation of the samples and hence the selected consistent subset is unique in terms of the number of samples (the author stated that the result is unique in terms of the selected samples but this is not absolutely true since ties may appear when identifying the most voted sample, and the selected sample is not determined). Consistent property is guaranteed at each iteration, then one could halt the process at a desired iteration (even before the second one); by fact, most of the reduction is achieved at the first.

By comparing its results with those obtained with other approaches, the author believed his method realizes the minimality goal in consistent subset selection, without formal mathematical analysis. However counterexamples to Dasarathy's conjecture may be provided as Kuncheva and Bezdek did in [2], when presented a 12-element consistent subset for the popular IRIS data set (Dasarathy's algorithm finds a 15-element for IRIS). Our research also found a smaller subset (even a 11-element one) than Dasarathy's technique.

Notwithstanding, Dasarathy's algorithm is the best known algorithm in terms of consistent subset size and one of its main features is that samples are selected due to its *representative nature*, then resulting in a negligible loss in recognition efficiency when the full training set is replaced by the selected consistent subset in the operational phase of testing an independent test data set. Therefore we will designate Dasarathy's algorithm as the best classical algorithm for the MCSS problem for further considerations.

3 Randomization and Meta-Heuristics for MCSS

Instead of designing new algorithms or modifications of the previous ones, we can envisage our problem as the exploration of a *solution space* consisting of all possible subsets of a original set, searching for a consistent subset as reduced as possible. Then we may face the problem with random search or with guided search in the form of meta-heuristics. We analyze the pros and cons of the different options and implement some of them.

3.1 Random Restart Procedure

Random exploration of the space of subsets will find lot of subsets revealed inconsistent after evaluation, wasting many efforts in unfeasible solutions. If we constraint the solution space to consistent subsets only, we need some procedure to randomly generate a consistent subset.

Fortunately, there is a simple method to randomly generate consistent subsets: Hart's algorithm described in section 2.1 that with a limited effort produces relatively

reduced subsets. Randomness in Hart's algorithm resides in the initial picked subset and in the order of consideration of samples.

We can use a *random restart procedure* by using Hart's algorithm. At each restart, we randomly pick an initial subset and we generate a random permutation of data, then proceeding until consistency is attained. After each iteration, we retain the current best solution. We can iterate as many times as desired or iterations may stop when a time deadline expires.

Due to the incremental nature of Hart's algorithm (only sample additions are considered, then subset size monotonically increases until consistency is ensured) we can implement a kind of bounding procedure. Any iteration can be halted when the subset size equals the size of the current best solution (no hope to improve it), then restarting a new iteration so saving computational efforts.

Contrary to expectations and despite its naive appearance, this *Restarting Hart's algorithm* incorporating bounding procedure competes surprisingly well as we will show in the Experimental Results section.

3.2 Meta-Heuristics

We characterize our Minimal Consistent Subset Selection problem as that of optimizing (here minimizing) an *objective function* f(x) subject to $x \in X$ (the solution space). As we will discuss below, X may be constrained to consistent subsets then the objective function is simply the subset size, or X may consists of all possible subsets then f(x) is a linear or nonlinear function of size of subset x and number of misclassified prototypes (number of resubstitution errors when the original data set is submitted to subset x as Nearest Prototype Classifier).

In classical heuristic procedures, each $x \in X$ has an associated *Neighborhood* N(x) and each $x' \in N(x)$ is reached from x by an operation called a *Move*. Applying pure *Local Search* concept to MCSS problem, possible moves are sample addition and sample deletion (set or clear binary variables reflecting *i* is present in the selected subset or not). *Descent methods* or *Steepest Descent methods* proceed iteratively from a initial solution to another (a better evaluated neighbor or the best evaluated neighbor in Steepest Descent) until no solutions immediately accessible improve the last one found.

The *meta-heuristic* term coined by Glover in 1986, refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule [3].

The emphasis on guidance distinguish meta-heuristics, based on a variety of interpretations of the "intelligent search" concept, resulting in different meta-heuristics. *Simulated Annealing* (SA) [8] imitate a physical process in metallurgy. *Genetic Algorithms* (GA) [9] are based on the biological phenomenon of evolutionary reproduction (GA are also referred as Evolutionary Computation). We paraphrase an arguable comment from Glover and Laguna claiming that the trend to associate methods with natural processes "embodies a wave of New Romanticism that [...]

suggest that by mimicking the rules we imagine to operate in nature we will similarly be able to produce remarkable outcomes".

The use of Genetic Algorithms for solving the Minimal Consistent Subset Selection problem and other related Prototype Selection problems is presented in a recent study by Kuncheva and Bezdek [2] indicating the capacity of the technique and good-quality results, but no reference to computational complexity or time spent (in their Experimental Result section they only stated 500 iterations are executed).

Tabu Search is an "artificial" meta-heuristic based on selected concepts that unite the fields of artificial intelligence and optimization. The method is based on procedures designed to cross boundaries of feasibility or local optimality. One of its main components is the use of adaptive memory, which create a more flexible search behavior [3][10].

The kind of neighborhood exploration and the use of short-term and long-term memory distinguish Tabu Search from Genetic Algorithms and Simulated Annealing, resulting in lower computational cost and better space exploration for the MCSS problem, then we will describe Tabu Search use in next section.

4 Tabu Search for the Consistent Subset Selection

In our implementation, let X be the solution space consisting of all possible subsets of the original reference set (including both consistent and inconsistent subsets), and let the objective function to minimize f(x) for all $x \in X$ be

$$f(x) = \text{cardinality}(x) + K * \text{errors}(x) \; . \qquad (1)$$

where errors(x) denotes the number of resubstitution errors resulting from x use as NPC. The second term is a *penalty term* weighted by a constant $K \in R^{+}$ whose value will be determined in practice.

Neighborhood definition for each subset $x \in X$ is $N(x) \subset X$ consisting of all subsets that differ from x in only one sample addition or deletion. Then the Move definition is that of adding or deleting a sample to or from the current subset x.

The term Tabu Search involves a lot of techniques and strategies, but it mainly comes from the use of short-term memories (*tabu lists*) that keep track of recently examined solutions intending to avoid cycling in the space exploration. After a move (addition or deletion) is performed, the move is declared *tabu* for a predetermined number of moves, i.e. this move cannot be reversed until a *tabu tenure* expires. This means that TS is a dynamic neighborhood method, where neighborhood of x can change according to the history of the search (this situation is referred as reduced neighborhood). However, a tabu move is admissible if compliant with an *aspiration criterion*, usually that of improving the best current solution.

At each step, we select the least weight non-tabu move from those available (may be an ascending move in some situations of the search), and use the improved-best aspiration criterion to allow a move to be considered admissible in spite of its tabu status. Tabu Search saves best current solution at any time and proceeds iteratively until a chosen termination criterion is satisfied (usually when best solution wasn't improved in M iterations).

In a second level approach, Tabu Search includes additional mechanisms based on long-term memories to direct the search into a promising region (*intensification*) or toward previously unexplored regions of the solution space (*diversification*).

Main features of this Tabu Search implementation for the MCSS problem are presented in the following sections.

4.1 Neighborhood Exploration in TS Compared to GA

Evaluation of a new subset $x \in X$ involves a quadratic cost since for each sample it has to be determined its nearest neighbor present in x by examining distance to all selected prototypes. In contrast to this quadratic cost evaluation for methods relying on random or discontinuous exploration as Genetic Algorithms, evaluation time is reduced in Tabu Search due to TS systematic neighborhood search.

In our implementation for the MCSS problem, a sample addition move is just evaluated by testing whether the newly inserted prototype is closer than the previous nearest neighbor for each class, resulting in linear time cost. A sample deletion move just involves nearest neighbor decision of those samples whose previous nearest neighbor was the deleted one, better than quadratic time. Then neighborhood exploration in TS is more efficient than in GA due to its pure Local Search nature.

4.2 Initial Solutions and Constructive Methods

Tabu Search may start from any initial solution. A first option is to operate on a fully constructed solution (here a consistent subset) produced with other technique (as Hart's algorithm) then guiding transition moves to optimize the condensed subset. A second option is to start from the obvious solution, the full original reference set (consistent with itself) then proceed condensing the selected subset.

The third option considers *constructive moves* for generating initial solutions, being these constructive moves subjected to Tabu Search guidance. This option has significant consequences for the range of strategies available to the meta-heuristic approach, and as we checked in the experimental test, drives to better solutions than former options.

4.3 Constraining to Feasible Regions

In our implementation, solution space exploration makes no distinction between consistent and inconsistent solutions except for the penalty term weighted by the constant k. Consistent solutions form several disconnected regions. If constant k has a high value (say 10) search tends to remain in a local region without "stepping" on inconsistent solutions while crossing to different regions of consistent solutions. A lower value of k (to be 1) allows this boundary crossing, relying on the penalty term to drive the search towards consistent solutions (experimental tests demonstrate that this is the case).

An alternative approach is to constrain neighborhood to moves only among consistent subsets (feasible solutions). To encompass infeasible solutions, the search may be strategically driven to cross the feasibility boundary by deleting samples whose deletion produce inconsistency. After a selected depth is reached (certain

number of samples are dropped) the search changes direction by sample addition driving back toward a feasibility, a consistent solution. A planned use of this called *Strategic Oscillation* will allow to visit the different disconnected feasible regions.

4.4 Intensification and Diversification

Beyond first level Tabu Search approach, use of longer-term memory make possible to better explore promising solutions or regions (*intensification phase*) or to explore less explored regions (*diversification phase*). Two intensification procedures are here proposed. The simplest one is to maintain a list of best solutions then starting first level TS from one of these solutions after clearing all tabu lists that were reducing neighborhood when this solution was found.

Another intensification procedure for the MCSS is to combine best solutions then resulting in a subset containing the most selected prototypes (even reducing to those prototypes present in all best solutions). If the resulting subset is inconsistent then proceeds with constructive moves followed by the local TS phase.

The diversification phase uses a memory containing information relative to visited solutions since the beginning of the search. As in [14] we use an array V representing the number of iterations where sample i is selected. In order to generate a diversified solution, only samples of which V[i] has a value less than a threshold are included in the diversified solution.

4.5 Additional TS Options

Tabu Search comprises a lot of techniques and strategies adaptable to the MCSS problem, as they are *Asymmetrical Tabu Tenures* (a longer tabu tenure for sample addition than for sample deletion, provided that while optimizing there are much more samples to add than to delete), *Candidate List Strategy* for narrowing the examination of elements of a neighborhood in order to achieve an effective tradeoff between the quality of the move and the effort expended in it, or *One Sided Strategic Oscillation* to remain predominantly on the feasible region. Use of these and other TS techniques are beyond the scope of this study and will explored in the near future.

5 Parallelization Strategies

High-performance computing potential offered by parallel computers suggests its use to solve optimization problems by computationally intensive exact algorithms like Branch and Bound [11]. However, solving problems of large dimensions requires a great amount of time even in the presence of efficient parallelization and a high number of processing elements.

Extensive literature is available on parallel search algorithms for discrete optimization techniques [12]. Here we will just analyze parallelization of the randomization and meta-heuristics presented in this paper. Different sources of parallelism exist in Tabu Search algorithm. Four of these sources are:

1. parallelism in cost function evaluation
2. parallelism in problem decomposition

3. parallelism in neighborhood examination
4. parallelism in solution domain exploration by different search paths

The first source represent a low level approach and the second one is not applicable to the MCSS problem. The last two sources of parallelism will be studied in detail.

5.1 Parallelizing Restarting Hart's Algorithm

Parallelization of the random restart procedure presented in section 3.1 is straightforward and suitable for distributed architectures and asynchronous schemes due to its low communication requirements. Initially all processing elements (PE) set a local variable *best_value* to the original set size. Each processing element just proceeds with the restarting Hart's algorithm bounded by this best_value. If an iteration results in a solution smaller than the best_value, the PE saves this solution in local memory and broadcasts its size to all other PEs. After each iteration (improving or not) a non-blocking read is performed to listen to improvements if any, updating best_value to the current lowest value. After a global termination criterion is satisfied (by means of a centralized control) best solution is transferred from the proper local memory.

5.2 Single-Walk Tabu Search

Parallelism in neighborhood examination is also called *single-walk search.* Only a single walk in the solution space is carried out. Since the search for best move at each iteration is a computationally intensive task, moves to be evaluated are distributed over a number of processors. In our parallel TS implementation for the MCSS problem, each processor will be responsible of evaluating state reversal of a group of prototypes. All processors work on the same current solution. A master processor receives best evaluated move from each processor and selects the best one, then communicates the move to slave processors that perform it locally. TS variables and tabu lists remain local to each processor.

This simple parallelization of sequential tabu search produced good performance in optimization problems as Task Scheduling under Precedence Constraints [13]. Its benefits for the problem at issue should be assessed.

5.3 Multiple-Walk Tabu Search

When using parallelism in solution domain exploration, different parallel processes, called *parallel search threads*, are created and distributed over the available PEs. Each search thread consists in executing a TS algorithm from an initial solution (may be equal or different at each thread) and using a set of local parameters. This set of parameters determines the TS behavior specifying a Strategy (again may be unique or multiple).

The simplest approach is to perform *multiple independent walks*. In a better coordinated job, parallel search threads have the possibility to exchange information, then called *interacting walks*. In this case it should be decided the nature of information to be exchanged, such as the occurrence of improved solutions or the

availability of promising paths, and the way to use the additional information resulting from interaction of walks. Interesting results of this approach for the Multidimensional Knapsack Problem are presented in [14]

6 Experimental Results

We have implemented the following techniques in C language:

- Dasarathy's algorithm described in section 2.2.
- Restarting Hart's algorithm as described in section 3.1.
- Tabu Search for the MCSS problem as described in section 4.

For comparison purposes, three different termination criteria are implemented for restarting Hart's and Tabu Search: termination when a given quality solution is reached, termination by given time deadline or termination after a given number of moves without improvement (standard termination criterion).

6.1 Tabu Search Capacities

Just by using a first level Tabu Search approach we have shown Tabu Search capacities. In our first experiment, we used the popular IRIS data set comprising 150 labeled samples in R^4, 50 each from three physically labeled subspecies of IRIS flowers. Dasarathy's technique finds a 15-element consistent subset.

The most reduced known subset for the IRIS set is a 11-element one obtained by Tabu Search with K=1 (the constant in the penalty term of the objective function), tabu tenure of 15 moves and a stopping criterion of 100 moves without improving the best solution (by fact best solution is found in the 279th move), starting from a randomly picked subset (one sample per class) and constructive moves in tabu search style. This solution improves best known result in the literature that was a 12-element subset [2].

6.2 Parallelization Results

For time considerations and parallelization benefits assessment we use a larger data set in further experiments, consisting of 500 samples from two classes in R2 (sample dimension is not relevant for the condensation phase since we may compute all distances in advance).

We have implemented the following parallel versions

- Parallel Restarting Hart's algorithm as described in section 5.1.
- Single-Walk Tabu Search with parallel neighborhood evaluation (see section 5.2.).

Multiple-Walk Tabu Search is currently under development. For the parallel versions we developed a program in C language and used a message passing programming model by using the PVM library. The parallel architecture used during tests is the SGI Origin 2000 with 64 processors R10000 nodes. Applications were compiled with native SGI PVM library. To generalize the results, time are expressed in relative units.

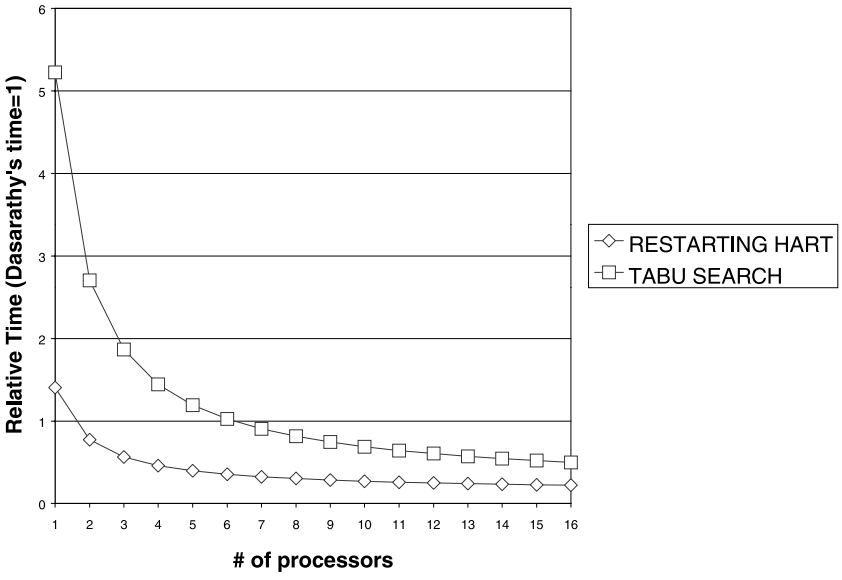


Fig. 1. Time comparison for same or better quality solutions than Dasarathy's algorithm for the 500 prototype test set

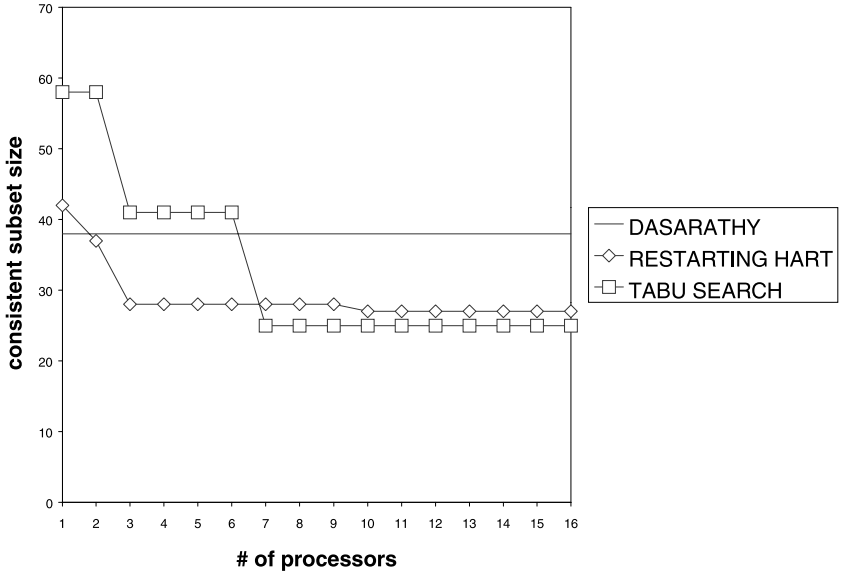


Fig. 2. Quality solution comparison for limited time (Dasarathy's time) for test data set

Our second experiment uses termination by given quality solution, stopping with Dasarathy's result is improved. Figure 1 shows that our Restarting Hart's algorithm comfortably competes with elaborated Dasarathy's procedure, overtaking it with just two processors.

Our third experiment uses termination by deadline expiration, given execution time of sequential Dasarathy's algorithm. Figure 2 shows that Restarting Hart's quickly get better results, but additional computational efforts drive Tabu Search to the best solutions.

Finally, table 1 shows quality of best solution from each technique in longer time, proceeding until termination criterion is satisfied, chosen as a reasonable number of iterations without improvement. Computational effort is here measured in user time for a sequential run in a single processor.

Table 1. Best solution from each technique, for a 500 prototype test data set

Used technique	Subset Size	Computational effort (in relative time)
Dasarathy's	38	1.00
Restarting Hart's	26	22.17
Tabu Search	19	31.49

7 Discussion and Future Work

Our results for the MCSS problem seem to improve those obtained by Genetic Algorithms presented in [2], when using the IRIS data set, resulting in 10 and 11-element subset with one resubstitution error and 12-element consistent subset, while the TS presented here resulted in a 11-element consistent subset.

Our experiments are simply illustrative because of the number of runs, being its purpose to show their capacity and not to analyze robustness, convergence, etc. The points are that a random restarting procedure as the presented Restarting Hart's algorithm easily get good-quality solutions for the MCSS problem and that meta-heuristic Tabu Search get better results than respectable algorithms (and than popular genetic algorithms) and they get them in reasonable times.

While some MCSS techniques require specifying the number of prototypes in advance or they converge to a set whose cardinality cannot be specified or changed as desired. In Tabu Search (and in GA [2] too) the optimal number of prototypes is decided in the course of the computation. As we showed, Tabu Search has many options and degrees of freedom to embed any kind of desired requirements.

Last, parallel implementations allow both reducing execution time and obtaining better solutions. As future work, parallel cooperative (Multiple-Walk) Tabu Search for the MCSS should be implemented and analyzed in depth, and it should be compared to other approaches like parallel genetic algorithms [15] adapted for the MCSS problem.

Acknowledgements

The authors gratefully acknowledge the scientific support provided to this research by Dr. Francesc Ferri and Dr. Ramón Álvarez-Valdés, from the Universitat de València (Spain).

References

1. Dasarathy, B.V.: Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. Los Alamitos, CA: IEEE Computer Society Press, 1991
2. Kuncheva, L.I., Bezdek, J.C.: Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search?. IEEE Trans. Systems, Man, and Cybernetics, Vol. 28, No. 1, pp. 160-164, Feb., 1998
3. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, 1997
4. Verhoeven, M.G.A., Aarts, E.H.L.: Parallel Local Search. Journal of Heuristics, 1 pp. 43-65, Kluwer Academic Publishers, 1995
5. Hart, P.E.: The condensed nearest neighbor rule. IEEE Trans. Information Theory, vol. IT-14, no. 3, pp. 515-516, May 1968
6. Gates, G.W.: The reduced nearest neighbor rule. IEEE Trans. Information Theory, vol. IT-18, no. 3, pp. 431-433, Nov, 1974
7. Dasarathy, B.V.: Minimal Consistent Set (MCS) Identification for Optimal Neighbor Decision Systems Design. IEEE Trans. Systems, Man, and Cybernetics, Vol. 24, No. 3, pp. 511-517 March 1994
8. Dowsland, K.: Simulated Annealing (in Modern Heuristic Techniques for Combinatorial Problems). Ed. C.R.Reeves, Blackwell Scientific Pub., Oxford, 1993
9. Dowsland, K.: Genetic Algorithms. A tool for OR. JORS, vol. 47, pp 550-561, 1996
10. Glover, F., Taillard, E., De Werra, D.: A user's guide to tabu search. Annals of Operations Research, 41, pp. 3-28, 1993
11. Abdelrahman, T.S.: Performance of Parallel Branch and Bound Algorithms on the KSR1 Multiprocessor. Departament of Electrical and Computer Engineering, The University of Toronto, 1993
12. Kumar, V., Grama, A.: Search Algorithms for Discrete Optimization Problems (in Introduction to Parallel Computing). The Benjamin/Cummings Publishing Co., 1994
13. Porto, S.C.S., Ribeiro, S.C.S.: Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling Under Precedence Constraints. Journal of Heuristics, 1 pp. 207-223, Kluwer Academic Publishers, 1995
14. Niar, S., Freville, A.: A Parallel Tabu Search Algorithm For the 0-1 Multidimensional Knapsack Problem. International Parallel Processing Symposium, April 1997
15. Levine, D.: A Parallel Genetic Algorithm for the Set Partitioning Problem. Ph.D. thesis, Illinois Institute of Technology, 1994

On Various Cooling Schedules for Simulated Annealing Applied to the Job Shop Problem*

K. Steinhöfel A. Albrecht[1] C.K. Wong[2]

GMD FIRST
Rudower Chaussee 5
D-12489 Berlin
Germany

Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T.
Hong Kong

Abstract. In this paper, we focus on the complexity analysis of three simulated annealing-based cooling schedules applied to the classical, general job shop scheduling problem. The first two cooling schedules are used in heuristics which employ a non-uniform neighborhood relation. The expected run-time can be estimated by $O\left(n^{3+\varepsilon}\right)$ for the first and $O\left(n^{7/2+\varepsilon}/m^{1/2}\right)$ for the second cooling schedule, where n is the number of tasks, m the number of machines and ε represents $O(\ln\ln n/\ln n)$. The third cooling schedule utilizes a logarithmic decremental rule. The underlying neighborhood relation is non-reversible and therefore previous convergence results on logarithmic cooling schedules are not applicable. Let $l_{\max}$ denote the maximum number of consecutive transition steps which increase the value of the objective function.We prove a run-time bound of $O(\log^{1/\rho} 1/\delta) + 2^{O(l_{\max})}$ to approach with probability $1-\delta$ the minimum value of the makespan. The theoretical analysis has been used to attack famous benchmark problems. We could improve five upper bounds for the large unsolved benchmark problems YN1, YN4, SWV12, SWV13 and SWV15. The maximum improvement has been achieved for SWV13 and shortens the gap between the lower and the former upper bound by about 57%.

1 Introduction

In the job 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 on a given machine. 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. This scheduling problem is NP-hard [7, 18] and there exist problem specifications which are even hard to approximate within a polylogarithmic distance to the optimum solution [23]. To find a schedule that is shorter than 5/4 times the optimum is also NP-hard for the general problem setting [22].

* Research partially supported by the Strategic Research Programme at The Chinese University of Hong Kong under Grant No. SRP 9505.

[1] On leave from BerCom GmbH, Bruno-Taut-Straße 4 - 6, D-12524 Berlin, Germany.

[2] On leave from IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y., U.S.A.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 260-279, 1998.

In the present paper, we are concentrating on the complexity analysis of three simulated annealing-based cooling schedules applied to the general job shop problem. The first two cooling schedules are used in heuristics which employ a detailed analysis of the objective function. For these heuristics we introduced a non-uniform neighborhood relation with biased generation probabilities of neighbors. The preference is given to transitions where a decrease of longest paths is most likely. The expected run-time can be estimated by $O\left(n^{3+\varepsilon}\right)$ for the first and $O\left(n^{7/2+\varepsilon}/m^{1/2}\right)$ for the second cooling schedule, where ε represents $O(\ln\ln n/\ln n)$.

The third cooling schedule utilizes a logarithmic decremental rule. Together with a neighborhood relation introduced by Van Laarhoven et al. in [21] we obtain a stochastic algorithm with a provable convergence rate. The neighborhood relation determines a landscape of the objective function over the configuration space $\mathcal{F}$ of feasible solutions of a given job shop scheduling problem. The general framework of logarithmic cooling schedules has been studied intensely, e.g., by B. Hajek [8] and O. Catoni [3, 4]. To analyze the convergence rate they utilize specific symmetry properties of the configuration space with respect to the underlying neighborhood relation. Our chosen neighborhood relation does not provide these symmetry properties but nevertheless we could perform a convergence analysis of the corresponding stochastic algorithm.

Let $\mathbf{a}_S(k)$ denote the probability to obtain the schedule $S \in \mathcal{F}$ after k steps of a logarithmic cooling schedule. The non-reversible neighborhood from [21] a priori ensures that transitions always result in a feasible solution. Therefore, the problem is to find an upper bound for k such that $\sum_{S\in\mathcal{F}_{\min}} \mathbf{a}_S(k) > 1-\delta$ for schedules S minimizing the makespan. Our convergence result, i.e., the upper 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 $O(\log^{1/\rho} 1/\delta) + 2^{O(l_{\max})}$ to have with probability $1-\delta$ a schedule with the minimum value of the makespan, where $l_{\max}$ is a parameter for the energy landscape. The present approach has been briefly outlined in the context of equilibrium computations in specific physical systems [2].

The theoretical analysis has been used to attack famous benchmark problems. We could improve five upper bounds for the large unsolved benchmark problems YN1, YN4, SWV12, SWV13 and SWV15. The maximum improvement has been achieved for SWV13 and shortens the gap between the lower and the former upper bound by about 57%.

2 The Job Shop Problem

The general job shop scheduling problem can be formalized as follows. There are a set $\mathcal{J}$ of l jobs, a set $\mathcal{M}$ of m machines, and a set $\mathcal{T}$ of n tasks. For each task $t \in \mathcal{T}$ there is a unique job $J(t) \in \mathcal{J}$ to which it belongs, a unique machine $M(t) \in \mathcal{M}$ on which it requires processing, and a processing time $p(t) \in \mathbb{N}$. There is a binary relation R on $\mathcal{T}$ that decomposes $\mathcal{T}$ into chains corresponding to the jobs. This binary relation, which represents *precedences* between the tasks

is defined as follows: For every $t \in \mathcal{T}$ 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. Furthermore, if $(v,w) \in R$ then $M(v) \neq M(w)$. A schedule is a function $S : \mathcal{T} \to \mathbb{N} \cup \{0\}$ that for each task t defines a starting time $S(t)$.

Definition 1 *A schedule is* feasible*, if*

$$\begin{aligned} &\forall v,w \in \mathcal{T} : (v,w) \in R &&\to S(v)+p(v) \leq S(w), \\ &\forall v,w \in \mathcal{T}, v \neq w : M(v) = M(w) &&\to S(v)+p(v) \leq S(w) \vee S(w)+p(w) \leq S(v). \end{aligned}$$

The *length*, respectively the *makespan* of a schedule S is defined by

$$\lambda(S) := \max_{v \in \mathcal{T}} \big(S(v) \ + \ p(v)\big), \tag{1}$$

i.e., the earliest time at which all tasks are completed. The problem is to find an *optimal* schedule, i.e., a feasible schedule of minimum length. Minimizing the makespan $\lambda(S)$ in a job shop scheduling problem with no recirculation can be represented by a *disjunctive graph*, a model introduced by Roy and Sussmann in [16]. The disjunctive graph is a graph $G = (V, A, E, \mu)$, which is defined as follows:

$$\begin{aligned} V &= \mathcal{T} \ \cup \ \{I, O\}, \\ A &= \big\{[v,w] \mid v,w \in \mathcal{T}, (v,w) \in R\big\} \cup \big\{[I,w] \mid w \in \mathcal{T}, \nexists v \in \mathcal{T} : (v,w) \in R\big\} \cup \\ &\quad \big\{[v,O] \mid v \in \mathcal{T}, \nexists w \in \mathcal{T} : (v,w) \in R\big\}, \\ E &= \big\{\{v,w\} \mid v,w \in \mathcal{T}, v \neq w, M(v) = M(w)\big\}, \\ \mu &\ : V \to \mathbb{N}. \end{aligned}$$

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 $v,w \in \mathcal{T}$ 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 $\delta : E \to \mathcal{T} \times \mathcal{T}$ such that $\delta(\{v,w\}) \in \{\langle v,w\rangle, \langle w,v\rangle\}$ for each $\{v,w\} \in E$. A feasible schedule corresponds to an orientation δ on E ($\delta(E) = \{\delta(e) \mid e \in E\}$) for which the resulting directed graph (called digraph) $D := G' = (V, A, E, \mu, \delta(E))$ 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 $(x_i, x_{i+1}, ..., x_j) \in V$ such that for all $i \leq k < j$, $[x_k, x_{k+1}] \in A$ or $\langle x_k, x_{k+1}\rangle \in \delta(E)$.

The length of a path $P(x_i, x_j)$ is defined by the sum of the weights of all vertices in P: $\lambda(P(x_i, x_j)) = \sum_{k=i}^{j} \mu(x_k)$. 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 δ on E that minimizes the length of $\lambda(P_{\max})$.

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. In that case, 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. Therefore, we have $|\mathcal{F}| \leq \left(l!\right)^m$.

To describe the neighborhood of a solution $S \in \mathcal{F}$, we define a *neighborhood function* $\eta : \mathcal{F} \to \wp(\mathcal{F})$. The neighborhood of S is given by $\eta(S) \subseteq \mathcal{F}$, and each solution in $\eta(S)$ is called a neighbor of S. Van Laarhoven et al. [21] propose a neighborhood function η_L which is based on interchanging two adjacent operations of a block. A *block* is a maximal sequence of adjacent operations that are processed on the same machine and do belong to a longest path. We will use an extension of the neighborhood where changing the orientation of a larger number of arcs is allowed within a path related to a single machine:

1. Choosing two vertices v and w such that
   $M(v) = M(w) = k$ and there exists a path $P(v, w)$ with
   $\forall x \in P(v, w) : M(x) = k$ and $\langle v, x\rangle$, $\langle x', w\rangle \in P_{max}$ for $x, x' \in P$;
2. Reversing the order of the path $P(v, w)$ such that
   $\forall \langle x_i, x_j\rangle \in P(v, w) : \langle x_i, x_j\rangle \in \delta(E) \to \langle x_j, x_i\rangle \in \delta'(E)$;
3. 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 $\langle u, w\rangle$;
4. 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 2 *The schedule S' is a neighbor of S, $S' \in \eta(S)$, if S' can be obtained by the transition rules $1 - 4$.*

The transition rules do not guarantee a priori that the resulting schedule is feasible, i.e., that the corresponding digraph is acyclic. Therefore, a test of feasibility has to be performed after any proposed transition. But the feasibility test can be combined with the computation of the length $\lambda(P_{\max})$ which has to be done for any transition. For the special case of η_L, Van Laarhoven et al. have proved the following

Theorem 1 [21] *For each schedule $S \notin \mathcal{F}_{\min}$, there exists a finite sequence of transitions leading from S to an element of $\mathcal{F}_{\min}$.*

As already mentioned in Section 2, the objective is to minimize the makespan of feasible schedules. Hence, we define $\mathcal{Z}(S) := \lambda(P_{\max})$, where $P_{\max}$ is a longest path in $D(S)$. Furthermore, we set

$$\mathcal{F}_{\min} := \{\, S \mid S \in \mathcal{F} \text{ and } \forall S'\big(S' \in \mathcal{F} \to \mathcal{Z}(S') \geq \mathcal{Z}(S)\big) \,\}. \tag{2}$$

We introduce biased generation probabilities which give a preference to transitions where a decrease of longest paths is most likely, i.e., the selection of v, w will depend on the number of longest paths to which $a = \langle v, x_i\rangle$ and $b = \langle x_j, w\rangle$ with $M(x_i) = M(x_j) = k$ do belong in D.

In case of adjacent v and w, i.e., if there is an arc $\langle v, w\rangle \in \delta_S(E)$ and there exist arcs $\langle u, v\rangle$ and $\langle w, x\rangle$ such that $v \neq u, w \neq x$, the transition η_L introduced by Van Laarhoven et al. [21] will be executed. Therefore, the transition η_L is a special case of our neighborhood function.

In simulated annealing, the transitions between neighboring elements depend on the objective function $\mathcal{Z}$. Given a pair of feasible solutions $[S,\, S']$, $S' \in \eta(S)$, we denote by $G[S,\, S']$ the probability of generating S' from S, and by $A[S,\, S']$ the probability of accepting S' once it has been generated from S. Since we consider a single step of transitions, the value of $G[S,\, S']$ depends on the set $\eta(S)$. In most cases, a uniform probability with respect to S is taken which is given by $\mid \eta(S) \mid^{-1}$. In our approach, we are trying to connect the generation probability with the number of longest paths that might be shortened by a single transition. Hence, instead of a single longest path, we have to calculate the number of longest paths to which a single arc $\langle x, y\rangle$ belongs, where $\langle x, y\rangle$ is on the path $P(v, w)$ specified by choosing v and w in the first transition rule. We introduce the following values: $\nu(z) := \lambda\big(P(I, z)\big)$, where $[x, z] \in A$ and $P(I, x)$ is a longest path from I to z, and $\kappa[\langle x, y\rangle] := \mid \{\, P \mid P = P'(I, x)\langle x, y\rangle P''(y, O)$ and $\lambda(P) = \lambda(P_{\max})\} \mid$. The values $\nu(z)$ and $\kappa(z)$ can be computed in expected linear time $O(\mid V \mid)$.

Now, the generation probability depends on the uniform probability $1/m$ of choosing a path $P(z', z'')$ of length l (the number of jobs) such that $M(v) = k$ is fixed for all vertices v of $P(z', z'')$. Then, for any $\langle x, y\rangle$ from $P(z', z'')$, the number of longest paths $\kappa\big[\langle x, y\rangle\big]$ to which $\langle x, y\rangle$ belongs is calculated. We denote

$$g\big[\langle x, y\rangle\big] := \frac{\kappa\big[\langle x, y\rangle\big]}{\sum\limits_{\langle u,v\rangle \text{ on } P(z',z'')} \kappa\big[\langle u, v\rangle\big]}. \tag{3}$$

Next, two independent random choices $\langle x, y\rangle$, $\langle x', y'\rangle$ are made on $P(z', z'')$ in accordance with the probability g. If x precedes x' or $x = x'$, the vertex x is taken as v of the first transition rule, and y' is taken as w.

If the probability of generating a feasible solution S' is denoted by $\tilde{g}\big[S' \mid P(z', z'')\big]$, the generation probability can be expressed by

$$G[S,\, S'] := \begin{cases} 1/m \cdot \tilde{g}\big[S' \mid P(z', z'')\big], & \text{if } S' \in \eta(S), \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The choice of the generation probability has been confirmed by our computational experiments (see Table 1): For our best solutions on the YN and SWV benchmark problems we observed a number of longest paths between one and five, while the maximum number of longest paths was about 50 for the YN and 64 for the SWV benchmark problems, even for solutions close to the upper bounds. Since the neighborhood function η_L from [21] is a special case of our transition rules $1-4$, we have:

Lemma 1 *Given $S \in \mathcal{F}\backslash\mathcal{F}_{\min}$, there exists $S' \in \eta(S)$ such that $G[S,\, S'] > 0$.*

The acceptance probability $A[S,\, S']$, $S' \in \eta(S) \subseteq \mathcal{F}$, is given by:

$$A[S, S'] := \begin{cases} 1, & \text{if } \mathcal{Z}(S') - \mathcal{Z}(S) \leq 0, \\ e^{-\frac{\mathcal{Z}(S') - \mathcal{Z}(S)}{c}}, & \text{otherwise,} \end{cases} \tag{5}$$

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', $S, S' \in \mathcal{F}$, is defined by

$$\mathbf{Pr}\{S \to S'\} = \begin{cases} G[S,\, S'] \cdot A[S,\, S'], & \text{if } S' \neq S, \\ 1 - \sum\limits_{Q \neq S} G[S,\, Q] \cdot A[S,\, Q], & \text{otherwise.} \end{cases} \tag{6}$$

Let $\mathbf{a}_S(k)$ denote the probability of being in the configuration S after k steps performed for the same value of c. The probability $\mathbf{a}_S(k)$ can be calculated in accordance with

$$\mathbf{a}_S(k) := \sum_{Q} \mathbf{a}_Q(k-1) \cdot \mathbf{Pr}\{Q \to S\}. \tag{7}$$

The recursive application of (7) defines a Markov chain of probabilities $\mathbf{a}_S(k)$. 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 $\mathbf{a}(k)$ is an *inhomogeneous* Markov chain.

As pointed out in [21] (see Section 2 there), the convergence to minimum elements of $\mathcal{F}_{\min}$ is based on a subdivision of recurrent computations into irreducible ergodic sets and an additional set of transient elements of $\mathcal{F}$, respectively. From transient feasible solutions, elements of ergodic sets can be reached, but not vice versa. Thus, if $\mathcal{F}_{\min}$ is reachable with a non-zero probability from any $S \in \mathcal{F}$, the following convergence properties can be shown for infinite Markov chains:

$$\lim_{c \to 0} \Big(\lim_{k \to \infty} \sum_{S \in \mathcal{F}\backslash\mathcal{F}_{\min}} \mathbf{a}_S(k) \Big) = 0; \qquad \lim_{c \to 0} \Big(\lim_{k \to \infty} \sum_{S_0 \in \mathcal{F}_{\min}} \mathbf{a}_{S_0}(k) \Big) = 1. \tag{8}$$

Since Lemma 1 provides that $\mathcal{F}_{\min}$ is reachable from any $S \in \mathcal{F}$, we obtain:

Theorem 2 *For Markov chains defined by (4), (5) and (7), the probability to be in a feasible solution $S_0 \in \mathcal{F}_{\min}$ is equal to 1 after an infinite number of steps and for a decreasing control parameter $c \longrightarrow 0$.*

Because the computation of an infinite Markov chain cannot be performed in practice, the calculations have to be interrupted at any fixed "temperature" c after a certain number of steps. Hence, one has to define some heuristic rules bounding the number L_c of transition steps for fixed values $c := c(t)$. Furthermore, it is necessary to determine how the parameter $c(t)$ has to be changed. These problems are discussed in the next section, where two cooling schedules are defined and their complexity will be analyzed.

Additionally, we consider a third cooling schedule which defines a special type of inhomogeneous Markov chains. For this cooling schedule, the value $c(k)$ changes in accordance with

$$c(k) = \frac{\Gamma}{\ln(k+2)},\ k = 0, 1, \ldots . \tag{9}$$

The choice of $c(k)$ is motivated by Hajek's Theorem [8] on logarithmic cooling schedules for inhomogeneous Markov chains. If there exists $S_0, S_1, \ldots, S_r \in \mathcal{F}\big(S_0 = S \wedge S_r = S'\big)$ such that $G[S_u, S_{u+1}] > 0$, $u = 0, 1, \ldots, (r-1)$ and $\mathcal{Z}(S_u) \leq h$, 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 $\mathcal{Z}(S') > \mathcal{Z}(S)$ for all $S' \in \eta_L(S) \setminus S$. By $depth(S_{\min})$ we denote the smallest h such that there exists a $S' \in \mathcal{F}$, where $\mathcal{Z}(S') < \mathcal{Z}(S_{\min})$, which is reachable at height $\mathcal{Z}(S_{\min}) + h$.

The following convergence property has been proved by B. Hajek:

Theorem 3 [8] *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 $\sum_{H \in \mathcal{C}} \mathbf{a}_H(k) \underset{k \to \infty}{\dashrightarrow} 1$ of the stochastic algorithm, which is based on (2), (5), and (6), is guaranteed if and only if

(i) $\forall H, H' \in \mathcal{C}\, \exists H_0, H_1, \ldots, H_r \in \mathcal{C}\big(H_0 = H \wedge H_r = H'\big) : G[H_u, H_{u+1}] > 0$, $l = 0, 1, \ldots, (r-1)$;

(ii) $\forall h : height(H \Rightarrow H') \leq h \iff height(H' \Rightarrow H) \leq h$;

(iii) $\Gamma \geq \max\limits_{H_{\min}} depth(H_{\min})$.

Hence, the speed of convergence associated with the logarithmic cooling schedule (9) is mainly defined by the value of Γ. The condition *(i)* expresses the connectivity of the configuration space. In our case of $\mathcal{F}$, the mutual reachability of schedules cannot be guaranteed as we will show in the following section. Therefore, Hajek's result cannot be applied to our scheduling problem.

4 Two Simulated Annealing-Based Heuristics

The following section describes the main parameter of the two cooling schedules designed for simulated annealing-based heuristics. For both cooling schedules, the starting "temperature" $c(0)$ is defined by

$$e^{-\frac{\Delta\mathcal{Z}^{\max}}{c(0)}} = 1 - p_1, \qquad c(0) = -\frac{\Delta\mathcal{Z}^{\max}}{\ln(1-p_1)}, \tag{10}$$

where p_1 is a small positive value.

The decremental rule of the first cooling schedule is given by the simple relation $c(t+1) := (1-p_2)\cdot c(t)$, where p_2 is a small value larger than zero. The stopping criterion is related to expected number $\widehat{R_c(S)}$ of trials that are necessary for leaving a given configuration S. In case of $L \leq \widehat{R_c(S)}$, it is indeed the time to finish the procedure of simulated annealing. By straightforward calculations, one can show that for arbitrary $S \in \mathcal{F}$ $\widehat{R_c}(S) \leq e^{\Delta\mathcal{Z}^{\max}/c(t)}$, where $\Delta\mathcal{Z}^{\max} := \max_{S\in\mathcal{F}} \max_{S'\in\eta(S)} |\mathcal{Z}(S') - \mathcal{Z}(S)|$. Therefore, one obtains the following conditions:

$$L < e^{\Delta\mathcal{Z}^{\max}/c(t_{fin})}, \quad c(t_{fin}) < \frac{\Delta\mathcal{Z}^{\max}}{\ln L}. \tag{11}$$

Let $\widehat{\chi(c)}$ denote the *expected* ratio of the total number of processed trials and the length L_c at temperature c. The number t_{fin} denotes the number of cooling steps, and we define the average acceptance rate by setting $\overline{\chi} := 1/t_{fin}\cdot\sum_c \widehat{\chi_c}$. Hence, for the length L of Markov chains the number of processed trials from $c(0)$ until $c(t_{fin})$ is given by $\overline{\chi}\cdot L\cdot t_{fin}$. Furthermore, let T denote an upper bound for the time needed to perform the updating of the objective function and the decisions in accordance with (5). Now, the number of steps t_{fin} reducing the parameter $c(t)$ can be calculated from

$$\big(1 - p_2\big)^{t_{fin}} \cdot c(0) = c(t_{fin}),$$

which implies

$$t_{fin} \leq \lceil \frac{1}{\ln(1-p_2)} \cdot \ln\Big(-\frac{\ln(1-p_1)}{\ln L}\Big) \rceil. \tag{12}$$

Therefore, the number of cooling steps does not depend on the objective function. Given the length of Markov chains L, the algorithm has to perform $L\cdot t_{fin}$ *accepted* moves before the algorithm halts because of (11).

Theorem 4 *For the first cooling schedule, the expected run-time is bounded by*

$$T_I \lessapprox \frac{L}{\ln(1-p_2)} \cdot \ln\Big(-\frac{\ln(1-p_1)}{\ln L}\Big) \cdot T \cdot \overline{\chi}.$$

The complexity of updating the objective function and further auxiliary operations can be upper bounded by $O(n)$. Thus, if we assume a square complexity for the basic arithmetic operations, we can use the time bound $T = O(n\cdot\ln^2 n)$. Based on (4), we have for the size of the neighborhood $|\eta(S)| \leq m\cdot l\cdot(l-1)/2 = O(n^2/m)$. Hence, we obtain:

Corollary 1 *For the neighborhood relation $\eta(S)$, the following upper bound of the expected run-time can be derived:*

$$T_I \lessapprox O\Big(\frac{n^3}{m}\cdot\ln^2 n\cdot\overline{\chi}\Big)$$

For the second cooling schedule, the control parameter $c(t)$ is decreased by the rule

$$c(t+1) := \frac{c(t)}{1+\varphi(p_3)\cdot c(t)} = \frac{c(0)}{1+(t+1)\cdot\varphi(p_3)\cdot c(0)}, \tag{13}$$

where $\varphi(p_3)$ is defined by $\varphi(p_3) := \ln(1+p_3)/(\mathcal{Z}^{\max} - \mathcal{Z}_{\min}) < 1$ (see equation (15) in [20]).

Our stopping criterion is derived from the condition

$$c(t)\cdot\frac{\partial\widehat{\mathcal{Z}_c}}{\partial c}\;|_{c=c(t)} \leq \varepsilon\cdot\widehat{\mathcal{Z}^{\max}}, \tag{14}$$

i.e., the changes of the objective function are very small compared to the expected initial value of $\mathcal{Z}$ at $c(0)$. In our specific case the condition leads after a series of transformations to the following inequality:

$$\frac{\ln|\mathcal{F}|}{c(0)}\cdot c^2 + c < \varepsilon\cdot\mathcal{Z}^{\max}. \tag{15}$$

If we assume integer values for the processing times $p(t)$, the minimum improvement of the objective function is lower bounded by 1. Hence, we can take $\varepsilon := 1/\mathcal{Z}^{\max}$ as a lower bound for ε. From (15) we can derive the following upper bound of cooling steps:

$$t_{fin} < \sqrt{\frac{-\ln(1-p_1)\cdot\ln|\mathcal{F}|}{\Delta\mathcal{Z}^{\max}}}\cdot\frac{\mathcal{Z}^{\max}-\mathcal{Z}_{\min}}{\ln(1+p_3)} \tag{16}$$

in case of $\varepsilon := 1/\mathcal{Z}^{\max}$. The upper bound is related to the approximation (15). Finally, we have :

Theorem 5 *For the length L of Markov chains, $\varepsilon := 1/\mathcal{Z}^{\max}$, and the second cooling schedule, the expected run-time can be upper bounded by*

$$T_{II} \lessapprox L\cdot\sqrt{\frac{-\ln(1-p_1)}{\ln^2(1+p_3)}}\cdot\sqrt{\frac{\ln|\mathcal{F}|}{\Delta\mathcal{Z}^{\max}}}\cdot\left(\mathcal{Z}^{\max}-\mathcal{Z}_{\min}\right)\cdot T\cdot\overline{\chi}.$$

We use again $T = O(n\cdot\ln^2 n)$ and obtain:

Corollary 2 *The following upper bound of the expected run-time is valid for the second cooling schedule:*

$$T_{II} \lessapprox O\Big(\frac{n^{7/2}}{m^{1/2}}\cdot\ln^{5/2} n\cdot\overline{\chi}\Big).$$

In the second cooling schedule, the run-time is longer compared to the bound given in Theorem 4, but one has a better control of the final outcome because the objective function is explicitly used in this cooling schedule.

5 The Logarithmic Cooling Schedule

In this section, we consider a uniform generation probability which is based on the neighborhood η_L introduced by Van Laarhoven et al. [21]:

$$G[S, S'] := \frac{1}{|\eta_L(S)|}. \tag{17}$$

Before we perform the convergence analysis of the logarithmic cooling schedule defined in (9), we point out some properties of the configuration space and the neighborhood function. The condition *(i)* of Hajek's Theorem (see Theorem 3 of the previous section) requires that for every two schedules S, S' there exists a finite sequence of transitions leading from S to S'. It is not difficult to construct a problem instance containing pairs of schedules S, S' where such finite sequence does not exist. Moreover, not every transition move is reversible.

Let S and S' be feasible schedules and $S' \in \eta_L(S)$. To obtain S' from S, we chose the arc $e = \langle v, w\rangle$ and $e \in P_{\max}$. If $\mathcal{Z}(S) \geq \mathcal{Z}(S')$, it is not guaranteed that $e' \in P'_{\max}$ and therefore the move might be not reversible.

Lemma 2 *Any transition move which increases the value of the objective function is reversible.*

If the value of the objective function increases, only a path containing one of the selected vertices v, w can determine the new makespan after a transition move. It can be shown that all paths whose length increases contain the edge $e' = \langle w, v\rangle$. Since e' belongs to a longest path it can be selected for the next transition move and the previous step will be reversed. The same idea is used to prove the upper bound $\big(p(v) + p(w)\big)$ for the increase of the objective function value within a single transition step.

Lemma 3 *The increase of the objective function $\Delta\mathcal{Z}$ in a single step according to η_L ($S \longrightarrow_{\eta_L} S'$) can be upper by $\big(p(v) + p(w)\big)$.*

To express the relation between S and S' according to their value of the objective function we will use $<_{\mathcal{Z}}$, $>_{\mathcal{Z}}$, and $=_{\mathcal{Z}}$:

$$\begin{aligned}
S <_{\mathcal{Z}} S' \text{ instead of } & \quad S' \in \eta_L(S)\,\&\,(\mathcal{Z}(S) < \mathcal{Z}(S')),\\
S >_{\mathcal{Z}} S' \text{ instead of } & \quad S' \in \eta_L(S)\,\&\,(\mathcal{Z}(S) > \mathcal{Z}(S')),\\
S =_{\mathcal{Z}} S' \text{ instead of } S \neq S' \,\&\, & S' \in \eta_L(S)\,\&\,(\mathcal{Z}(S) = \mathcal{Z}(S')).
\end{aligned}$$

The notations ${}_{\mathcal{Z}}<$, ${}_{\mathcal{Z}}>$, and ${}_{\mathcal{Z}}=$ will be used for the analogous relation between S and S' in case that S can be generated from S'. Furthermore, we denote:

$$\begin{aligned}
p(S) &:= |\{\, S <_{\mathcal{Z}} S'\}|, & \tilde{p}(S) &:= |\{\, S \,{}_{\mathcal{Z}}< S'\}|,\\
q(S) &:= |\{\, S =_{\mathcal{Z}} S'\}|, & \tilde{q}(S) &:= |\{\, S \,{}_{\mathcal{Z}}= S'\}|,\\
r(S) &:= |\{\, S >_{\mathcal{Z}} S'\}|, & \tilde{r}(S) &:= |\{\, S \,{}_{\mathcal{Z}}> S'\}|.
\end{aligned}$$

These notations imply

$$p(S) + q(S) + r(S) = |\eta_L(S)| - 1. \tag{18}$$

Lemma 4 *For all $S \in \mathcal{F}$, the relations $p(S) \leq \tilde{p}(S)$ and $r(S) \geq \tilde{r}(S)$ are valid. The relation between $q(S)$ and $\tilde{q}(S)$ is not predetermined. Note, the strong inequality between p, $\tilde{p}$ and r, $\tilde{r}$ is possible.*

The relations follow from Lemma 2. For the relation $p(S) \leq \tilde{p}(S)$ we note that there might be a schedule S' which reaches S with a single decreasing transition step, but since a decreasing step is not guaranteed to be reversible, S' is not necessarily a neighbor of S. Therefore, the strong inequality between p, $\tilde{p}$ is possible. The relation $r(S) \geq \tilde{r}(S)$ is considered in a similar way.

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 (9), and we use the notation

$$\frac{1}{\big(k+2\big)^{\frac{\mathcal{Z}(S)-\mathcal{Z}(S')}{\Gamma}}} = e^{-\frac{\mathcal{Z}(S)-\mathcal{Z}(S')}{c(k)}},\ k \geq 0. \tag{19}$$

By using (6) and (17), one obtains from (7) by straightforward calculations

$$\mathbf{a}_S(k) = \mathbf{a}_S(k-1) \cdot \left(\frac{p(S)+1}{\mid \eta_L(S) \mid} - \sum_{\substack{i=1 \\ S <_{\mathcal{Z}} S_i}}^{p(S)} \frac{1}{\mid \eta_L(S) \mid} \cdot \frac{1}{\big(k+1\big)^{\frac{\mathcal{Z}(S_i)-\mathcal{Z}(S)}{\Gamma}}} \right) +$$
$$+ \sum_{\substack{i=1 \\ S_{\mathcal{Z}} \leq S_i}}^{\tilde{p}(S)+\tilde{q}(S)} \frac{\mathbf{a}_{S_i}(k-1)}{\mid \eta_L(S_i) \mid} + \sum_{\substack{j=1 \\ S_{\mathcal{Z}} > S_j}}^{\tilde{r}(S)} \frac{\mathbf{a}_{S_j}(k-1)}{\mid \eta_L(S_j) \mid} \cdot \frac{1}{\big(k+1\big)^{\frac{\mathcal{Z}(S)-\mathcal{Z}(S_j)}{\Gamma}}}.$$

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:

$$L_0 := \mathcal{F}_{\min};\ L_{h+1} := \{S :\ S \in \mathcal{F} \wedge \forall S' \big(S' \in \mathcal{F} \setminus \bigcup_{i=0}^{h} L_i \ \rightarrow\ \mathcal{Z}(S') \geq \mathcal{Z}(S)\big)\}.$$

The highest level within $\mathcal{F}$ is denoted by $L_{h_{max}}$. Given $S \in \mathcal{F}$, we further denote by $\mathcal{W}_{min}(S) := [S, S_{k-1}, \cdots, S']$ a shortest sequence of transitions from S to $\mathcal{F}_{\min}$, i.e., $S' \in \mathcal{F}_{\min}$. Thus, we have for the distance $d(S) := \mathit{length}\big(\mathcal{W}_{min}(S)\big)$. We introduce another partition of $\mathcal{F}$ with respect to $d(S)$:

$$S \in M_i \Longleftrightarrow d(S) = i \geq 0, \quad \text{and} \quad \mathcal{M}_{\mathbf{s}} = \bigcup_{i=1}^{\mathbf{s}-1} M_i, \quad \text{i.e.,} \quad \mathcal{F} = \mathcal{M}_{\mathbf{s}}.$$

Thus, we distinguish between distance levels M_i related to the minimal number of transitions required to reach an optimal schedule from $\mathcal{F}_{\min}$ and the levels L_h which are defined by the objective function. By definition, we have $M_0 := L_0 = \mathcal{F}_{\min}$. We will use the following abbreviations:

$$f(S', S, t) := \frac{1}{\big(k+2-t\big)^{\frac{\mathcal{Z}(S')-\mathcal{Z}(S)}{\Gamma}}} \text{ and} \tag{20}$$

$$(21)\quad D_S(k-t) := \frac{p(S)+1}{\mid \eta_L(S) \mid} - \sum_{\substack{i=1 \\ S <_{\mathcal{Z}} S_i}}^{p(S)} \frac{1}{\mid \eta_L(S) \mid} \cdot \big(k+2-t\big)^{-\frac{\mathcal{Z}(S_i)-\mathcal{Z}(S)}{T}}.$$

During the expansion of $\mathbf{a}_S(k)$, $S \notin M_0$, 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 $\mathbf{a}_{S'}(k-1)$ and can therefore be summarized with terms generated by the expansion of S'. However, it is important to distinguish between elements from M_1 and elements from M_i, $i \geq 2$. For all $S \notin M_1$, we obtain:

$$\mathbf{a}_S(k-1) \cdot \left(\frac{p(S)+1+q(S)+r(S)}{\mid \eta_L(S) \mid} - \sum_{\substack{i=1 \\ S <_{\mathcal{Z}} S_i}}^{p(S)} \frac{1}{\mid \eta_L(S) \mid} \cdot \frac{1}{\big(k+1\big)^{\frac{\mathcal{Z}(S_i)-\mathcal{Z}(S)}{T}}} + \right.$$

$$\left. + \sum_{\substack{i=1 \\ S <_{\mathcal{Z}} S_i}}^{p(S)} \frac{1}{\mid \eta_L(S) \mid} \cdot \frac{1}{\big(k+1\big)^{\frac{\mathcal{Z}(S_i)-\mathcal{Z}(S)}{T}}} \right) = \mathbf{a}_S(k-1).$$

In case of $S \in M_1$, some neighbors S' of S are elements of M_0 and do not generate the terms related to $S >_{\mathcal{Z}} S'$ because the $\mathbf{a}_{S'}(k)$ are not expanded since they are not present in the sum $\sum_{S \notin M_0} \mathbf{a}_S(k)$. Therefore, $r'(S) \leq r(S)$ many terms are missing for $S \in M_1$ and the following arithmetic term is generated:

$$(22)\qquad \mathbf{a}_S(k-1) \cdot \left(1 - \frac{r'(S)}{\mid \eta_L(S) \mid}\right),$$

where $r'(S) := \mid \{S' : S' \in \eta_L(S) \wedge S' \in M_0\} \mid$. On the other hand, the expansion of $\mathbf{a}_S(k)$ generates terms related to $S' \in M_0$ with $S_{\mathcal{Z}} > S'$ and containing $\mathbf{a}_{S'}(k-1)$ as a factor. Those terms are not canceled by expansions of $\mathbf{a}_{S'}(k)$. All $S \in M_1$ therefore generate the following term:

$$(23)\qquad \sum_{\substack{j=1 \\ S_j \in M_0 \cap \eta_L(S)}}^{r'(S)} \frac{\mathbf{a}_{S_j}(k-1)}{\mid \eta_L(S_j) \mid} \cdot \frac{1}{\big(k+1\big)^{\frac{\mathcal{Z}(S)-\mathcal{Z}(S_j)}{T}}}.$$

Now, we consider the entire sum and take the negative product $\mathbf{a}_S(k) \cdot r'(S) / \mid \eta_L(S) \mid$ separately. By using the abbreviations introduced in (21) we derive the following lemma.

Lemma 5 *After one step of the expansion of* $\sum_{S \notin M_0} \mathbf{a}_S(k)$, *the sum can be represented by*

$$\sum_{S \notin M_0} \mathbf{a}_S(k) = \sum_{S \notin M_0} \mathbf{a}_S(k-1) - \sum_{S \in M_1} \frac{r'(S)}{\mid \eta_L(S) \mid} \cdot \mathbf{a}_S(k-1) +$$

$$+ \sum_{S \in M_1} \sum_{\substack{j=1 \\ S_j \in M_0 \cap \eta_L(S)}}^{r'(S)} \frac{f(S,S_j,1)}{\mid \eta_L(S) \mid} \cdot \mathbf{a}_{S_j}(k-1).$$

The diminishing factor $\left(1 - r'(S)/\mid \eta_L(S) \mid\right)$ appears by definition for all elements of M_1. At subsequent reduction steps, the factor is "transmitted" successively to all probabilities from higher distance levels M_i because any element of M_i has at least one neighbor from M_{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

$$(24)\quad \sum_{S \notin M_0} \mathbf{a}_S(k) = \sum_{S \notin M_0} \mu(S,t) \cdot \mathbf{a}_S(k-t) + \sum_{S' \in M_0} \mu(S',t) \cdot \mathbf{a}_{S'}(k-t),$$

i.e., the coefficients $\mu(S,t)$ and $\mu(S',t)$ are the factors at probabilities after t steps of an expansion of $\sum_{S \notin M_0} \mathbf{a}_S(k)$. Hence, for $S \in M_1$ we have $\mu(S,1) = 1 - r'(S)/\mid \eta_L(S) \mid$, and $\mu(S,1) = 1$ for the remaining $S \in \mathcal{M}_\mathbf{s} \backslash (M_0 \cup M_1)$. For $S' \in M_0$ we have from Lemma 5:

$$(25)\quad \mu(S',1) = \sum_{\substack{i=1 \\ S_i \in M_1 \wedge S' \in \eta_L(S_i)}}^{\bar{p}(S')} \frac{f(S_i,S',1)}{\mid \eta_L(S') \mid}.$$

Starting from step $(k-1)$, the generated probabilities $\mathbf{a}_{S'}(k-u)$ are expanded in the same way as all other probabilities. We set $\mu(S,j) := 1 - \nu(S,j)$ because we are mainly interested in the convergence $\mu(S,j) \to 0$. We perform an inductive step from $(k-t+1)$ to $(k-t)$ and obtain for $t \geq 2$:

Lemma 6 *The following recurrent relation is valid for $\nu(S,t)$, $t \geq 2$:*

$$\nu(S,t) = \nu(S,t-1) \cdot D_S(k-t) + \sum_{S \geq_z S'} \frac{\nu(S',t-1)}{|\eta_L(S)|} + \sum_{S <_z S''} \frac{\nu(S'',t-1)}{|\eta_L(S)|} \cdot f(S'',S,t).$$

Furthermore, for the special cases $S \in M_j$, $j > t$, $S \in M_1$, $t = 1$, and $S \in M_0$, $t = 1$ we have, $\nu(S,t) = 0$, $\nu(S,t) = r'(S)/\mid \eta_L(S) \mid$, and $\nu(S,t) = 1 - \sum_{j=1}^{\bar{p}(S)} f(S_j,S,1)/\mid \eta_L(S) \mid$, with $S_j \in M_1 \wedge S \in \eta_L(S_j)$ respectively.

Exactly the same structure of the equation is valid for $\mu(S,t)$ which will be used for elements of M_0 only because these elements are not present in the original sum $\sum_{S \notin M_0} \mathbf{a}_S(k)$. Now, any $\nu(S,t)$ and $\mu(S,t)$ is expressed by a sum $\sum_u T_u$ of arithmetic terms. We consider in more details the terms associated with elements S^0 of M_0 and S^1 of M_1. We assume a representation $\mu(S^0,t-1) = \sum T(S^0)$, and $\nu(S,t-1) = \sum T(S)$, $S \notin M_0$.

If we consider $r'(S^1)/\mid \eta_L(S^1) \mid$ and $\sum_{S^0 <_z S^1} f(S^1,S^0,t)/\mid \eta_L(S^0) \mid$ separately, the difficulties arising from the definition $\nu(S,t) := 1 - \mu(S,t)$ can be avoided, i.e., we have to take into account only changing signs of terms during the transmission from M_1 to M_0 and vice versa.

Definition 3 *The expressions $r'(S^1)/\mid \eta_L(S^1) \mid$, and $\sum_{S^0 <_z S^1} f(S^1,S^0,t)/\mid \eta_L(S^0) \mid$, are called* source terms *of $\nu(S^1,t)$ and $\mu(S^0,t)$, respectively.*

During an expansion of $\sum_{S \notin M_0} \mathbf{a}_S(k)$, the source terms are distributed permanently to higher distance levels M_j. Therefore, at higher distance levels the notion of a source term can be defined by an inductive step:

Definition 4 *For all $S \in M_i$, $i > 1$, any term which is generated according to the equation of Lemma 6 from a source term of $\nu(S', t-1)$, where $S' \in M_{i-1} \cap \eta_L(S)$, is said to be a source term of $\nu(S,t)$.*

We introduce a counter $\mathbf{e}(T)$ to terms T which indicates the step at which the term has been generated from source terms. The value $\mathbf{e}(T)$ is called the rate of a term and we set $\mathbf{e}(T) = 1$ for source terms T.

The value $\mathbf{e}(T) > 1$ is assigned to terms related to M_0 and M_1 in a slightly different way compared to higher distance levels because at the first step the S^{o} do not participate in the expansion of $\sum_{S \notin M_0} \mathbf{a}_S(k)$. Furthermore, in the case of M_0 and M_1 we have to take into account the changing signs of terms which result from the simultaneous consideration of $\nu(S^{\mathrm{1}}, t)$ and $\mu(S^{\mathrm{o}}, t)$.

Definition 5 *A term T^{o} is called a j^{th}* rate term *of $\mu(S^{\mathrm{o}}, t)$ and $j \geq 2$, if either $T^{\mathrm{o}} = -T$ and $\mathbf{e}(T) = j-1$ for some $\nu(S, t-1)$, $S \in M_1 \cap \eta_L(S^{\mathrm{o}})$, or $\mathbf{e}(T^{\mathrm{o}}) = j-1$ for some $\mu(S', t-1)$, $S' \in M_0 \cap \eta_L(S^{\mathrm{o}})$.*

A term T is called a j^{th} rate term *of $\nu(S^{\mathrm{1}}, t)$ and $j \geq 2$, if either $\mathbf{e}(T) = j-1$ for some $\nu(S, t-1)$, $S \in \big(M_1 \cup M_2\big) \cap \eta_L(S^{\mathrm{1}})$, or $T = -T'$, and $\mathbf{e}(T') = j-1$ for some $S' \in M_0 \cap \eta_L(S^{\mathrm{1}})$ with respect to $\mu(S', t-1)$.*

A term T is called a j^{th} rate term *of $\nu(S,t)$, $S \in M_i$ and $i, j \geq 2$, if $\mathbf{e}(T) = j-1$ for some $\nu(S', t-1)$, $S' \in \big(M_i \cup M_{i+1}\big) \cap \eta_L(S)$, or T is a j^{th} rate term of $\nu(S'', t-1)$ for some $S'' \in M_{i-1}$.*

Let $\mathcal{T}_j(S,t)$ be the set of j^{th} rate arithmetic terms of $\nu(S^{\mathrm{1}}, t)$ ($\mu(S^{\mathrm{o}}, t)$) related to $S \in \mathcal{M}_{\mathbf{s}}$. We set $\mathbf{A}_j(S,t) := \sum_{T \in \mathcal{T}_j(S,t)} T$. The same notation is used in case of $S = S^{\mathrm{o}}$ with respect to $\mu(S^{\mathrm{o}}, t)$.

Lemma 7 *If $S \in M_i \neq M_0$, then $\mathbf{A}_j(S,t) = 0$ for $j > t - i + 1$. For S^{o} the condition $j > t$ implies $\mathbf{A}_j(S^{\mathrm{o}}, t) = 0$ and*

$$\nu(S,t) \;=\; \sum_{j=1}^{t-i+1} \mathbf{A}_j(S,t) \quad \textit{and} \quad \mu(S^{\mathrm{o}}, t) \;=\; \sum_{j=1}^{t} \mathbf{A}_j(S^{\mathrm{o}}, t).$$

In order to simplify the analysis of products of factors $D_S(k-t)$ in positive arithmetic terms, we consider the following representation: From Lemma 3 we have

$$\omega := \max_{S' > S \in \mathcal{F}} \mathcal{Z}(S') \;-\; \mathcal{Z}(S) \;\leq\; p(w) + p(v); \tag{26}$$

The upper bound is applied to:

$$D_S(k-t) \;\leq\; \frac{p(S)+1}{\mid \eta_L(S) \mid} \cdot \Big(1 \;-\; \tfrac{1}{2} \cdot \big(k+2-t\big)^{-\dfrac{\omega}{\Gamma}}\Big).$$

The general structure of reductions is explained by

Lemma 8 *The sum $\mathbf{A}_j(S,t)$ of j^{th} rate terms of $\nu(S,t)$ is equal to*

$$\sum_{S_q \in \mathcal{M}_{\mathbf{s}} \setminus M_0} \mathrm{P}_S(j, t-q) \cdot \mathbf{A}_p(S_q, t-q) \;-\; \sum_{S'_q \in M_0} \mathrm{P}'_S(j, t-q) \cdot \mathbf{A}_{p'}(S'_q, t-q),$$

where $p \leq (t-q) - i + 1$, $S_q \in M_i \neq M_0$, *and* $p' \leq t - q$. *The* $\mathrm{P}_S(j, t-q)$, $\mathrm{P}'_S(j, t-q)$ *denote the product of factors* $1/\mid \eta_L(S) \mid$, $(k+2-r)^{-\omega}/\mid \eta_L(S) \mid$, *and* $D_S(k-r)$.

The same structure, with exchanged sets $\mathcal{M}_\mathbf{s} \backslash M_0$ and M_0, is valid for $\mathbf{A}_j(S,t)$, which is part of the value of the coefficient $\mu(S^0, t)$.

Let $pr_{c(k-t+u)}\{S_u \to S_{u+1}\}$ be the factor that is generated according to Lemma 8 during the transition $S_u \to S_{u+1}$. We recall that there are two different types of factors corresponding to transitions $S_u \to S_u = S_{u+1}$: The factor D_{S_u} and the factor $1/\mid \eta_L(S_u) \mid$. The transitions $S_u \to S_u$ are called *selftransitions*. Thus, starting with $S = S_0$, the expansions are performed until the values $\mathbf{A}_1$ have been reached and we denote by

$$\mathcal{P}_S(j,t) = sg(S_{t-1}) \cdot pr_{c(k-t+1)}\{S_0 \to S_1\} \ldots pr_{c(k-1)}\{S_{t-2} \to S_{t-1}\} \cdot \mathbf{A}_1(S_{v-1}, 1)$$

a single product which represents to a particular path of transitions. Here, $sg(S_q) = +1$ for $S_q \notin M_0$, and $sg(S_q) = -1$, otherwise. Based on Lemma 8, $\mathbf{A}_j(S,t)$ can be expressed by

$$\mathbf{A}_j(S,t) \;=\; \sum_{i \in \mathcal{J}} \mathcal{P}^i_S(j,t),$$

where $\mid \mathcal{J} \mid$ depends on the number of neighbors at any step of the expansions.

Since we are interested in absolute values of differences of $\nu_S(k)$, cf. (35), it is sufficient to consider upper bounds for the absolute values of $\mathbf{A}_j(S,t)$. We note that the types of factors $pr_{c(k-t+u)}$ are the same in positive and negative products $\mathcal{P}_S(j,t)$ because the positive summands are recursively generated from negative summands and vice versa. Hence, for an upper bound of $\mid \mathbf{A}_j(S,t) \mid$ we can consider w.l.o.g. only the positive summands of $\mathbf{A}_j(S,t) = \sum_{i \in \mathcal{J}} \mathcal{P}^i_S(j,t)$. This applies to $S \notin M_0$ as well as to $S' \in M_0$. Thus, we obtain

$$\mid \mathbf{A}_j(S,t) \mid \leq \sum_{i \,\in\, \mathcal{J},\; sg \,=\, +1} \mathcal{P}^i_S(j,t) \;\leq\; \sum_{i \in \mathcal{J}} \mid \mathcal{P}^i_S(j,t) \mid . \tag{27}$$

Since we consider absolute values only, we use the notation $\mathcal{P}^i_S(j,t) := \mid \mathcal{P}^i_S(j,t) \mid$, i.e., the sign $sg(S_{t-1})$ is deleted from the product. The computation of products $\mathcal{P}^i_S(j,t)$ can be represented by a tree $\mathbf{T}(S,j,t)$, where $\mathbf{A}_j(S,t)$ denotes the root and the edges denote the transitions $S_u \to S_{u+1}$. The internal nodes are marked by the corresponding S_u, and the node S_u leads to $\mid \eta_L(S_u) \mid +1$ nodes of a greater distance to the root. The $\mid \eta_L(S_u) \mid +1$ results from the two types of factors generated by selftransitions. The edges are marked by the corresponding factor from

$$\frac{1}{\mid \eta_L(S_u) \mid},\; \frac{\big(k+2-t+u\big)^{-\frac{(\mathcal{Z}(S_u)-\mathcal{Z}(S))}{\Gamma}}}{\mid \eta_L(S_u) \mid},\;\; \text{or} \;\; D_{S_u}(k-t+u) \tag{28}$$

with $\mid \eta_L(S_u) \mid$ in the denominator. The leaves are marked by $\mathbf{A}_1(S_{t-1}, 1)$.

We note that the products $\mathcal{P}^i_S(j,t)$ contain $(t-1)$ factors $pr_{c(k-t+u)}$ and we will classify the products mainly by the number $a \leq t-1$ of selftransitions

$S_u \to S_u$, and particularly by the number of factors D_{S_u}. For the different types of transitions let b denote the number of transitions from L_h to a higher neighboring level $L_{h'}$, $h' > h$, c the number of transitions to a lower level $L_{h''}$, and d the number of transitions to the same level L_h. We set $\mathbf{v} := [a, b, c, d]$ and denote by $\mathcal{P}_S[j, t, \mathbf{v}]$ a product containing a factors $D_{S_u}(k - t_u)$ and b, c, d factors of the corresponding type. A given $\mathbf{v}$ induces a fixed subtree $\mathbf{T}_\mathbf{v}$ of $\mathbf{T}$.

Lemma 9 *The position of a particular transition is independent of k and only defined by the structure of the entire computation tree $\mathbf{T}$.*

According to (27) we have

$$|\,\mathbf{A}_j(S,t)\,| \le \sum_{a=0}^{(t-1-b-c-d)} \sum_{b=0}^{(t-1-c-d)} \sum_{c=0}^{(t-1-d)} \sum_{d=0}^{(t-1)} \sum_{i \in \mathcal{J}(\mathbf{T}_\mathbf{v})} \mathcal{P}_S^i[j, t, \mathbf{v}],$$

where $\mathcal{J}(\mathbf{T}_\mathbf{v})$ enumerates the products (paths) in the subtree $\mathbf{T}_\mathbf{v}$.

By Lemma 3, the total increase of the objective function is upper bounded by $b \cdot \big(max\{p(w) + p(v)\}\big) = b \cdot \omega$. Therefore, to reach M_0 or M_1 at most $b \cdot \omega$ decreasing steps are necessary, i.e., $c \le b \cdot \omega$. Since

$$b = (t-1) - a - c - d, \tag{29}$$

we have from $c \le b \cdot \omega$ the relation $b \ge (t-1) - a - b \cdot \omega - d$, $b \cdot (1+\omega) \ge (t-1) - a - d$,

$$b \ge \frac{1}{1+\omega} \cdot \big((t-1) - a - d\big). \tag{30}$$

If $\mathcal{P}_S[j, t, \mathbf{v}]$ contains b transitions to higher levels, the product of the b corresponding factors $f(S_u, S'_u, t_u)$ can be upper bounded by

$$\prod_{u=1}^{b} f(S_u, S'_u, t_u) \le \prod_{u=0}^{b-1} \frac{1}{\big(k + 2 - t_u\big)^{\frac{1}{T}}} \quad < \quad \Big(\frac{e}{b+1}\Big)^{\frac{(b+1)}{T}}. \tag{31}$$

Given a particular product $\mathcal{P}_S(j, t)$, the a factors from selftransitions can be considered together and one obtains the upper bound

$$\prod_{u=1}^{a} \Big(1 - \sum_{i=1}^{p(S)} \frac{\big(k + 2 - t + u\big)^{-\frac{(\mathcal{Z}(S_i) - \mathcal{Z}(S))}{T}}}{n}\Big). \tag{32}$$

The product becomes larger, if only a single transition to higher levels is chosen in any factor. We note that any choice of a single transition is possible. Therefore, it is possible to choose a consecutive chain of increasing values of the objective function $\mathcal{Z}(S_{i_{l-1}}) - \mathcal{Z}(S_{i_l})$; $\mathcal{Z}(S_{i_{l-2}}) - \mathcal{Z}(S_{i_{l-1}})$; $\cdots$; $\mathcal{Z}(S_{i_1}) - \mathcal{Z}(S_{i_2})$. The product is subdivided into sequences of factors which belong to chains of an increasing objective function. We denote by

$$l_{\max} := \max\big\{\, l \,:\, \mathcal{Z}(S_{i_{l-1}}) - \mathcal{Z}(S_{i_l}); \cdots ; \mathcal{Z}(S_{i_1}) - \mathcal{Z}(S_{i_2}) \text{ within } \mathcal{F}\big\} \tag{33}$$

the maximum possible length of an increasing chain within the configuration space $\mathcal{F}$. We obtain a further increase of the product (32) if any $k+2-t+u$ is substituted simply by $k \geq 1$. The factors which belong to the j^{th} increasing chain of length $l_j \leq l_{\max}$ are considered together and by x we denote the number of subdivisions. From (26) we have for $x \geq a/l_{\max}$ and $l_j \leq l_{\max}$

$$(34) \qquad \prod_{u=1}^{a}\Big(1-\sum_{i=1}^{p(S)}\frac{(k+2-t+u)^{-\frac{(\mathcal{Z}(S_i)-\mathcal{Z}(S))}{\Gamma}}}{n}\Big) \;\leq\; e^{-\frac{a}{2^{l_{\max}}\cdot k^{l_{\max}\cdot\omega/\Gamma}}}.$$

Now, we incorporate (31) and (34) in upper bounds of $\sum_{i\in\mathcal{J}(\mathbf{T_v})}\mathcal{P}_S^i[j,t,\mathbf{v}]$ for a fixed $\mathbf{v}$, where we take into account that except for selftransitions of the first type the corresponding factors are divided by $|\,\eta_L(S)\,|$.

Lemma 10 *Given $S \in \mathcal{F}$ and $k > a \geq 0$, then*

$$\sum_{i\in\mathcal{J}(\mathbf{T_v})}\mathcal{P}_S^i[j,t,\mathbf{v}] \;<\; e^{-\frac{a}{2^{l_{\max}}\cdot k^{l_{\max}\cdot\omega/\Gamma}}}\cdot\Big(\frac{(1+\omega)\cdot e}{t-a-d+\omega}\Big)^{\frac{(t-a-d+\omega)}{\Gamma\cdot(1+\omega)}}\cdot\Big(1-\tfrac{1}{n}\Big)^{t-a-b}.$$

Based on the upper bound for the particular products, we can derive an upper bound for the total sum:

Lemma 11 *Given $S \in \mathcal{F}$, $k \geq 2^{O(l_{\max})}$, and $\Gamma \geq O(l_{\max}\cdot\omega)$, then*

$$|\,\mathbf{A}_j(S,t)\,| = \sum_a\sum_b\sum_c\sum_d\sum_{i\in\mathcal{J}_\mathbf{v}}\mathcal{P}_S^i[j,t,\mathbf{v}] \;<\; n\cdot t^{9/2}\cdot 2^{-\frac{k^\rho}{\xi}},$$

where $\xi > 1$ and $\rho > 0$ are suitable constants.

The proof is based on the following three cases, where we assume $t \geq k/\gamma$: $a \geq t/4 \geq k/(4\cdot\gamma)$; $a < t/4$ and $d < 2\cdot a$; $a < t/4$ and $d \geq 2\cdot a$.

Now, we compare the computation of $\nu(S,t)$ (and $\mu(S^{\mathrm{o}},t)$) for two different values $t = k_1$ and $t = k_2$, i.e., $\nu(S,t)$ is calculated backwards from k_1 and k_2, respectively. To distinguish between $\nu(S,t)$ and related values, which are defined for different k_1 and k_2, we will use an additional upper index. At this point, we use again the representation of $\nu(S,t)$ from Lemma 7(and the corresponding equation for $\mu(S^{\mathrm{o}},t)$).

Lemma 12 *Given $k_2 \geq k_1$ and $S \in M_i$, then*

$$\mathbf{A}_2^1(S,t) \;=\; \mathbf{A}_2^2(S,k_2-k_1+t), \quad \textit{if} \qquad t \geq i+2.$$

The proposition can be proved by induction over i, i.e., the sets M_i. Lemma 12 implies that at step $\mathbf{s}+2$ (with respect to k_1)

$$\mathbf{A}_2^1(S,\mathbf{s}+2) = \mathbf{A}_2^2(S,k_2-k_1+\mathbf{s}+2) \quad \text{for all} \quad S \in \mathcal{M}_\mathbf{s}.$$

For $\mathbf{A}_1^1(S,t)$, the corresponding equality is already satisfied in case of $t \geq \mathbf{s}$. The relation can be extended by induction to all values $j \geq 2$:

Lemma 13 *Given $k_2 \geq k_1$, $j \geq 1$, and $S \in M_i$, then*

$$\mathbf{A}_j^1(S,t) = \mathbf{A}_j^2(S, k_2 - k_1 + t), \quad \textit{if} \quad v \geq 2 \cdot (j-1) + i.$$

We recall that our main goal is to upper bound the sum $\sum_{S \notin M_0} \mathbf{a}_S(k)$. If $\mathbf{a}(0)$ denotes the initial probability distribution, we have from (24) for the two values $k_2 \geq k_1$

$$(35) \quad |\sum_{S \notin M_0} \mathbf{a}_S(k_1) - \sum_{S \notin M_0} \mathbf{a}_S(k_2)| \leq |\big(\sum_{S \notin M_0} \nu(S,k_2) - \sum_{S \notin M_0} \nu(S,k_1)\big) \cdot \mathbf{a}_S(0)| + \\ + |\big(\sum_{S' \in M_0} \mu(S',k_1) - \sum_{S' \in M_0} \mu(S',k_1)\big) \cdot \mathbf{a}_{S'}(0)|.$$

Lemma 14 *Given the parameter $l_{\max}$ which characterizes the maximum number of consecutive transition moves which increase the value of the objective function, then there exist a constant $\rho > 0$ and $c > 1$ such that $k_2 \geq k_1 > 2^{O(l_{\max})}$ implies*

$$|\sum_{S \notin M_0} \big(\nu(S,k_2) - \nu(S,k_1)\big) \cdot \mathbf{a}_S(0)| < 2 \cdot 2^{-\frac{k_1^\rho}{c}}.$$

For $|\sum_{S' \in M_0} \big(\mu(S',k_1) - \mu(S',k_2)\big) \cdot \mathbf{a}_{S'}(0)|$ we obtain the corresponding upper bound in the same way.

Theorem 6 *The stochastic algorithm which is defined by (5), (6), and (9) computes for the job shop scheduling problem after*

$$k > (c \cdot \log \frac{6}{\delta})^{\frac{1}{\rho}} + 2^{O(l_{\max})}$$

steps of the inhomogeneous Markov chain a schedule S such that

$$\sum_{S \notin M_0} \mathbf{a}_S(k) < \delta \quad \textit{and therefore,} \quad \sum_{S \in M_0} \mathbf{a}_S(k) > 1 - \delta,$$

Therefore, the probability that after k steps a schedule S has a makespan of minimum length is larger than $1 - \delta$.

Proof: We choose $k > 2^{O(l_{\max})}$ and $\Gamma \geq O(l_{\max} \cdot \omega)$ in accordance with Lemma 11 and 14. Furthermore, we have

$$\sum_{S \notin M_0} \mathbf{a}_S(k) = \sum_{S \notin M_0} \big(\mathbf{a}_S(k) - \mathbf{a}_S(k_2)\big) + \sum_{S \notin M_0} \mathbf{a}_S(k_2) = \sum_{S \notin M_0} \big(\nu(S,k_2) - \nu(S,k)\big) \cdot \mathbf{a}_S(0) + \\ + \sum_{S' \in M_0} \big(\mu(S',k) - \mu(S',k_2)\big) \cdot \mathbf{a}_{S'}(0) + \sum_{S \notin M_0} \mathbf{a}_S(k_2).$$

The value k_2 from Lemma 14 is larger but independent of $k_1 = k$, i.e., we can take a $k_2 > k$ such that $\sum_{S \notin M_0} \mathbf{a}_S(k_2) < \frac{\delta}{3}$.

If additionally both differences $\sum_{S \notin M_0} \big(\nu(S,k_2) - \nu(S,k)\big)$ and $\sum_{S' \in M_0} \big(\mu(S',k) - \mu(S',k_2)\big)$ are smaller than $\delta/3$, we obtain the stated inequality.

Lemma 14 implies that the condition on the differences is satisfied in case of $2 \cdot 2^{-k^\rho/c} < \delta/3$, which is valid, if $\left(c \cdot \log(6/\delta)\right)^{\frac{1}{\rho}} < k$.

q.e.d.

If $l_{\max} \leq n/const.$, Theorem 6 implies an exponential lower bound for the number of steps of the inhomogeneous Markov chain. The theoretical result was used in a simulated annealing procedure to attack famous benchmark problems. The value $l_{\max}$ was estimated by computational experiments for the YN and the SWV benchmark problems. These estimations of $l_{\max}$ were used as the parameter Γ in the logarithmic cooling schedule (9). We could improve five upper bounds for the large unsolved benchmark problems YN1, YN4, SWV12, SWV13, and SWV15. The maximum improvement has been achieved for SWV13 and shortens the gap between the lower and the former upper bound by about 57%. The results are shown in Table 1).

Instance	J × M	LB	UB	$t \to \infty$	CS_{SAW}	Time
YN1	20 × 20	826	888	886	894	70267
YN2	20 × 20	861	909	910	918	63605
YN3	20 × 20	827	894	899	904	61826
YN4	20 × 20	918	972	970	975	63279
SWV11	50 × 10	2983	3005	3017	3149	18271
SWV12	50 × 10	2972	3038	3012	3188	28112
SWV13	50 × 10	3104	3146	3122	3289	33048
SWV15	50 × 10	2885	2940	2924	3088	35477

Table 1. Results on still unsolved problems YN and SWV

LB denotes the lower and UB the upper bounds known from the OR-Library. Five of these upper bounds (YN1, YN4, SWV12, SWV13 and SWV15) could be improved and the gap between LB and UB has been shortened by about 57% on SWV13. For the results in column CS_{SAW} the table indicates the run-time of our simulated annealing procedure CS_{SAW} in seconds on a Sun Ultra 1/170 SPARC machine.

References

1. E.H.L. Aarts. *Local Search in Combinatorial Optimization.* Wiley & Sons, New York, 1998.
2. A. Albrecht, S.K. Cheung, K.S. Leung, and C.K. Wong. Stochastic Simulations of Two-Dimensional Composite Packings. *Journal of Computational Physics*, 136(2):559 – 579, 1997.
3. O. Catoni. Rough Large Deviation Estimates for Simulated Annealing: Applications to Exponential Schedules. *Annals of Probability*, 20(3):1109 – 1146, 1992.
4. O. Catoni. Metropolis, Simulated Annealing, and Iterated Energy Transformation Algorithms: Theory and Experiments. *Journal of Complexity*, 12(4):595 – 623, 1996.
5. J.-H. Cho and Y.-D. Kim. A Simulated Annealing Algorithm for Resource Constrained Project Scheduling Problems. *Journal of the Operational Society*, 48:736–745, 1997.
6. P. Chretienne, E.G. Coffman, Jr., J.K. Lenstra, and Z. Liu. *Scheduling Theory and Its Applications.* Wiley & Sons, New York, 1998.

7. M.R. Garey and D.S. Johnson. Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.
8. B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13:311 – 329, 1988.
9. J.A. Hoogeveen, J.K. Lenstra, and S.L. van de Velde. Sequencing and Scheduling: An Annotated Bibliography. Technical Report COSOR 97–2, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, 1997.
10. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671 – 680, 1983.
11. C.Y. Lee and L. Lei, editors. *Scheduling: Theory and Applications.* Annals of Operations Research, Journal Edition. Baltzer Science Publ. BV, Amsterdam, 1997.
12. C.Y. Lee, L. Lei, and M. Pinedo. Current Trends in Deterministic Scheduling. *Annals of Operations Research*, 70:1–41, 1997.
13. J.F. Muth, G.L. Thompson, and P.R. Winters, editors. *Industrial Scheduling.* Prentice-Hall, Englewood Cliffs, N.J., 1963.
14. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall International Series in Industrial and Systems Engineering. Prentice Hall, Englewood Cliffs, N.J., 1995.
15. P. Ross and D. Come. Comparing Genetic Algorithms, Simulated Annealing and Stochastic Hillclimbing on Timetabling Problems. In T.E. Fogarty, editor, *Evolutionary Computing, Selected Papers*, Lecture Notes in Computer Science, vol. 993, pages 94–102, 1995.
16. B. Roy and B. Sussmann. *Les problèmes d'Ordonnancement avec Constraints Disjonctives.* Note DS No.9 bis. SEMA, 1964.
17. N.M. Sadeh and Y. Nakakuki. Focused Simulated Annealing Search: An Application to Job Shop Scheduling. *Annals of Operations Research*, 63:77–103, 1996.
18. J.D. Ullman. *NP*-Complete Scheduling Problems. *Journal of Computer and System Science*, 10(3):384–393, 1975.
19. R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Local Search. *INFORMS Journal on Computing*, 8:302–117, 1996.
20. P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. *Simulated Annealing: Theory and Applications.* D. Reidel Publ. Comp., Dordrecht, 1988.
21. 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.
22. 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.
23. D. Zuckerman. On Unapproximable Versions of *NP*-Complete Problems. *SIAM Journal on Computing*, 25(6):1293–1304, 1996.

A High Performance Approximate Algorithm for the Steiner Problem in Graphs

Pere Guitart and Josep M. Basart

Departament d'Informàtica, Edifici C
Universitat Autònoma de Barcelona
08193 Bellaterra, Catalunya (Spain)
{pguitart, jmbasart}@ccd.uab.es

Abstract. A new approximate algorithm (GBA) for the Steiner problem in graphs (SPG) based on an iterative execution of a previous heuristic to the problem (SPH) is presented. GBA looks for a subset of vertices which, used as terminals, can produce a near-optimal solution. In addition, the tree associated to each of these subsets is selected at random. The worst-case time complexity of the algorithm is $\mathcal{O}(|V|^3)$ and the cost of the solution is guaranteed to be less than twice the optimal. GBA is tested on classical benchmark problems and its performance compares favorably to that of some of the best existing SPG approaches with respect both to solution quality and specially runtime.

1 Introduction

The SPG asks for the shortest tree interconnecting a subset of vertices in a connected graph. It is one of the NP-hard classical problems in combinatorics and it appears to be very useful in the design of several kinds of communication, distribution and transportation systems. In particular, the wire routing phase in VLSI design can be formulated in terms of the Steiner problem. The SPG allows a substantial variety of exact and heuristic algorithms which produce optimal or near-optimal solutions. While exact methods are only useful for solving small instances of the problem, well-known heuristic techniques must be used in order to attain an approximation to the optimal solution.

There are two main criteria when evaluating the quality of heuristic algorithms. In the first, the best algorithm assures the lowest performance ratio bound (PRB), which is the ratio between worst approximated solution cost and optimal solution cost. In this case, Karpinski and Zelikowsky's [7] algorithm with PRB equal to 1.644 —the solution cost will not be higher than a 64.4% above the optimum— is the best up to now. In the second criterion, the best algorithm performs empirically better in benchmark problem instances. Genetic algorithms [5, 4, 10] had produced the best experimental results up to now with the widely-used graph instances of the OR-library [2].

Both criteria can be considered from a critical perspective. On the one hand, PRBs are often quite difficult to be proved and they are usually close enough to assure better empirical performance. On the other hand, graph instances

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 280-293, 1998.

could not be representative enough, and the number of experiments too small to consider the results statistically significant to the problem. In fact, the algorithms which compete with the empirical criterion are usually based on combination of heuristics with a good PRB. They evaluate a relatively high number of candidate solutions and return the best one they find which, in most of the cases, is nearer to the optimal solution than it should have been expected according to these bounds.

Our aim is to introduce a new approximate algorithm (GBA) based on the well-known *shortest path heuristic* (SPH) proposed by Takahashi and Matsuyama [9]. In GBA, SPH is iteratively applied by using the vertices of the tree attained in the previous iteration as the subset of vertices to be connected in the SPG. This way, our algorithm can obtain solutions which could not be obtained by the SPH itself. In other words, GBA increases the exploration power of SPH because it considers as a candidate solution some trees out of range to the SPH.

This work is organized as follows: SPG notation and some basic properties are presented in section 2. Section 3 describes SPH and introduces a new hill-climbing algorithm (HCSPH) based on this heuristic. In section 4, GBA is introduced by incorporating new elements to HCSPH in order to increase its performance. Experimental results are given in section 5 and, finally, section 6 ends the exposition with some conclusions.

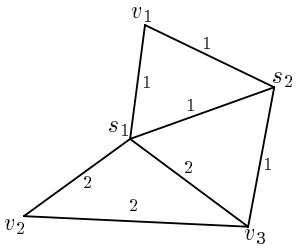
2 SPG Formulation

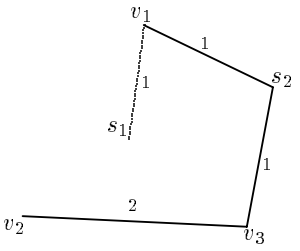
Let $G = (V, E, c)$ be an undirected graph with set of vertices V, set of edges E and a cost function $c : E \to \mathcal{R}^+$ defined on the edges, and let $N \subseteq V$ be a non-empty set of vertices called *terminals*. The SPG is to find a minimum cost subgraph of G which interconnects all the vertices in N. The *cost* —$c(G)$— of a graph G being the sum of the costs of its edges.

A *Steiner tree* of G for N —$StT(G, N)$— is a tree of G interconnecting N. Since the costs of the edges are all positive, the SPG is to find a minimum cost $StT(G, N)$ or $MStT(G, N)$. Vertices not in N —*non-terminals*— from a given $StT(G, N)$ are called *Steiner vertices* and those of them not used to connect N are called *ties*. In other words, a tie is a Steiner vertex with degree one or a Steiner vertex that changes to degree one by recursive deletion of ties. (It is clear that a $MStT(G, N)$ cannot have ties). The function $trim(T)$ returns the tree T once its ties have been deleted or *trimmed*.

A minimum spanning tree of G for $U \subseteq V$ —$MSpT(G, U)$— is a minimum cost tree of G with set of vertices U. It can be found by Prim's $\mathcal{O}(|U|^2)$ or Kruskal's $\mathcal{O}(|E_U|\log|E_U|)$ algorithms, where $E_U \subseteq E$ is the subset of edges of the subgraph of G induced by U. Since a $MStT(G, N)$ with set of Steiner vertices S is also a $MSpT(G, N \cup S)$ the SPG can be solved by finding $S \subseteq V$ so that it minimizes the cost of $MSpT(G, N \cup S)$. The number of Steiner candidate subsets is $2^{|V|-|N|}$.

In Figure 1 there is an example that summarizes this notation.





$G = (V, E, c)$
$N = \{v_1, v_2, v_3\}$ terminals
$V - N = \{s_1, s_2\}$ non-terminals

T is $StT(G, N)$ and $MSpT(G, N \cup \{s_1, s_2\})$
s_2 is a Steiner vertex and s_1 is a tie
$trim(T) = T - \{s_1\}$ is $MStT(G, N)$

Fig. 1. The Steiner Problem in Graphs

The distance graph D_G of G is the complete graph with set of vertices V in which the cost of every edge (v_i, v_j) is given by the cost of the shortest path between v_i and v_j in G. Every $MStT(G, N)$ is also a $MStT(D_G, N)$ and, moreover, given the latter we can obtain the former by replacing every edge by its associated shortest path in G. Since there is a $MStT(D_G, N)$ with at most $k = |N| - 2$ Steiner vertices, D_G can be used instead of G in order to reduce the number of subsets of Steiner vertices to be considered.

3 HCSPH Algorithm

The SPH is one of the classical approximation algorithms for the SPG. Its worst-case time complexity is $\mathcal{O}(|N||V|^2)$ and it returns an approximation — $SPH(G, N)$— never greater than twice the cost of the optimal solution. (The PRB of the SPH is $2 - 1/|N|$) [9]. The SPH works as follows:

- `Step 1`. Let T be a subtree formed by an isolated terminal. Take T as an initial partial solution.
- `Step 2`. Find a terminal v closest to a vertex v' in T. Add to T the shortest path joining v to v'.
- `Step 3`. Repeat step 2 until T contains all the non-terminals ($|N| - 1$ times).

It is easily seen that SPH is closely related to Prim's algorithm for the MSpT problem. In fact, the SPH grows a single subtree which is expanded during each iteration by the addition of a closest terminal v together with the non-terminals on the shortest path from v to the subtree.

Rayward-Smith and Clare [8] noticed that the solution given by this heuristic does not ensure a MSpT and proposed the following two steps to further improve the solution attained by SPH.

- `Step 4`. $T_1 := MSpT(G, vertices(T))$.
- `Step 5`. $T_2 := trim(T_1)$ is the approximate solution.

These two steps do not alter the complexity of the method which depends on the computation of the shortest paths. When the initial tree T is not improved by the steps 4 or 5 these two conditions must hold: T is already a $MSpT(G, vertices(T))$ and T_1 does not have ties. The ties distribution depends on the vertices ordering, that is to say on the way in which MSpT and SPH build their trees. If T_2 is shorter than T, a new improvement can be attained by considering the Steiner vertices of T_2 as terminal vertices and starting again the whole heuristic. This is the main point in which the following new hill-climbing algorithm HCSPH is based on:

- `Step 0`. $V_N := N$.
- `Step 1`. $T_1 := SPH(G, V_N)$; $T_2 := trim(T_1)$.
- `Step 2`. $T_3 := MSpT(G, vertices(T_2))$; $T_4 := trim(T_3)$.
- `Step 3`. if $c(T_1) > c(T_4)$ then $V_N := vertices(T_4)$; go to Step 1
  else T_1 is the approximate solution.

HCSPH "inherits" the PRB from the SPH in the first iteration of the algorithm. It is not difficult to see that $c(T_1) \geq c(T_2) \geq c(T_3) \geq c(T_4)$ so HCSPH is obviously a hill-climbing algorithm. Moreover, it can be used as a hill-climber for any approximate solution given by other SPG heuristic. In that case the set of vertices of this solution should be used instead of N at Step 0.

Complexity Analysis. The worst-case time complexity per iteration of HCSPH is $\mathcal{O}(|V_N||V|^2)$. This is due to the computation of the shortest paths. Since V_N might contain all the vertices, the complexity of a single iteration becomes $\mathcal{O}(|V|^3)$. Consequently, the possibility of computing beforehand all the shortest paths should be taken into account. Once all the shortest paths have been computed by means of *Floyd*'s $\mathcal{O}(|V|^3)$ algorithm, the worst-case and the best-case time complexities per iteration are $\mathcal{O}(|V|^2)$ and $\mathcal{O}(|N|^2)$ respectively.

The worst-case time complexity of HCSPH is $\mathcal{O}(|V|^3 + k \cdot |V|^2)$, where k is the number of iterations. Since in each iteration the cost of the solution is improved, we can assume that k will never be greater than $|V|$ —or $c|V|$ for a small constant c— and, therefore, HCSPH complexity is $\mathcal{O}(|V|^3)$.

It is important to point out that if the complexity of computing all shortest paths cannot be reduced, HCSPH has the lowest complexity that a competitive SPH heuristic —an heuristic aiming to attain the best possible solution— can achieve. In order to being competitive, a graph pre-processing has to be done to reduce the size of the graph. Having in mind this goal, several graph reduction rules have been proposed [6]. Since all the shortest paths are required even for some of the most basic reduction rules, HCSPH complexity cannot be lower than the one of the shortest paths computation. Moreover, the size of the graph could have been reduced so the number of vertices used to evaluate the complexity of one HCSPH iteration can be lower than the number of vertices of the initial —non reduced— graph.

In the next section GBA algorithm is introduced by updating HCSPH basic template in order to increase its performance.

4 GBA

There are only two ways in which a shorter tree can be attained by an iteration of HCSPH: By adding Steiner vertices via SPH or by trimming tie vertices. These two ways depend on the SPH and MSpT implementation, on the vertices ordering and, also, on the method in which the shortest paths are computed. Our proposal here is to introduce new elements aiming to decrease this vertices ordering dependency, to reduce the solution search space and to increase the probability that both Steiner and tie vertices come up.

In the next subsections we explain the main decisions that have been taken to design GBA using HCSPH as a basis. A template of the final version of GBA is shown in the last of these subsections. Therefore, this subsection will provide a detailed description of the way in which these elements have been incorporated to the algorithm.

4.1 Vertices Reordering

Given a subset of vertices V_S we can generally find several $SPH(G, V_S)$ and also some $MSpT(G, V_S)$. However, since the implementation of these functions is usually deterministic, the tree produced given a subset of vertices will always be the same. As a matter of fact, we do not have any *a priori* way to decide which of these trees is better for our purpose. In other words, the way in which SPH and MSpT select their respective solutions is arbitrary so it has nothing to do with the decisions by which the algorithms had been designed. As a result, HCSPH is exploring the space of candidate solutions for the SPG in a slanted way which produces different results even for isomorphic graphs.

Aiming both to avoid unclear slant and to increase GBA exploratory solution power, the vertices are reordered at random before each computation of SPH and MSpT. This way, these functions become non-deterministic and several advantages can be taken:

- Given the same subset of terminals, different trees can be returned by these functions so there are more trees that can effectively be reached by GBA.
- The tendency to return similar trees with consecutive subset of terminals, which are likely to be very similar to each other, is reduced. Consequently, the chances that tie vertices and Steiner ones come up are increased.
- Two iterations starting with the same initial subset of terminals can produce a different solution so the stop conditions given for the HCSPH can be improved. GBA stops when a better solution has not been found after a fixed number of iterations. Using this *last improvement time* —LIT— GBA explores more candidate solutions in order to attain a better approximation.
- Part of the vertices order dependence, which is present in most of the SPG algorithms, is avoided by GBA. (As we show below, GBA does not avoid the dependency when the shortest paths are computed). Consequently, the results obtained by GBA given a graph can be nearly exported to any isomorphic graph.

4.2 Deleting Non-Terminals of Degree Two

One of the ways in which a shorter tree is found by HCSPH is by the inclusion of new Steiner vertices. It is clear that if these vertices have at least degree three they contribute to reduce the cost of the solution. But this is not the case for Steiner vertices of degree two. It can be observed that, at the same time, HCSPH is looking for an approximate solution in both G and D_G. Non-terminals of degree two are completely unnecessary to shorten the tree in D_G.

The function $dd2$ deletes these non-terminals of degree two. In particular, the subset $V_S := dd2(T)$ contains the vertices of the tree T with the exception of its non-terminals of degree two. In *GBA* this subset is used as the initial subset of terminals for the next iteration instead of the vertices of T. It can be observed that a tree T' not larger than T can be constructed by connecting every pair of neighbour vertices of a deleted non-terminal by an edge which cost is equal to the distance between the vertices. By means of $dd2$:

- The tree T' might be shorter than T as well as larger than $MSpT(D_G, V_S)$ so the chances that a shorter tree is found in the next iteration are increased.
- The number of Steiner subsets to be considered is reduced to those with at most $|N| - 2$ vertices, which is the maximum number that V_S can contain, so the solution search space is reduced.
- The SPH $\mathcal{O}(|V_S||V|)$ worst-case time complexity becomes $\mathcal{O}(|N||V|)$, therefore GBA speed-up is increased.

In addition to these advantages, $dd2$ is also a selective way to "shake" the tree in order to embody new Steiner vertices. This so-called shaking effect is considered in the next subsection.

4.3 Shaking the Tree

The inclusion of a function which randomly produces small perturbations to the terminal vertices subset V_S at the end of each iteration can be considered a sensitive point in *GBA*. This function allows each non-terminal vertex to be either included in V_S or excluded from it according to a very small probability. We consider these perturbations as a way of shaking the tree —$shake(V_S)$— in order to make new tie leaves to *fall down* and doing so to leave their place to new Steiner vertices.

It is clear that the number of trees that can be accessed by GBA is increased by means of the shaking function even though it will not necessarily be an advantage. In any case, an important property is incorporated by this function to the algorithm:

- It becomes clear that not all the vertices can be incorporated to a tree by HCSPH algorithm. An example of that is a non-terminal vertex that is not an intermediate vertex in any shortest path. By shaking the tree all the vertices have chances to be added to the terminal vertices subset so all the candidate solutions are reachable. Consequently, no optimal solution is never put aside of consideration.

The shaking must not alter the basic structure of the tree. Consequently, we work out the probability of shaking a vertex depending on the desired expected number of shaken vertices —ES— in the whole tree. For example, if we expect to change —add or delete— ES vertices, the probability of shaking a vertex $P_S(v)$ will be:

$$P_S(v) := ES/(|V| - |N|) \;\; \forall v \in V - N$$

The *shake* function is the most unsteady of all the elements that we have incorporated to GBA, particularly because it enables the possibility of exploring solutions which are larger than the best solution previously found. We have already contemplated the possibility of shaking only when a shorter tree has not been found in the last iteration. Because the global results of our experimentations have improved slightly by shaking at each iteration, we have decided to incorporate the function accordingly. Anyway, by using more selective or careful ways to shake the tree, like $dd2$ function, the algorithm performance could probably be improved.

4.4 Giving Priority to the Paths Instead of the Edges

The performance of path based heuristics for the SPG is closely related to the way in which the paths are constructed. For example, non-terminal vertices that do not appear as intermediate vertex of any of the computed shortest paths, will never be incorporated as Steiner vertex of any tree by the SPH. To make the matters worse, when several same cost paths between two vertices are available, the same arbitrary path is always chosen. This way, some vertices can be arbitrarily put aside of consideration.

We have rejected the possibility of reordering the vertices each time a path is required —in a similar way as it has been done when a tree is computed— because it has an important impact into the algorithm complexity. So, our aim would be to reduce, as far as possible, this new "out of control" slant by trying to select the paths which are likely to be more suitable. We would like to incorporate as intermediate path vertices as many non-terminal vertices as possible.

By now, GBA is at an early stage of this path selection strategy. Since the empirical results are more or less influenced by the way in which the paths are computed, we use a simple path selection strategy that can be explained as follows: Whenever a choice between a path and an edge of the same length is required, the path is selected. This strategy is used in every GBA function as well as in the computation of the paths.

Since this strategy scarcely alters the algorithm in which it is applied, we consider it useful to introduce it in any path based heuristic algorithm.

4.5 GBA Template

The following algorithm shows the final template of GBA.

- `Step 0`. $V_N := N$.
- `Step 1`. $T_1 := SPH(G, reorder(V_N))$; $T_2 := trim(T_1)$.
- `Step 2`. $T_3 := MSpT(G, reorder(vertices(T_2)))$; $T_4 := trim(T_3)$.
- `Step 3`. if $stopCond(LIT)$ then $V_N := shake(dd2(T_4))$; go to Step 1
  else return the shortest tree found.

Complexity Analysis. Two parameters have to be given before GBA can be executed: The LIT to stop the algorithm, and the expected number of shaken vertices per tree ES. The worst-case per iteration of GBA is $\mathcal{O}(|N||V| + |V_2|^2)$, where $|V_2|$ is the number of vertices of T_2. The first term is related to the SPH computation and the second to the MSpT computation. Since V_2 could virtually contain all the vertices, the complexity of a single iteration becomes $\mathcal{O}(|V|^2)$. (In fact, the best-case time complexity is $\mathcal{O}(|N|^2)$.)

We have considered the possibility of suppressing *step 2* in order to reduce the complexity to $\mathcal{O}(|N||V|)$. In that case, T_4 should be replaced by T_2 at step 3. There are two main reasons to reject this possibility. On the one hand, when $dd2$ is applied to a tree that is not a MSpT, some vertices that would produce a shorter tree in the next step might be deleted. These vertices could find it difficult to be incorporated to a solution when $dd2$ is executed after each tree computation. On the other hand, this worst-case time complexity do not show the real behaviour of the algorithm. Assuming that the graph is not sparse enough to use Kruskal's MSpT algorithm instead of the Prim's one, the complexity upper-bound will hold when SPH incorporates most of the non-terminals to T_2 at Step 2 and, besides, this tree has few ties. But it can be seen that this is a very unusual case, specially, when the graph is not sparse. In practice, the complexity of each iteration is usually bounded by the computation of the SPH and, even in this case, the complexity bound does not reflect the real behaviour of one iteration of GBA.

Like in HCSPG, the worst-case time complexity of GBA is $\mathcal{O}(|V|^3 + k|V|^2)$, where k is the number of iterations. In contrast to HCSPG, this number depends on the LIT so it is more difficult to be bounded. A $\mathcal{O}(|V|)$ number of iterations has shown to be enough to produce the empirical results which are shown in the following section. Consequently, we propose to limit the maximum number of iterations to $c|V|$, for a very small constant c. This way, GBA achieves a worst-case time complexity of $\mathcal{O}(|V|^3)$.

5 Experimental Results

GBA have been tested on the 60 largest SPG instances from the OR-Library[2] denoted c1,...,c20, d1,...,d20, e1,...,e20. Every c-graph, d-graph and e-graph has, respectively, 500, 1000 and 2500 vertices.

The performance is compared to that of a genetic algorithm (GA) [5] that clearly outperformed all the previous tested algorithms [3, 4, 10] in both, solution quality and runtime. In [3], up to 6 different algorithms had been compared using the same graph instances in similar conditions. In [5], GA performance was compared to the genetic algorithm proposed by H. Esbensen [4] which had produced the best quality results in the shortest time.

Before GBA itself is executed an attempt to reduce the size of the given problem is performed using standard graph reduction techniques. Since the reduction routines used are the ones produced by Esbensen himself [3], the comparisons have been done with exactly the same problem instances.

Experimental results comparing GBA and GA performances are given in the next four tables. In addition, table 4 includes the quality results obtained by an iterative algorithm (*IKMB*) [1], which is considered as one of the best deterministic heuristics. IKMB results have been extracted from [3] and may give an idea of the difficulty associated to those problem instances. GA results have been taken from [5]. The parameters setting for GBA are $LIT = 1000$ and $ES = 3$. Both algorithms were implemented in C programming language and run on a *SUN Ultra 30*, 250 MHz, 256Mb RAM. Tables 1, 2 and 3 give details of the experiments with graphs c, d and e respectively, and Table 4 summarizes these results.

Table 1. Experimental results with graphs *c*

	Reduced Size			Cost			CPU-Time(Secs)			Iterations	
Graphs	$\|V\|$	$\|N\|$	$\|E\|$	Opt.	GA	GBA	Red.	GA	GBA	Avg.	Max.
c1	145	5	263	85	.0	.0	5	5	0	1	1
c2	130	8	239	144	.0	.0	5	6	1	1	1
c3	120	35	232	754	.0	.1 (0,1)	5	14	2	315	826
c4	109	38	221	1079	.0	.0	5	15	2	143	446
c5	37	17	91	1579	.0	.0	5	4	0	1	1
c6	369	5	847	55	.0	.0	5	10	1	6	46
c7	382	9	869	102	.0	.0	5	12	1	5	46
c8	336	54	818	509	.0	.0	5	37	3	26	228
c9	349	78	832	707	.0	.3 (0,1)	6	67	8	395	1179
c10	213	76	624	1093	.0	.0	6	39	4	92	222
c11	499	5	2184	32	.0	.0	5	13	1	15	75
c12	498	9	2236	46	.0	.0	5	15	1	6	13
c13	463	65	2108	258	.0	.0	5	58	5	72	301
c14	427	81	1961	323	.0	.0	6	55	5	63	175
c15	299	92	1471	556	.0	.0	6	49	4	13	43
c16	500	5	4740	11	.0	.0	5	12	1	5	28
c17	499	9	4698	18	.0	.0	5	13	1	17	47
c18	486	70	4668	113	.1 (0,1)	.0	6	50	4	36	71
c19	473	98	4490	146	.0	.0	6	69	6	55	152
c20	386	143	3850	267	.0	.0	6	85	7	12	27

Tables 1, 2 and 3.

1. Reduced Size: The characteristics of the graphs once reduced.
2. Cost: The cost of the optimal solution (Opt) and the average difference between approximated and optimal costs in 10 executions per graph of both (GA) and (GBA). (For the e-graphs, GA has been executed only once). When the difference is not zero, the minimum and maximum differences between the cost of an approximated solution found and the optimal cost, is given in brackets on the right side.
3. CPU-Time(Secs): The time in seconds spent in the pre-processing to reduce the graphs size (Red.) and the average time used by both (GA) and (GBA) without taking into account the pre-processing.
4. Iterations: The average (Avg.) and the maximum (Max.) number of iterations until the best solution has been found by GA.

Table 2. Experimental results with graphs *d*

	Reduced Size			Cost			CPU-Time(Secs)			Iterations	
Graphs	$\|V\|$	$\|N\|$	$\|E\|$	Opt.	GA	GBA	Red.	GA	GBA	Avg.	Max.
d1	274	5	510	106	.0	.0	50	9	1	8	16
d2	285	10	523	220	.0	.0	50	13	1	2	11
d3	224	67	441	1565	.0	.2 (0,2)	51	42	5	198	519
d4	159	66	339	1935	.0	.0	51	33	3	3	9
d5	97	48	246	3250	.0	.0	52	17	2	84	321
d6	761	5	1741	67	.0	.0	50	22	1	16	45
d7	754	10	1735	103	.0	.0	50	22	1	1	1
d8	731	124	1708	1072	.2 (0,1)	1.3 (0,3)	51	259	25	257	620
d9	654	155	1613	1448	.0	.3 (0,2)	52	348	30	274	556
d10	418	146	1317	2110	.0	.8 (0,2)	53	156	19	303	889
d11	993	5	4674	29	.0	.0	50	27	2	128	270
d12	1000	10	4671	42	.0	.0	50	29	2	1	1
d13	922	122	4433	500	.0	.0	51	232	19	108	344
d14	853	160	4173	667	.0	.0	52	317	24	71	195
d15	550	157	2925	1116	.0	.0	54	146	19	89	385
d16	1000	5	10595	13	.0	.0	50	23	1	11	53
d17	999	9	10531	23	.0	.0	51	25	1	3	13
d18	978	145	10140	223	1.3 (1,2)	1.4 (1,2)	50	320	24	406	967
d19	938	193	9676	310	1.4 (1,2)	1.0 (1,1)	51	353	29	182	468
d20	814	324	8907	537	.0	.0	52	568	63	26	78

Table 3. Experimental results with graphs *e*

	Reduced Size			Cost			CPU-Time(Secs)			Iterations	
Graphs	$\|V\|$	$\|N\|$	$\|E\|$	Opt.	GA	GBA	Red.	GA	GBA	Avg.	Max.
e1	680	5	1286	111	0	.0	773	18	1	2	6
e2	710	9	1328	214	0	.0	773	21	1	13	37
e3	637	199	1233	4013	0	7.4 (4,10)	779	442	52	684	1895
e4	435	164	964	5101	0	2.1 (1,3)	783	220	25	661	1524
e5	222	108	649	8128	0	.0	800	62	6	164	393
e6	1845	5	4318	73	0	.0	756	56	3	7	28
e7	1891	10	3388	145	0	.0	775	66	3	23	128
e8	1723	286	4193	2640	1 (1,1)	2.4 (0,5)	791	1803	202	618	1219
e9	1608	358	4069	3604	0	3.9 (2,6)	799	2431	320	978	1481
e10	1046	366	3388	5600	0	4.3 (1,6)	827	1427	122	169	582
e11	2498	5	12093	34	0	.0	780	81	3	7	27
e12	2500	10	12123	67	0	.0	777	87	4	107	641
e13	2341	321	11760	1280	2 (2,2)	2.0 (0,4)	788	3461	312	882	2898
e14	2139	388	11325	1732	0	1.6 (1,2)	802	3930	229	179	361
e15	1461	443	8514	2784	0	.0	837	2002	180	201	504
e16	2500	5	29332	15	0	.0	781	74	3	19	80
e17	2500	10	29090	25	0	.0	779	76	3	11	35
e18	2429	355	28437	564	7 (7,7)	2.8 (2,4)	781	2300	193	502	1823
e19	2351	485	27779	758	3 (3,3)	0.4 (0,1)	787	2923	310	374	653
e20	1988	758	24423	1342	0	.0	801	5078	381	37	51

Table 4.

1. Error: The percentage of solutions of (IKMB), (GA) and (GBA) with error not greater that (0%) and (1%), and the percentage of graphs for which (GBA) has found an optimal solution when 10 executions per graph are considered (Best 10).
2. Avg. CPU-Time(Secs): (GA) and (GBA) average time per execution. (Partial) does not include the graphs pre-processing time while (Global) does.

Table 4. Summary of the experiments

	Error							Avg. CPU-Time(Secs)			
	IKMB		GA		GBA			GA		GBA	
Class	0%	< 1%	0%	< 1%	0%	< 1%	Best 10	Partial	Global	Partial	Global
c	55.0	80.0	99.5	100	98.0	100	100	31	37	3	8
d	35.0	80.0	89.0	100	80.5	100	90	148	199	14	65
e	36.8	68.4	80.0	95.0	59.0	100	70	1328	2116	118	906
Total	42.3	76.1	89.5	98.3	79.2	100	86.7	502	785	45	326

Performance analysis. Summarizing from Tables 1 to 4, GBA finds a globally optimal solution in 79.2% of all runs and is within 1% from a global optimum in all the executions. These results clearly overcome the ones obtained by IKMB but are slightly inferior to the ones obtained by means of GA. However, the time difference in the execution of the two algorithms greatly makes up for the small quality differences in the results. When the time of pre-process to reduce the graphs size, which includes that of the shortest paths computation, is not taken into account, GBA turns out to be 11 times faster than GA. As indicated in [5], GA was already clearly faster than the rest of algorithms seen. Taking the pre-processing time into account, GBA turns out to be more than twice as much faster than GA.

It is important to highlight the impact that the stop conditions chosen for each one of those algorithms have in the final results. We have verified that when more time is given to GBA, the quality of the results tend to get very near to GA. We feel that GBA converges in a much faster way towards approximate good solutions, which, we ought to remember, are the real object of our algorithm.

Finally, we cannot help stating that the results obtained both by GA and GBA can be considered to be surprisingly excellent, baring in mind that the problem is NP-hard. Part of these results can be attributed to the weakness of some of the graphs, in particular those with a small number of terminals, a fact proved with the small number of iterations that, as shown in tables 1, 2 and 3, are needed to find the optimum one. GBA attains better results than GA for graph e18 and e19 which, in theory, are the strongest graphs of the benchmark. We conclude that more tests with difficulty graphs would be helpful to decide which of the two algorithms performs definitely better.

6 Conclusions

A new approximate algorithm (GBA) for the SPG based on the SPH has been introduced. The central point of the algorithm is the iterative execution of the SPH using the vertices of the tree obtained in the previous iteration as a new set of terminals to be connected. Since the initial set of terminals is used at the first iteration, GBA assures the performance ratio bound of the SPH so the solution cost will be lower than twice the optimal solution cost. GBA can also be used to obtain a shorter tree once an approximation has been given by means of another SPG heuristic. In this case, the set of vertices of the approximate tree should be used as the initial set of terminals.

The worst-case time complexity of GBA is $\mathcal{O}(|V|^3)$, where $|V|$ is the number of vertices. (This bound is achieved by limiting the maximum number of iterations to $c|V|$ for a small constant c.) Since our aim is to improve the results in comparison to other SPG-heuristics, it is necessary to perform some routines in order to reduce beforehand the size of the graphs. Since shortest paths —Floyd's $\mathcal{O}(|V|^3)$ algorithm— are also required for these routines, GBA complexity is the lowest that a well performing SPH heuristic can achieve.

Three additional elements have been incorporated to GBA with the purpose of improving its performance. Firstly, the vertices are randomly reordered each time that a new tree is computed. Secondly, the Steiner vertices of degree two are not considered as terminals in the next iteration. Finally, small perturbations are applied to the new terminal subset of vertices allowing the inclusion or exclusion of vertices at random.

It has been shown that, at least one of the three additional elements contributes to each of the following positive effects to the algorithm:

- To increase the chances that both non-terminal leaves and Steiner vertices come up and therefore, that shorter trees might be obtained.
- To increase the capability of the algorithm to explore new solutions.
- To expand the number of candidate solutions that can be reached by the algorithm.
- To avoid "out of control" slants which depend on both the vertices order and the particular algorithm implementation.
- To reduce the solution space to subsets of Steiner vertices with no more elements than the number of terminals. (It is known that there is at least one optimal solution with these characteristics.)
- To give to these non-terminals which can never be incorporated to a solution by means of the SPH, the possibility of becoming part of a tree.
- To increase the speed-up of the algorithm.

GBA has replaced arbitrariness to non-determinism by means of the random vertices reordering. As a consequence, GBA becomes *near*[1] independent of the vertices order and its empirical results can be *nearly* exported to any isomorphic graph instance.

The performance of the algorithm has been tested on well-known benchmark instances with random graphs with up to 2,500 vertices and 62,500 edges. Experimental results show that in all the executions GBA finds a solution which is within 1% from the global optimum. Moreover, the optimal solution is found in 79% of all the executions and in 87% of the 60 larger graphs of the benchmark when 10 executions per graph are considered.

This performance is compared to that of a recently appeared genetic algorithm (GA) which clearly outperformed all the previous heuristics in both solution quality and runtime. The solutions attained by GA are marginally better than these of GBA. However, GBA clearly outperforms GA when runtime is considered. Using as stop conditions those from which both algorithms have attained these results and when the pre-processing time to reduce the graphs size is not taken into account, GBA is about 11 times faster on average than GA. Taking the pre-processing time into account, GBA turns out to be more than twice as much faster than GA.

[1] This reordering does not affect the shortest path computation

Acknowledgments

The authors would like to thank Dr. H. Esbensen for allowing us the use of some *C* routines produced by himself and M. Guitart for helping us with the writing.

References

1. Alexander, M. J., Cohhon, J. P., Ganley, J. L., Robins. G.: An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs. Proc. of The European Design Automation Conference (1994) 259–264
2. Beasley, J.E.: OR-Library: distributing test problems by electronic mail. J. Opl. Res. Soc. **41** (1990) 1069–1072
3. Esbensen, H.: Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm. Networks **26** (1995) 173–185
4. Esbensen, H.: Finding (Near-)Optimal Steiner Trees in Large Graphs. Proc. of the Sixth International Conference on Genetic Algorithms (1995) 485–491.
5. Guitart, P., Basart, J.M.: A Genetic Algorithm Approach for the Steiner Problem in Graphs. EUFIT'98 Proc. of the Sixth European Congress on Intelligent Techniques and Soft Computing (1998) (to appear).
6. Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem, Annals of Discrete Matematics. **53** North-Holland, Amsterdam, The Netherlands (1992)
7. Karpinski, M., Zelikovsky, A.Z.: New Approximation Algorithms for the Steiner Tree Problems. Journal of Combinatorial Optimization **1** (1997) 1–19
8. Rayward-Smith, V.J., Clare, A.: On finding Steiner Vertices. Networks **16** (1986) 283–294
9. Takahashi H., Matsuyama A.: An approximate solution for the Steiner problem in graphs. Math. Jap. **24** (1980) 573–577
10. Voβ, S., Gutenschwager, K.: A Chunking Based Genetic Algorithm for the Steiner Tree Problem in Graphs. DIMACS Series in Discrete Mathematics and Theoretical Computer Science **40** (1997) 335–355

Random Geometric Problems on $[0,1]^2$ *

Josep Díaz, Jordi Petit, and Maria Serna

Departament de Llenguatges i Sistemes
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C-6
Jordi Girona Salgado 1–3
08034-Barcelona, Spain
{diaz,jpetit,mjserna}@lsi.upc.es

Abstract. In this paper we survey the work done for graphs on random geometric models. We present some heuristics for the problem of the Minimal linear arrangement on $[0,1]^2$ and we conclude with a collection of open problems.

1 Introduction

The probabilistic method has become a powerful tool in combinatorics. A particular fruitful application of the probabilistic method has been the study of graph invariants for random graphs (chromatic number, independence number, etc.). The field started in 1959 with a paper by Erdös and Rényi [ER59]. In the last two decades, the techniques developed in the probabilistic method have been used in the design and analysis of algorithms. There have been two models of random graphs: Given $n \in \mathbb{N}$ and $0 \leq p \leq 1$, define $G_{n,p}$ as the probability space over the set of graphs on vertex set $V = [n] = \{1, \ldots, n\}$, and such that any two vertices $i, j \in V$ form an edge with probability p. For the second model, given $n, m \in \mathbb{N}$, let $G_{n,m}$ be the probability space of the graphs with vertex set $V = [n]$ and edge set E_m, a random subset of m edges from all possible edges in the complete graph on n vertices. In general $G_{n,m}$ will behave similarly to $G_{n,p}$ as $p \sim m/\binom{n}{2}$. Recall that $f \sim g$ denotes that $f/g \to 1$ as the variables tend to infinity. Erdös and Rényi [ER60] considered the $G_{n,m}$ model to study the threshold for several questions related to the connectivity of graphs. To study the evolution of graphs, they start with the empty graph on n vertices and add edges randomly one by one until having the m edges. Their main result is that **with high probability (whp)** a graph becomes connected when $m \geq \frac{n \log n}{2} + O(n)$. Recall that a sequence of events $\{\mathcal{E}_n\}$ occurs whp if $\Pr[\mathcal{E}_n] \to 1$ as $n \to \infty$. Therefore, for a threshold of $p = o(\frac{\log n}{n})$, whp the graphs $G_{n,p}$ will be connected. Good sources for random graphs are [Bol85,FM96] and chapter 10 of [ASE92]. See [MR95] for the "adaptation" of the probabilistic method to the design of algorithms.

* This research was partially supported by the ESPRIT LTR Project no. 20244 – ALCOM-IT, CICYT Project TIC97-1475-CE and CIRIT project 1997SGR-00366

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 294-306, 1998.

In this paper we will consider the geometric models of random graphs. Geometric random graphs are defined on the unit cube $[0,1]^d, d \geq 2$ by randomly choosing a sequence $X = \{X_n\}$ of independent and uniformly distributed points on $[0,1]^d$. We can consider two main models: **the Euclidian model**, considering the weighted complete graph on X, where the weight of an edge is its Euclidian distance (see section 2), and the **random geometric model** $G_n(r)$, where the existence of edges depends on a parameter $0 < r < 1$ that could be a function of n (see section 3). There is another model of geometric graph, called **the independent model**, where the distances are **independent and identically distributed (i.i.d.)** random variables from the uniform distribution on $[0,1]$ and neither symmetry nor triangle inequality are assumed. The model, was introduced in [Kar77], it is a difficult model to work with and we shall not enter into it (see for example [AB92] and chapter 2 of [Wei78] for an interesting discussion on the independent model).

Before going further, let us review some basic definitions from probability theory. Recall than given a sequence of random variables $\{X_n\}$ and a random variable X, we say that $X_n \to X$ **converges in probability** if $\forall \epsilon > 0, \Pr[|X_n - X| > \epsilon] \to 0$ as $n \to \infty$. We say that $X_n \to X$ **converges almost surely (a.s.)** if $\Pr[\limsup X_n = X = \liminf X_n] \to 1$ as $n \to \infty$. This type of convergence is also called **convergence with probability 1**. Convergence a.s. is quite a strong statement, but it is an asymptotic condition, it says nothing about finite n. Convergence a.s. implies convergence in probability but not the converse. A sufficient condition for convergence a.s. is the Borel-Cantelli theorem. (For good treatments of stochastic convergence see, for example chapter 1 of [Wei78] or chapter 4 of [Chu74]).

There are two different ways to get a set of uniform i.i.d. points on the unit square $[0,1]^d$ (they could be easily extended to the d-dimensional case):

1. The vertices are random variables X_i i.i.d. from the uniform distribution on $[0,1]^2$.
2. Use a two-dimensional point Poisson process with intensity n on $\mathbb{R}^2$ and keep the points that fall in the unit square.

Recall that a two-dimensional point Poisson process with intensity λ is a process of events in the plane such that (i) for any region of area A, the number of events in A is Poisson distributed with mean $\mu = \lambda A$, and (ii) the events in nonoverlapping regions are independent.

2 The BHH-theorem

One of the seminal papers in the area of probabilistic analysis of combinatorial optimization problems in the Euclidian d-dimensional unit cube $[0,1]^d$, Beardwood, Halton and Hammersley proved the following result [BHH59],

Theorem 1. *Let $X = \{X_i\}$ be a sequence of independent and uniformly distributed points in $[0,1]^d$. Let L_n denote the length of the optimal solution of*

the traveling salesman tour (TSP) among X. Then there exists a constant $0 < \beta(d) < \infty$ such that

$$\frac{L_n}{n^{\frac{(d-1)}{d}}} \to \beta(d) \text{ a.s. as } n \to \infty. \tag{1}$$

It is still an open problem to obtain the exact value of the constants for specific dimensions. Beardwood, Halton and Hammersley gave upper and lower bounds for certain particular d, for instance for $d = 2$ they prove: $0.44194 \leq \beta(2)/\sqrt{2} \leq 0.6508$. The best know result $0.70 \leq \beta(2) \leq 0.73$ was obtained empirically by D. Johnson (see section 2.3 of [Ste97]).

Because the value of some of the steps in the proof of the BHH theorem, let us give a sketch of the proof for the bidimensional case. The reader could look into [Ste97] for filling the details and extend the proof to d-dimensions. The basic steps in the proof of the BHH theorem are the following:

1. Show concentration around the mean value.
2. Bound (from above and below) the expected value.
3. Use (De)Poissonization to show the convergence to a constant of the mean value.

The first step for the TSP problem can be achieved using either Talagrand or Azuma inequalities, which together with Borel-Cantelli gives:

$$L_n - E[L_n] = O(\log n) \text{ a.s.} \tag{2}$$

To get a rough upper bound on the size of $E[L_n]$ for the TSP in $[0,1]^2$, it is used a **dissection technique** of the unit square. The square $[0,1]^2$ is covered with $O(n)$ squares of size $O(n^{-1/2} \times n^{-1/2})$. An easy pigeon-hole argument gives a constant c such that for any set $\{x_1, \ldots, x_n\} \subseteq [0,1]^2$,

$$\min\{|x_i - x_j| : \{x_i, x_j \in \{x_1, \ldots, x_n\}\} \leq cn^{-1/2}.$$

Therefore the length l_n of the longest TSP tour verifies the recursion $l_n \leq l_{n-1} + 2cn^{-1/2}$, and suming up the bound for the maximum gives an upper bound on the expected value. So, for some constant c_0, $E[L_n] \leq c_0 n^{1/2}$.

To get the lower bound, it is used the fact that for any set of n independent and uniformly distributed points in the unit square, there is a constant c_1 such that

$$E[\min\{|X_i - X_j| \; |\{X_i, X_j \in \{X_1, \ldots, X_n\}\}\}] \geq c_1 n - 1/2.$$

Furthermore any tour has n edges as large as the minimum distance, therefore

$$E[L_n] \geq c_1 n^{1/2}. \tag{3}$$

To extract asymptotics on $a_k = E[L_k]$ is useful to consider a continuous problem. For any set $S \subseteq \mathbb{R}^2$, let $L(S)$ denote the length of the shortest tour through S. Let Π denote a two-dimensional point Poisson process with unit intensity. We can define a new stochastic process $\{Z(t)\}$ by $Z(t) = L(\Pi[0,t]^2)$. The cardinality

of the set $\Pi[0,t]^2$ is a Poisson random variable with mean t^2. Scaling by a $1/t$ factor will made all the points fell into the unit square. Therefore the expected value of $Z(t)$ can be expressed in terms of a_k,

$$E[Z(t)] = t \sum_{k=0}^{\infty} a_k e^{-t^2} \frac{t^{2k}}{k!}.$$

From this formula it is possible to derive asymptotics for a_k. Notice that the fact that all a_k are bounded shows that Z is a continuous function of t.

The key step in the proof, is a **dissection technique** of $[0,1]^2$ that will give a geometric subadditivity expression for L. Given any $m \in \mathbb{N}$ and $t \in \mathbb{R}, t \geq 0$, let $\{Q_i\}, 1 \leq i \leq m^2$, be a partition of $[0,t]^2$ into squares of edge length t/m. We can select one point from each subsquare Q_i, and compute a shortest TSP tour on this set of points. Once we have this tour we can complete the tour gluing at each point of the tour, the optimal subtour of the corresponding square. Therefore the additional cost of the gluing process can be bound with the bound on L_{m^2} times the scaling factor, so there is a constant C such that,

$$L(\{x_1,\ldots,x_n\} \cap [0,t]^2) \leq \sum_{i=1}^{m^2} L(\{x_1,\ldots,x_n\} \cap Q_i) + Ctm, \tag{4}$$

From equation (4) we get the recursion, $E[Z(t)] \leq m^2 E[Z(t/m)] + Ctm$. Using standard algebraic techniques, together with a change of variables it is shown that there is a constant γ such that

$$\sum_{k=0}^{\infty} a_k e^{-t} t^k/k! \sim \gamma t^{1/2}. \tag{5}$$

To de-Poissonize it is needed a relation between the a_n values and the $E[L_n]$ for a Poisson random variable with mean n. The key fact is that the a_k values do not change rapidly. Notice that it takes an additional cost of $2\sqrt{2}$ to join a tour on n points with a disjoint tour on m points, to obtain a tour on the $n+m$ points. As the a_k values are all bounded, it holds $|a_n - a_k| \leq c|n-k|^{(d-1)/d}$.

If N is a Poisson random variable with mean n,

$$E[a_N] = \sum_{k=0}^{\infty} a_k e^{-n} n^k/k! \text{ and } |a_n - E[a_N]| \leq \sum_{k=0}^{\infty} |a_n - a_k| e^{-n} n^k/k!.$$

This relation together with some algebraic manipulation gives that $|a_n - E[a_N]| = O(\sqrt{n})$. Putting all together and transferring the result to the discrete problem,

$$5E[L_n]/n^{1/2} \to \gamma \text{ as } n \to \infty. \tag{6}$$

But notice that equation (5) gives the existence of β in the statement of the BHH-theorem, equation (3) gives the $0 < \beta(d) < \infty$ condition and equation (3) gives the a.s. limit.

3 Other work on the Euclidian model

The dissection technique of the unit cube into smaller disjoint subcubes, has been repeatedly used. In particular Karp [Kar77] used the dissection technique to give an algorithm to approximate the Euclidian TSP on $[0,1]^2$. Given a uniform i.i.d. set X in $[0,1]^2$, Karp's algorithm consists in dissecting $[0,1]^2$ into $n/m(n)$ squares $\{Q_i\}$, each of size $\sqrt{m(n)/n}$. Then, whp every square will contain at most $m(n)$ points. Using dynamic programming, construct in polynomial time an optimum tour in each square, so $m(n)$ should be of order $\log n$ or better $\log\log n$. Consider each tour in each Q_i as a single point p_i, and for any $1 \leq i,j \leq n/m(n)$, define the distance between p_i and p_j as the shortest Euclidian distance between any point in Q_i and any point in Q_j. Construct the minimal-length spanning tree joining all the $\{p_i\}$ points (it can be done in $O(n \log n)$ steps [SH75]). The solution to the TSP is the closed walk that traverses each subtour once and each tree edge twice. Karp proves that asymptotically, with probability 1 the algorithm tends to give near optimal solutions (it produces a tour of length within $(1+\epsilon)$ of the optimal), and with probability 1 the algorithm runs in $O(n^2 \log n)$ steps. The approximation heuristic of Karp was generalized to the d-dimensional cube in [HT82]. Some work has also been done on the **Euclidian directed TSP** on $[0,1]^2$, where the direction among any two random points in $[0,1]^2$ is choosed independently with probability $1/2$. If D_n denotes the length of the minimal Euclidian length on such a graph, Steele [Ste86] proves that asymptotically $E[D_n] \sim \gamma\sqrt{n}$ for a constant γ. It is open to prove an a.s. convergence result for the Euclidian directed TSP. As mentioned in the introduction, we shall not consider the asymmetric TSP and refer the reader to section 2.3.1 in [FM96].

The result of the BHH-theorem was extended to other combinatorial problems for the Euclidian model on $[0,1]^d$, the minimum matching [Pap78], the Steiner minimal tree [Ste81], and the minimum spanning tree [Ste88]. For further work on this last problem see [AB92] and [Pen97a]. In fact, Steele generalized the BHH-theorem for a class of combinatorial problems that could be formulated as a **subadditive Euclidian functional** F from finite subsets of $[0,1]^d$ to the nonnegative real numbers. The functional F measures the cost that the problem optimizes (length of tour, weight of the spanning tree, etc.). Define F to be a **subadditive functional** if it has the following properties:

1. $F(\emptyset) = 0$,
2. $F(cx_1 \ldots, cx_n) = cF(x_1 \ldots, x_n), \forall c > 0$,
3. $F(x_1 + y, x_2 + y, \cdots, x_n + y) = F(x_1, x_2, \cdots, x_n)\ \forall y \in [0,1]^d$,
4. **Geometric Subadditivity**

$$F(x_1 \ldots, x_n) \leq \sum_{i=1}^{m^2} F(\{x_1 \ldots, x_n\} \cap Q_i) + c_0 m, \exists c_0 > 0, \forall m, n \geq 1$$

   where Q_i is the i-th square of size $1/m \times 1/m$.
5. $F(x_1 \ldots, x_n) \leq F(x_1 \ldots, x_n, x_{n+1})$.

Notice that the geometric subadditivity is equation (4) in the proof of the BHH-Theorem. Steele proved that for each problem in this class, there exists a constant $\beta_F(d)$ (depending on F and d) such that with probability one we have

$$\lim_{n\to\infty} \frac{F_n}{n^{\frac{d-1}{d}}} \to \beta_F(d),$$

where F_n is the optimal value of the functional under consideration for the graph formed with the first points of X, (see chapter 3 of [Ste97]). It is an open problem to determine the values of β_F for any of the interesting functionals, even for the case $d = 2$.

An important set of problems motivated by the field of Computational Geometry are the following: Given a set $X = \{X_i\}$ of random points uniformly distributed on $[0, 1]^d$, we wish to find:

1. The *length of the largest nearest-neighbor links* defined by
$$Z = \max_{1\le i\le n} \min_{j\neq i} ||X_i - X_j||_2.$$
2. The *length $N_{k,n}$ of the kth. nearest graph*, in which each point is connected to its k-th nearest neighbor.
3. The *length V_n of the Voronoi diagram of the points X.*
4. The *length D_n of the Delaunay triangulation of the points X.*

For problem 1, Steele and Tierney [ST86] gave limiting distributions for $Z_n = Z(X_1, \ldots, X_n)$. If the sample Z is drawn uniformly from a d-dimensional torus, for $d \geq 3$ the limit distribution in this case differs from the case where the space X is drawn from $[0, 1]^d, d \geq 3$, due to the the boundary effects in the cube. This result should be taken into account by those who do simulation involving the computation of the nearest-neighbor procedure, for three or more dimensions they should work on the torus instead of the cube. The study of asymptotic dominance of the boundary with respect to the limit distribution of Z_n was continued in [DH89], where they reinforced the results obtained in [ST86].

Avram and Bertsimas [AB93] studied problems 2,3 and 4 and use a Poisson process with intensity n to get the set X of points uniformly distributed in $[0, 1]^2$. It is known that if L denotes the optimal length for each of the problems,

$$\lim_{n\to\infty} \frac{E[L_n]}{n^{1/2}} \to \beta,$$

where in this case, the constant β for each problem is explicitly known [Mil70]. Avram and Bertsimas prove for each problem a central limit theorems,

$$\lim_{n\to\infty} \Pr\left[\frac{L_n - E[L_n]}{\sigma[L_n]} \leq x\right] = \Phi(x),$$

where $\Phi(x)$ is the Normal distribution function. They also prove that the rate of convergence in each problem is $O(\frac{(\log n)^{1+3/4}}{n^{1/4}})$.

4 Random Geometric Graphs

Let us consider the following model of geometric graph: Given $n \in \mathbb{N}$ and $r \in [0,1]$ define the **random geometric graph** $G_n(r)$ by selecting a set of points uniformly from $[0,1]^2$ to form the vertex set $X = \{X_1, X_2, \cdots, X_n\}$. So each vertex X_i is a random variable i.i.d. Define the edge set $E_r = \{(X_i, X_j) | \, ||X_i, X_j||_\infty \leq r\}$.

Notice we are using the L^∞ norm on the unit square $[0,1]^2$, that if $X_i = (x_i, y_i)$ and $X_j = (x_j, y_j)$ the L^∞ distance is defined by $||X_i, X_j||_\infty = \max\{(x_i - x_j), (y_i - y_j)\}$.

Depending on the specific values of n and r, the $G_n(r)$ could be dense or sparse, connected or not connected, etc. Moreover, some of the techniques used for the previous structures doesn't seem work for $G_n(r)$ $(r < 1)$ (see section 6). The following result is given in [AR97a],

Lemma 1. *Given $G_n(r)$, let* $\Pr[r]$ *denote the probability that any two vertices X_i, X_j from the vertex set X form an edge (i.e. $||X_i, X_j||_\infty \leq r$). Then,*

$$\Pr[r] = (2r - r^2)^2.$$

Define the random variable $\kappa_n(r) = |E_r|$ then,

$$\kappa_n(r) = \binom{n}{2} \Pr[r].$$

In fact,

$$\frac{\kappa_n(r)}{\binom{n}{2}} \to \Pr[r] \quad \text{a.s.}$$

Moreover Apple and Russo study the ratio of convergence when both n and $r(n)$ evolve. The basic condition that the sequence $\{r_n\}$ must satisfy is that as $n \to \infty$,

$$\frac{n r_n^2}{\log n} \to c \tag{7}$$

for a constant $c \in (0, \infty]$.

Notice that equation (7) imposes a threshold condition $r = \Omega(\sqrt{\frac{\log n}{n}})$ on the evolution of r as n evolves to ∞. The reason for this threshold is that it keeps $G_n(r)$ connected. An important issue in the $G_n(r)$ graphs is the connectivity. Define the connectivity distance c_n of a random geometric graph by

$$c_n = \inf\{r | G_n(r) \text{is connected}\}.$$

Apple and Russo prove that asymptotically

$$c_n = O\left(\sqrt{\frac{\log n}{n}}\right) \quad \text{a.s.}$$

Penrose [Pen97b], generalized the connectivity result to any metric $L^p, 2 \leq p \leq \infty$, proving that asymptotically, with high probability, if one starts with single vertices and adds the corresponding edges as r increases, the resulting graph becomes $(k+1)$ connected at the moment it achieves a minimum degree of $k+1$. Recall than in the Erdös– Rényi model for random graphs $G_{n,p}$ the connectivity threshold is $\log n/n$ [Bol85]. Therefore, both models have similar behavior for connectivity, but while in the $G_{n,p}$ model, the edges are independent, the edges in the $G_n(r)$ can have non-zero correlations.

Under the threshold condition of equation (7), Appel an Russo give almost sure asymptotic rates of convergence (divergence) for several graph parameters. For example, they prove that as $n \to \infty$, the rate at which the **minimum vertex degree** $\Delta_n(r_n)$ of $G_n(r_n)$ diverges, is that of any particular vertex in $G_n(r_n)$.

For a related parameter, let $\omega_n(r)$ denote the **clique number** of $G_n(x)$. To get a bound on $\omega_n(r)$ once more we use dissection. Let $l \in \mathbb{N}$ be such that $l < 1/r < l+1$. Dissect $[0,1]^2$ in $(l+1)^2$ squares $\{Q_i\}$, each Q_i of side at most r, therefore all the points inside of any Q_i belong to a common clique in $G_n(x)$, which implies that $n \leq (l+1)^2 \omega_n(r)$ so we get the bound,

$$\omega_n(r) \geq \frac{nr^2}{(1+1/l)^2}.$$

Other related parameter is the **chromatic number** $\chi_n(r)$ of $G_n(x)$. From standard graph theory it is easy to see that $\omega_n(r) \leq \chi_n(r) \leq \Delta_n(r) + 1$ and $\Delta_n(r) \leq \omega_n(2r) - 1$. Using these inequalities, Apple and Russo gave theorems on the rate of convergence of $\omega_n(r)$ and $\chi_n(r)$ when $\{r_n\}$ satisfies equation (7). They also give asymptotic rates of growth for the **independence number** $\beta_n(r)$ of $G_n(x)$, under the assumption that $\{r_n\}$ satisfies equation (7). To prove this statement, they need an upper bound on $\beta_n(r)$. To get it, they use dissection. Let $\{X_{i_1}, \cdots, X_{i_m}\}$ be an independent set (in the graph theoretical sense) of a random graph in $G_n(r)$. Notice that the set of *open squares* $\{O_{i_j}\}$ with side r and centered at X_{i_j} must be disjoint. Moreover the area of each O_i is r^2. Using a pigeon-hole argument we can conclude that $m \leq (1+1/r)^2$. This must be true for the largest independent set, so for all n and r $\beta_n(r) \leq (1+1/r)^2$.

In a sequel paper, Appel and Russo [AR97b] prove that under equation 7, the **minimum degree** $\delta_n(r_n)$ of $G_n(r_n)$ has an asymptotic convergence of $\Omega(\log n)$.

5 The MinLa Problem

Given an undirected graph $G = (V, E)$ with $|V| = n$, a *layout* of G is a one-to-one function $\varphi : V \to [n]$. The *minimum linear arrangement* problem, from now on denoted MINLA , is a combinatorial optimization problem formulated as follows: given a graph $G = (V, E)$, find a layout φ of G that minimizes the cost

$$c_\varphi = \sum_{uv \in E} |\varphi(u) - \varphi(v)|.$$

MINLA is an interesting and appealing **NP**-hard problem with several different applications in Computer Science and Biology [DPSS98]. However, there exist exact polynomial time solutions for some particular kinds of graphs. The lack of efficient exact algorithms for general graphs has given rise to the possibility of finding approximation algorithms. An approximation scheme of MINLA for dense graphs was presented in [FK96]. Moreover, a $O(\log n)$ approximation for general graphs and a $O(\log \log n)$ approximation for planar graphs is proposed in [RR98]. An study on the $G_{n,p}$ model for sparse graphs is given in [DPST98]. An experimental analysis of several heuristics for this problem is given in [Pet98], where it is observed that approximating geometric random graphs is harder than approximating standard random graphs.

Let us consider geometric graphs $G_n(r)$ when $r = \sqrt{c \log n/n}$ for some constant $c \in \mathbb{R}^+$. As we shaw in the previous section, whp such a graph is connected and its expected number of edges is $\Theta(n \log n)$. Moreover, for a particular random graph, the real value of the number of edges is concentrated around the expectation. Random geometric graphs are not planar in general, therefore we can approximate the MINLA on these graphs within $O(\log n)$ using the algorithm from [RR98]. However we can get weaker approximability results but using much more practical methods.

Let us consider the MINLA problem for such a graph. There are $n!$ different layouts for $G_n(r)$. We can define the **average linear arrangement** cost as the normalized sum of the costs of all layouts. It is well known that given any graph with n vertices, the average length of an edge over all layouts is $(n+1)/3$. Thus we get,

Lemma 2. *The expected value of the average linear arrangement cost for a graph $G \in G_n(r)$ is $\Theta(n^2 \log n)$.*

In the $G_{n,p}$ model the average over all layouts is of the same magnitude as the minimum [DPST98]. However this does not hold anymore for random geometric graphs, as we will see that the minimum value is of smaller magnitude.

We first derive a lower bound for the minimum linear arrangement that holds for almost all random geometric graphs and then analyze two heuristics to obtain an upper bound.

The lower bound. Consider $[0,1]^2$and dissect it in $\alpha n/\log n$ disjoint squares each of size $\sqrt{\frac{\alpha n}{\log n}} \times \sqrt{\frac{\alpha n}{\log n}}$ for some fixed (but arbitrary) constant α. A pigeon-hole type of argument will give us the following result,

Lemma 3. *Every square contains at least one vertex and at most $\alpha \log n$ vertices whp.*

Using the first part of this lemma we can obtain a lower bound:

Theorem 2. *There exists a constant c_1 such that whp, for any layout φ,*

$$c_1(n/\log n)^{3/2} \leq c_\varphi.$$

Proof. Conditioning on the event of lemma 3, after dissecting the square $[0,1]^2$, we have one node in each square. Taking $\alpha = r/2$ the only possible connections correspond to neighboring cells. Therefore whp our graph contains a squared mesh with $\alpha n/\log n$ points. In [MD86] and [MP80] it is shown that the optimum linear arrangement for a mesh $n \times n$ has cost $0.868n^3 + O(n^2)$. We can apply the known optimal value for our mesh to obtain the claimed lower bound.

To obtain upper bounds we consider two natural heuristics for the MINLA problem on geometric graphs. These are the *dissection* and the *projection* heuristics.

The dissection heuristic. Given a geometric graph in $G_n(r)$,

1. Dissect $[0,1]^2$ into $\alpha(n/\log n)$ disjoint squares each of size $\sqrt{\frac{\alpha n}{\log n}} \times \sqrt{\frac{\alpha n}{\log n}}$.
2. Assign to all vertices in the same square consecutive numbers. By our selection of α, all vertices in the same subsquare are connected.
3. Sort the squares following the up to down and left to right fashion and follow this order to assign numbers to vertices.

Let $D_{G_n(r)}$ denote the cost of the layout obtained using dissection.

Theorem 3. *There exist a constant c_2 such that whp,*

$$D_{G_n(r)} \leq (c_2 n \log n)^{3/2}.$$

Proof. To prove the upper bound, given $G_n(r)$ and a dissection as before we construct a new graph G^u by adding extra vertices until all squares have $\log n$ vertices inside. The edges will be all possible connections between vertices in the same squares and all the connections of all the vertices in a square with the vertices in neighbor squares. Clearly, this construction is not always possible, it may be the case that a square has more than $\log n$ vertices, but by lemma 3, we can assure that whp G^u is a supergraph of $G_n(r)$.

Applying the dissection heuristic to G^u we get the upper bound.

The projection heuristic. The projection heuristic is also quite natural: the projection of each vertex into the x-axis induces a natural linear ordering of the vertices. Another way to see this heuristic is to slide a vertical line starting from position 0 to 1, and number vertices in the order the line touches them.

Let $P_{G_n(r)}$ denote the cost of the layout obtained using the projection heuristic.

Theorem 4. *There exist a constant c_3 such that whp,*

$$P_{G_n(r)} \leq (c_3 n \log n)^{3/2}.$$

Proof. As the points are distributed uniformly, the expected length (on the projected layout) of any edge is n times the distance of the two end points projection. Furthermore, the real distance is close to the average whp.

Therefore, as the distance between the projections is bounded by the value of r we have that whp $P_{G_n(r)} \leq nmr$. Substituting with the right parameters we get our bound.

Combining with the lower bound, both heuristics provide a $O(\log^3 n)$ approximation algorithm to MINLA on almost all random geometric graphs. These algorithms have an approximation ratio worse than the error bound of $O(\log n)$ in [RR98] although the algorithm described by Rao and Richa is extremely ingenious, it has the drawback of using the ellipsoid method and, thus it is not feasible for large graphs. On the other hand, the algorithms described here are simple to apply.

6 Open Problems

Concerning the MINLA problem and analogously with other problems, there are several natural questions to ask:

- What is the right order of magnitude of the mean value of the minimum linear arrangement cost over all graphs in $G_n(r)$?
- Is the minimum layout cost on $G_n(r)$ concentrated or not around this mean?
- Finally, is there some BHH type result for this measure? In this case, one may ask which is the limiting constant.

There are some subtle impediments when trying to answer these questions. Let us point out the main difficulties:

- Although when we partition the graph into pieces we can compute a layout from optimal layouts on the disjoint pieces, the additional cost is not a linear function of the piece size. Furthermore, the piece size must be related under some conditions with the selected value of r.
- The measure we are considering is *discrete*: Although the distance in the geometric graph has some good properties, we measure integer distances. Therefore, a scaling of the values will not scale the minimum cost. The only property that we can show is that if the transformation (scaling) creates a subgraph (supergraph) of our random graph then the cost in the new scaled graph is upper (lower) bounded by the cost in the original graph.
- We loose monotonicity: notice that the sequence $\{r(n)\}$ that we choose to get random sparse geometric graphs is decreasing. Therefore adding a point may change the graph dramatically, so that the new graph is neither a supergraph nor a subgraph of the previous one.

On the other hand, random geometric graphs seems to be formed by small clusters, that form a clique expanded in the same fashion. Therefore, it seems that they could also be related to some cases of bounded tree-width graphs. It is an open problem to analyze the expected tree-with of such graphs for different sequences of radio.

References

[AB92] F. Avram and D. Bertsimas. The minimum spanning tree constant in geometric probability and under the independent model: A unified approach. *The Annals of Applied Probability.*, 2:113–130, 1992.

[AB93] F. Avram and D. Bertsimas. On the central limit theorems in geometrical probability. *The Annals of Applied Probability.*, 3:1033–1046, 1993.

[AR97a] M.J. Appel and R.P. Russo. The maximum vertex degree of a graph on uniform points in $[0,1]^d$. *Adv. Applied Probability.*, 29:567–581, 1997.

[AR97b] M.J. Appel and R.P. Russo. The minimum vertex degree of a graph on uniform points in $[0,1]^d$. *Adv. Applied Probability.*, 29:582–594, 1997.

[ASE92] N. Alon, J.H. Spencer, and P. Erdös. *The probabilistic method.* Wiley-Interscience, New York, 1992.

[BHH59] J. Beardwood, J. Halton, and J.M. Hammersley. The shortest path through many points. *Proceedings of the Cambridge Philos. Society.*, 55:299–327, 1959.

[Bol85] B. Bollobas. *Random Graphs.* Academic Press, London, 1985.

[Chu74] K. Chung. *A course in Probability Theory.* Academic Press, New York, 1974.

[DH89] H. Dette and N. Henze. The limit distribution of the largest nearest–neighbour link in the unit d-cube.. *Journal Applied Probability.*, 26:67–80, 1989.

[DPSS98] J. Díaz, J. Petit i Silvestre, M. Serna, and P. Spirakis. Heuristics for the MinLA Problem: An Empirical and Theoretical Analysis. Technical Report LSI-98-42-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 1998.

[DPST98] J. Díaz, J. Petit, M. Serna, and L. Trevisan. Approximating layout problems on random sparse graphs. Technical Report LSI 98-44-R, Universitat Politècnica de Catalunya, 1998.

[ER59] P. Erdos and A. Renyi. On random graphs-i. *Publicationes Matematicae*, 6:290–297, 1959.

[ER60] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungarian Academy of Sciences.*, 5:17–61, 1960.

[FK96] A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *37th. FOCS*, pages 62–71, 1996.

[FM96] A Frieze and C. McDiarmid. Algorithmic theory of random graph. Technical report, Department of Mathematics. Carnegie Mellon University., 1996.

[HT82] J.H. Halton and R. Terada. A fast algorithm for the euclidian travelling-salesman problem, optimal with probability one. *SIAM Journal on Computing*, 11:28–46, 1982.

[Kar77] R.M. Karp. Probabilistic analysis of partitioning algorithms for the travelling-salesman problem in the plane. *Mathematics of Operation Research.*, 2:209–224, 1977.

[MD86] G. Mitchison and R. Durbin. Optimal numberings of an $n \times n$ array. *SIAM J. on Alg. Disc. Math.*, 7(4):571–582, 1986.

[Mil70] R. Miles. On the homogeneous planar poisson point process. *Mat. Bioscience*, 6:85–125, 1970.

[MP80] D.O. Muradyan and T.E. Piliposjan. Minimal numberings of vertices of a rectangular lattice. *Akad. Nauk. Armjan. SRR*, 1(70):21–27, 1980. In Russian.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Pap78] C.H. Papadimitriou. The probabilistic analysis of matching heuristics. In *Proc. 15th. Annual Conference on Comm. Control Computing.*, pages 368–378, 1978.

[Pen97a] M. Penrose. The longest edge of the random minimal spanning tree. *The Annals of Applied Probability.*, 7:340–361, 1997.

[Pen97b] M. Penrose. On the k-connectivity for a geometric random graph. Technical report, Department of Mathematical Science. University of Durham., 1997.

[Pet98] J. Petit i Silvestre. Approximation Heuristics and Benchmarkings for the MinLA Problem. In R. Battiti and A. Bertossi, editors, *Alex '98 — Building bridges between theory and applications*, 1998.

[RR98] S. Rao and A. W. Richa. New Approximation Techniques for Some Ordering Problems. In *9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 211–218, 1998.

[SH75] M.I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th IEEE Annual Symposium on Foundations of Computer Science*, pages 224–233, 1975.

[ST86] J.M. Steele and L. Tierney. Boundary domination and the distribution of the largest nearest-neighbpr link in higher dimensions. *Journal Applied Probability.*, 23:524–528, 1986.

[Ste81] J.M. Steele. Subadditive euclidean functional and nonlinear growth in geometric probability. *Annals of Probability.*, 9:365–376, 1981.

[Ste86] J.M. Steele. Probabilistic algorithm for the directed traveling salesman problem. *Math. od Operation Research.*, 11:343–350, 1986.

[Ste88] J.M. Steele. Growth rates of euclidean minimal spanning trees with power weighted edges. *Annals of Probability.*, 16:1767–1787, 1988.

[Ste97] J.M. Steele. *Probability theory and Combinatorial Optimization.* SIAM CBMS-NSF Regional Conference Series in Applied Mathematics., 1997.

[Wei78] B. Weide. *Statistical Methods in Algorithmic Design and Analysis.* Phd thesis, Department of Computer Science, Carnegie-Mellon University, 1978.

A Role of Constraint in Self-Organization

Carlos Domingo[1]*, Osamu Watanabe[2], Tadashi Yamazaki[2]

[1] Department de LSI, Universitat Politècnica de Catalunya Campus Nord, Mòdul C5, 08034-Barcelona, Spain carlos@lsi.upc.es
[2] Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo 152-8552, Japan {watanabe,tyam}@is.titech.ac.jp

Abstract. In this paper we study a neural network model of self-organization. This model uses a variation of a Hebb rule for updating its synaptic weights, and surely converges to the equilibrium status. The key point of the convergence is the update rule that constrains the total synaptic weight and this seems to make the model stable. We investigate the role of the constraint and show that it is the constraint that makes the model stable. For analyzing this setting, we propose a simple probabilistic game that abstracts the neural network and the self-organization process. Then, we investigate the characteristics of this game, namely, the probability that the game becomes stable and the number of the steps it takes.

1 Introduction

How does the brain establish connections between neurons? This question has been one of the important issues in Neuroscience, and theoretical researchers have proposed various models for self-organization mechanisms of the brain. In many of these models, competitive learning, or more specifically, competitive variants of a Hebb rule have been used as a key principle. In this paper, we study one property of such competitive Hebb rules.

As one typical example of self-organization, "orientation selectivity" [WH63] has been studied intensively. In the primary visual cortex (area17) of cats, there is some group of neurons that strongly reacts to the presentation of light bars of a certain orientation, which we call *orientation selectivity*. An interesting point is that in a very early stage after birth, every neuron reacts to all bars of every orientation. This indicates that orientation selectivity is obtained after birth; that is, each neuron selects one preferred orientation among all orientations. To explain the development of orientation selectivity, a considerable number of mathematical models have been investigated; see, e.g., [Swi96]. Although these models may look quite different, most of them use, as a principal rule for modifying synaptic strength, a competitive variant of a Hebb rule, which is essentially the same as the rule proposed in the pioneer paper of von der Malsburg [Mal73],

* Supported by ESPRIT LTR Project no. 20244 - ALCOM-IT and CICYT Project TIC97-1475-CE.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 307-318, 1998.

the paper that first gave a mathematical model for the development of orientation selectivity.

A *Hebb rule* is a simple rule for updating, e.g., the weight of connection between two neurons. The rule just says that the connection between two neurons is strengthened if they both become active simultaneously. This rule has been used widely for neural network learning. Von der Malsburg constrained this updating rule so that the total connection weight of one neuron is kept under some bound. In this paper, we call this variation of a Hebb rule a *constrained Hebb rule*. He showed through computer experiments that orientation selectivity is surely developed with his constrained Hebb rule.

Since the work of von der Malsburg, many models have been proposed, and some have been theoretically analyzed in depth; see, e.g., [Tan90]. For example, a feature of various constrained Hebb rules as a learning mechanism has been discussed in [MM94]. Yet, the question of why orientation selectivity is obtained by following a constrained Hebb rule has not been addressed. Note that the development of orientation selectivity is different from ordinary learning in the sense that a neuron (or, a group of neurons) establishes a preference to one particular orientation from given (more or less) uniformly random orientation stimuli. In this paper, we discuss why and how one feature from equally good features is selected with a constrained Hebb rule.

In order to simplify our analysis, we propose a simple probabilistic game called "monopolist game" for abstracting Hebb rules. In the monopolist game, an updating rule corresponds to game's rule and the selectivity is interpreted as that a single winner of a game — monopolist — emerges. Then, we prove that a monopolist emerges with probability one in games following a von der Malsburg type rule. On the other hand, we showed theoretical evidence supporting that (i) the chance of having a monopolist is low without any constraint, and (ii) a monopolist emerges even under a rule with a weaker constraint. These results indicate the importance of constraint in Hebb rules (or, more generally, competition in learning) to select one feature from equally good features.

We also analyzed how fast a monopolist emerges in games following a von der Malsburg type rule. This analysis can be used, in future, to estimate the convergence speed of constrained Hebb rules. (In this extended abstract, some of the proofs are omitted. See [DWY98] for those proofs.)

2 Von der Malsburg's Model and Monopolist Game

Here we first explain briefly the model considered by von der Malsburg. (Von der Malsburg studied the selectivity for a set of neurons, but here we only consider its basic component.)

Neural Network Structure

We consider two layer neural network. In particular, we discuss here the orientation selectivity for one neuron, and thus, we assume that there is only one output cell. On the other hand, the input layer consists of 19 input cells that

are (supposed to be) arranged in a hexagon like the ones in Figure 1. We use i for indicating the ith input cell, and IN for the set of all input cells.

Stimuli and Firing Rule

We use 9 stimuli with different orientations (Figure 2), which are given to the network randomly. Here $\bullet$ indicates an input cell that gets input 1, and $\circ$ indicates an input cell that gets input 0.

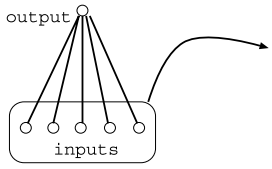


Fig. 1. Neural network model.

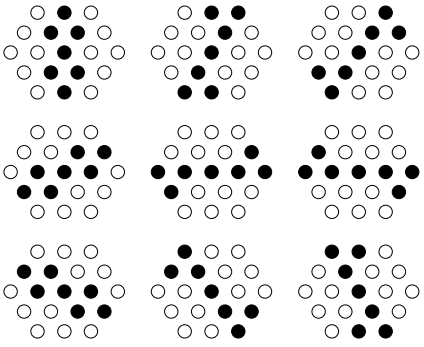


Fig. 2. Nine stimuli.

We use a_i to denote input value (either 0 or 1) to the ith input cell. The output value V is computed as $V = \mathrm{Th}_p\left(\sum_{i \in IN} w_i a_i\right)$, where w_i is the current synaptic strength between the output cell and the ith input cell. $\mathrm{Th}_p(x)$ is a threshold function that gives $x - p$ if $x > p$ and 0 otherwise, where p is given as a parameter.

Updating Rule

Initially, each weight w_i is set to some random number between 0 to some constant. The weights are updated each time according to the following variation of a Hebb rule, which we call the *constrained Hebb rule* (of von der Malsburg).

$$w_i' \;=\; w_i + c_{\mathrm{inc}} a_i V, \quad \text{and} \quad w_i \;=\; w_i' \times \left(W_0 / \sum_{k \in IN} w_k' \right).$$

Where c_{inc} (which is called a *growth rate*) and W_0 (*total weight bound*) are constants given as parameters. The first formula may be considered as the original Hebb rule; on the other hand, the second one is introduced in order to keep the total weight within W_0. (In fact, it is kept as W_0.)

With this setting, von der Malsburg demonstrated that the selectivity is developed through computer simulations. Thus, it seems likely that some selection occurs even from uniformly random examples, and that the constraint of the von der Malsburg's rule is a key for such a selection. In this paper we would

like to study this feature of the constrained Hebb rule. For this, we further simplify von der Malsburg's computation model, and propose the following simple probabilistic game.

Monopolist Game
Basic Rule: Consider a finite number of players. Initially they are given the same amount of money. The game goes step by step and, at each step, one of the players wins where all the players have the same winning probability. The winner gets some amount of money, while the others lose some.
Details: A player who loses all his money is called *bankrupt*. Once a player becomes bankrupt, he cannot get any amount of money, though he can still win with the same probability. (See below for the motivation.)
Goal: The game terminates if all but one player become bankrupt. If the survived player keeps enough money at that point, then he is called a *monopolist*. We call a situation when a monopolist appears a *monopoly*.

Notations. We use n and n' to denote the number of initial players and that of the remaining (not being bankrupt) players at some fixed step, and use i, $1 \leq i \leq n$, to denote players' indices. Throughout this paper, each player's wealth is simply called a *weight*, and let w_i denote the i-th player's current weight. Let I and W_0 respectively denote the initial weight of each player and the total amount of initial weights; that is, $W_0 = nI$.

The connection of this game with von der Malsburg's computation model is clear; each player's weight corresponds to the total synaptic strength between the output cell and a set of input cells corresponding to one type of stimulus, and the emergence of a monopolist means that the network develops preference to one orientation. From this correspondence, it is natural to require that even a bankrupt player can win with the same probability $1/n$, which reflects the fact that the probability of a stimulus of each orientation appears is the same no matter how neural connections are organized.

An updating rule of players' weights corresponds to a rule of changing synaptic strength in the network. Here we can state updating rules in the following way. (In the following, let i_0 denote the player who wins at the current step.)

$$w_i = \begin{cases} w_i + f_{\mathrm{inc}} - f_{\mathrm{dec}}, & \text{if } i = i_0, \text{ and} \\ w_i - f_{\mathrm{dec}}, & \text{otherwise.} \end{cases}$$

Here f_{inc} and f_{dec} are the amount of increment and decrement at each step respectively, and one type of monopolist game is specified by defining f_{inc} and f_{dec}. In the following, we assume that these values are determined from w_i, w_{i_0}, n, and n'. From the relation to von der Malsburg's computation model, we require that both f_{inc} and f_{dec} are 0 if $w_i = 0$; that is, once a player loses all money, he stays forever in the 0 weight state. (In the following, we will omit stating this requirement explicitly.)

Now we consider the rule that corresponds to the constrained Hebb rule of von der Malsburg's rule. For constant $c_{\rm inc}$ it is defined as follows:

$$f_{\rm inc} = c_{\rm inc}, \quad \text{and} \quad f_{\rm dec} = c_{\rm inc}/n'. \tag{1}$$

(Recall that n' is the number of currently remaining players.)

Note that with this rule, the total amount of wealth is kept constant. Thus, in this sense, it corresponds to von der Malsburg's rule, and we call it *constrained rule*. Note that we may also consider a similar rule such that $f_{\rm inc}$ is not constant but proportional to w_i. (Similarly, $f_{\rm dec}$ is also proportional to w_i.) This rule might be closer to the original von der Malsburg's rule. This difference is, however, not essential for discussing the probability of having a monopolist, i.e., for our discussion in Section 3. On the other hand, there is a significant difference in convergence speed; but roughly speaking, the difference disappears if we take the log of weight. Thus, we will discuss with the above simpler rule.

3 Importance of Competition

Here we compare three different updating rules for monopolist game, and show that constraint in the rule is an important characteristic to derive a monopolist. From this, we could infer that some sort of constraint, (or, more generally, competition) is important in learning rules for selecting one feature among a set of features through a random process.

In the following, we consider the following three updating rules: (1) constrained rule, (2) local rule, and (3) semi local rule. Below we define these rules (except (1) that has been defined in the previous section) and discuss the probability P_* that a monopolist emerges.

Constrained Rule

We show that under constrained rule, P_* is 1, that is, a monopolist emerges with probability 1.

A monopolist game in general expressed by an one-dimensional random walk. More precisely, for any i, we can express the player i's wealth w_i as the following random walk.

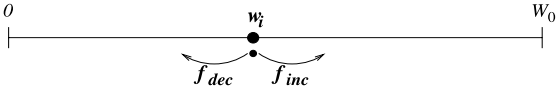


Fig. 3. One-dimensional random walk.

Note that the particle (i.e., the weight w_i) moves to the left (resp., to the right) with probability $1 - 1/n$ (resp., $1/n$). The left (resp., right) end of the interval means that the player i becomes bankrupt (resp., a monopolist). Thus, these two ends are absorbing walls.

In a monopolist game under constrained rule with $n = 2$, we have $f_{\rm inc} = c_{\rm inc}/2$ and $f_{\rm dec} = c_{\rm inc}/2$. Hence, the above random walk becomes standard one

(see, e.g., [Fel68]), and it is well-known that the particle in such a standard random walk goes to one of the absorbing walls in finite amount of steps with probability 1. This proves that $P_* = 1$ when $n = 2$. Then by induction, we can prove $P_* = 1$ when $n > 2$; thus, we have the following theorem.

Theorem 1 . *Under constrained rule, a monopolist emerges (in finite amount of steps) with probability 1.*

Local Rule

In the constrained rule, for computing $f_{\rm dec}$, we need the number of remaining players; that is, weights cannot be updated locally. In general, in order to be competitive, an updating rule must not be local. Thus, to see the importance of competition, we consider here the following purely local updating rule.

$$f_{\rm inc} = c_{\rm inc}, \text{ and } f_{\rm dec} = c_{\rm dec}. \tag{2}$$

Notice that for this local rule (and the next semi local rule), the notion of monopolist is less clear than in the case of a game with a constrained rule, because the notion of "enough amount of money" is not clear. Here we simply consider it as $W_0/2$, a half of the total initial weight. That is, we regard a single survivor as a monopolist if his weight is more than $W_0/2$; hence, P_* is the probability that the game reaches to the state where $w_{i_1} \geq W_0/2$ for some i_1 and $w_i = 0$ for the others i.

We first discuss one feature of this updating rule. In the following, let us fix $c_{\rm dec} = 1$. Our computer experiments show that the probability of having a single survivor (in a reasonable amount of steps) drops rapidly when $c_{\rm inc} \geq n + 1$. The reason is clear from the following fact.

Theorem 2 . *Fix $c_{\rm dec}$ to be one, and consider one player's weight. For any t, it increases, by $t\left(\frac{c_{\rm inc}}{n} - 1\right)$ on average, after t steps.*

Thus, if $c_{\rm inc} > n$, then it is quite likely that all players increase their weights, and thus no bankrupt appears in the game. On the other hand, if $c_{\rm inc} < n$, then every player dies quickly, and hence, no monopolist occurs even though someone may become the last player. This means that the most crucial case is the case $c_{\rm inc} = n$. Next we discuss P_* for such a case.

Recall that P_* is the probability that, at some point in the game, all but one players become bankrupt and that the survivor has weight $\geq W_0/2$. Since it is difficult to estimate P_* directly, we analyze the following probability P'_* instead of P_*: P'_* is the probability that at least one player's weight reaches to $W_0/2$ and no more than two players have weight larger than a sufficiently large value, say, kW_0 for some $k > 0$. Notice that if a monopolist emerges at some point, then clearly, someone needs to reach $W_0/2$ in the game. Furthermore, it is unlikely that two players reach to kW_0 and one of them become bankrupt afterwards. Thus, we may regard P'_* as an upper bound of P_*. For this P'_*, we have the following bound.

Theorem 3. *For any W_1 and sufficiently large W_2, we have*

$$P'_* \;<\; \Big(1-\frac{I}{W_2+n}\Big)^n + \frac{nI}{W_2}\Big(1-\frac{I}{W_2+n}\Big)^{n-1} - \Big(1-\frac{I}{W_1}\Big)^n.$$

For example, by taking W_1 and W_2 as $\frac{nI}{2}$ and knI respectively, we have

$$P'_* \;<\; e^{-I/(kI+1)} + e^{-I(n-1)/(kI+1)n}/k - e^{-2} \;\approx\; (1+1/k)e^{-1/k} - e^{-2},$$

which is less than 0.6 if $k = 1$. On the other hand, our computer experiments show that P_* is less than 0.5 for various sets of parameters.

Semi Local Rule

As a third updating rule, we consider somewhat mixture of the above two rules. It keeps a certain amount of locality, but it still has some constraint. This rule is defined as follows.

$$f_{\rm inc} \;=\; \min(c_{\rm inc}, W_0 - \sum_j w_j), \;\; \text{and} \;\; f_{\rm dec} \;=\; c_{\rm dec}. \tag{3}$$

That is, we want to keep the total weight smaller than W_0, where W_0 is the total initial weight. Thus, a winner can gain $c_{\rm inc}$ (in net, $c_{\rm inc} - c_{\rm dec}$) if there is some room to increase its weight. In this case, only the winner needs to know the current total weight, or the amount of room to the limit W_0, and the other players can update its weight locally.

Our computer experiments show that the probability P_* that a monopolist emerges is fairly large if $c_{\rm inc}$ is large enough, say $c_{\rm inc} \geq 2n$. On the other hand, P_* gets small when $c_{\rm inc}$ is small, which is explained in the same way as local rule. Although we have not been able to prove that P_* is large for sufficiently large $c_{\rm inc}$, we can give some analytical result supporting it.

Here instead of analyzing P_*, we estimate (i) the average number of steps until all but one players become bankrupt, and (ii) the average number of steps until the total weight (which is initially W_0) becomes $W_0/2$. Let $T_{n\to 1}$ and $T_{W_0\to W_0/2}$ denote the former and the latter numbers respectively. We prove below that $T_{n\to 1}$ is smaller than $T_{W_0\to W_0/2}$ if W_0 is large enough. This means that it is likely that at the time when all but one players become bankrupt, the total weight, which is the same as the survivor's weight, is larger than $W_0/2$, that is, the survivor is a monopolist.

Theorem 4. *Fix again $c_{\rm dec}$ to be one. For large n, if $I \geq (\ln 6)n(n-2)$ and $c_{\rm inc} \geq 2n$, then we have $T_{W_0\to W_0/2} > T_{n\to 1}$.*

4 Efficiency Analysis

In this section we discuss how fast a monopolist emerges in games with constrained rule. We estimate an upper bound on the average number of steps needed for monopoly to emerge, and we give some justification (not a rigorous proof) supporting that it is $O(n^3 \ln n (I/c_{\rm inc})^2)$.

We start with some definitions and notations that are used through the section. Here we modify our monopolist game and define a variation of it. Let game_0 denote the original monopolist game and let game_1 denote a variant of game_0 in which no bankrupt player can win. That is, in game_1, the winning probability of the remaining players is $1/n'$ instead of $1/n$. As we will see game_1 is useful for induction and it is easier to analyze.

These two game types are defined on different probability spaces. Let us define them more precisely. For all two game types, (the execution of) a game is specified by a *game sequence*, i.e., a string from $\{1,\ldots,n\}^*$ that defines a history of winners. (Precisely speaking, we also need to consider infinite strings; but as we will see below, we may ignore infinite strings.) We say that a game sequence x *kills* a player i if w_i becomes 0 (or, negative) in the game following x just after the end of x, and we say that x *derives a monopolist* if the second last player is killed and monopoly emerges just after x. We say that a game sequence x is *valid* (resp., *strongly valid*) if it derives a monopolist and no prefix of it derives a monopolist (resp., x contains no indices of previously killed players). Note that the meaning of these notions may vary depending on game types. Now for any n, let X_n (resp., Y_n) be the set of game sequences for n player games that are strongly valid w.r.t. game_0 (resp., valid w.r.t. game_1). For each x in X_n, its probability $\Pr\{x\}$ is $n^{-|x|}$. On the other hand, the probability $\widetilde{\Pr}\{y\}$ of $y \in Y_n$ depends on the number of remaining players, and it is rather complicated. (We omit specifying $\widetilde{\Pr}\{y\}$ because it is not important for our discussion.) Note that X_n and Y_n are all prefix free. Also it is not hard to show that $\Pr\{X_n\}$ and $\widetilde{\Pr}\{Y_n\}$ are one. (For example, $\Pr\{X_n\} = 1$ follows from Theorem 1 .) Therefore, we may regard X_n and Y_n as the probability spaces of the corresponding games, and we do not have to worry about infinite strings.

We denote by $T(n, I_1, \ldots, I_n)$ (resp., $T_1(n, I_1, \ldots, I_n)$) the number of steps needed until monopoly emerges in game_0 (resp, game_1) with n players and initial weight $I_1, \ldots, I_n$. When all the weights are equal, we use the simpler notation $T(n, I)$. Our goal is to get some upper bound on $\mathrm{E}[T(n, I)]$. But instead, we will analyze an upper bound on $\mathrm{E}[T_1(n, I)]$, which gives us an upper bound on $\mathrm{E}[T(n, I)]$, as the following lemma guarantees.

Lemma 5 . *There exists c_1 such that for any sufficiently large n and any I, we have* $\mathrm{E}[T(n, I)] \leq c_1 n \mathrm{E}[T_1(n, I)]$.

Now we analyze the convergence speed of game_1. For our analysis, we split a game execution into stages where each stage is a part of the game until some amount of players become bankrupt. More specifically, we denote by $t_1(n, I_1, \ldots, I_n)$ the number of steps needed in a game with n players and initial weights $I_1, \ldots, I_n$ until at least 1 player becomes bankrupt. The following lemma relates the two terms $T_1(n, I_1, \ldots, I_n)$ and $t_1(n, I_1, \ldots, I_n)$.

Lemma 6 . *For any n and $I_1, \ldots, I_n$, there exists a constant c_2, $c_2 \geq 1$, and weights $I'_1, \ldots, I'_{n-c_2}$ such that the following inequality holds.*

$$\mathrm{E}[T_1(n, I_1, \ldots, I_n)] \;\leq\; \mathrm{E}[t_1(n, I_1, \ldots, I_n)] + \mathrm{E}[T_1(n - c_2, I'_1, \ldots, I'_{n-c_2})].$$

Proof. Let $Y \subseteq Y_n$ be the set of all valid game sequences y such that the number of players becomes strictly smaller than n for the first time just after y. By definition of $\mathrm{E}[T_1(n, I_1, \ldots, I_n)]$, we have the following equality.

$$\mathrm{E}[T_1(n, I_1, \ldots, I_n)] \;=\; \sum_{x \in Y_n} \widetilde{\Pr}\{x\} \cdot |x| \;=\; \sum_{\substack{x \in Y_n,\, y \in Y \\ x = yz}} \widetilde{\Pr}\{yz\}(|y| + |z|).$$

Notice here that we can split $\widetilde{\Pr}\{yz\}$ in two factors, $\widetilde{\Pr}\{y\}$ and $\widetilde{\Pr}_y\{z\}$, where $\widetilde{\Pr}_y\{z\}$ determines the probability of z after the game follows y. Also note that the set Y_y of strongly valid z depends on y. Thus, we can rewrite the above expression as follows.

$$\begin{aligned}
&\mathrm{E}[T_1(n, I_1, \ldots, I_n)] \\
&= \sum_{y \in Y} \sum_{z \in Y_z} \widetilde{\Pr}\{y\}\widetilde{\Pr}_y\{z\} \cdot |y| + \sum_{y \in Y} \sum_{z \in Y_z} \widetilde{\Pr}\{y\}\widetilde{\Pr}_y\{z\} \cdot |z| \\
&= \left(\sum_{y \in Y} \widetilde{\Pr}\{y\}|y| \right) \sum_{z \in Y_z} \widetilde{\Pr}_y\{z\} + \sum_{y \in Y} \widetilde{\Pr}\{y\} \left(\sum_{z \in Y_z} \widetilde{\Pr}_y\{z\} \cdot |z| \right) \\
&= \mathrm{E}[t_1(n, I_1, ..., I_n)] \sum_{z \in Y_z} \widetilde{\Pr}_y\{z\} + \sum_{y \in Y} \widetilde{\Pr}\{y\} \mathrm{E}[T_1(n_y, I_{i_1(y)}, \ldots, I_{i_{n_y}(y)})] \\
&\leq \mathrm{E}[t_1(n, I_1, \ldots, I_n)] + \mathrm{E}[T_1(n - c', I'_1, \ldots, I'_{n-c'})].
\end{aligned}$$

where the values of n_y and $I_{i_1(y)}, \ldots, I_{i_{n_y}(y)}$ are determined by the result of the game following y. On the other hand, c' and I'_i are chosen so that the value of $\mathrm{E}[T_1(n_y, I_{i_1(y)}, \ldots, I_{i_{n_y}}(y))]$ is maximized. These values always exist since even if there is an infinite number of game sequences y that appear on the summation, there is only a finite number of possible values for n_y (since n_y must be between 1 and $n-1$) and $I_{i_j(y)}$ (since $\sum_j I_{i_j(y)} = In$).

By this lemma we can use induction for bounding the expected value of T_1. Recall that when analyzing $t_1(n, I_1, \ldots, I_n)$, by the way it is defined, no player becomes bankrupt, and thus, the amount of decrement is fixed to c_{inc}/n. Thus, game_1 until at least one player becomes bankrupt is regarded as a n-dimensional random walk, which is much easier to analyze. In fact, we can use the following lemma.

Lemma 7. *Let X be a random variable that is 1 with probability $1/n$ and 0 with probability $1 - 1/n$, and let $S = X_1 + \ldots + X_t$, the sum of the outcomes of t random trials of X. Then, for some constant $\alpha > 0$, the following holds for any t and n.*

$$\Pr\left\{ S \leq \frac{t}{n} - \alpha\sqrt{\frac{t}{n}} \right\} \;>\; 1/3.$$

Proof. We estimate the probability that the statement of the lemma is false and show that it is less than 2/3. That is, we upper bound the following probability.

$$\Pr\left\{S > \frac{t}{n} - \alpha\sqrt{\frac{t}{n}}\right\} = \Pr\left\{S > \frac{t}{n}\right\} + \Pr\left\{\frac{t}{n} \geq S > \frac{t}{n} - \alpha\sqrt{\frac{t}{n}}\right\}.$$

The first term in the sum is bounded by $1/2$; see, e.g., [JS72]. Let s be the smallest integer such that $s > t/n - \alpha\sqrt{t/n}$. We calculate, by using Stirling's approximation, the second term of the above sum as follows.

$$\begin{aligned}
\Pr\left\{\frac{t}{n} \geq S > \frac{t}{n} - \alpha\sqrt{\frac{t}{n}}\right\} &= \sum_{i=s}^{t/n} \binom{t}{i} \left(\frac{1}{n}\right)^i \left(1 - \frac{1}{n}\right)^{t-i} = \sum_{i=s}^{t/n} \binom{t}{i} \frac{(n-1)^{t-i}}{n^t} \\
&\leq \sum_{i=s}^{t/n} \sqrt{\frac{t}{2\pi i(t-i)}} \left(\frac{t}{t-i}\right)^t \left(\frac{t-i}{i}\right)^i \frac{(n-1)^{t-i}}{n^t} \\
&\leq \sum_{i=s}^{t/n} \sqrt{\frac{t}{2\pi i(t-i)}} \left(\frac{t(n-1)}{n(t-i)}\right)^t \left(\frac{t-i}{i(n-1)}\right)^i.
\end{aligned}$$

Also routine calculations show that $\left(\frac{t(n-1)}{n(t-i)}\right)^t \left(\frac{t-i}{i(n-1)}\right)^i$ is always less than 1 for $s \leq i \leq t/n$ and that this factor is maximized when $i = t/n$. From this by simple calculation, we obtain the desired bound with $\alpha = \sqrt{\pi/12}$.

Now we are now ready to make the following claim[3].

Claim.

$$\mathrm{E}[T(n, I)] = O\left(n \ln n \left(\frac{In}{c_{\mathrm{inc}}}\right)^2\right).$$

Justification. We start with estimating $\mathrm{E}[t_1(n, I_1, \ldots, I_n)]$ by using the above lemma. For a given t and for any i, let t_i be the number of times that player i wins within t steps. Then w_i, the weight of player i, in game_1 is expressed as follows.

$$w_i = I_i + c_{\mathrm{inc}} t_i - c_{\mathrm{dec}} t \leq I_i + c_{\mathrm{inc}} \left(t_i - \frac{t}{n}\right).$$

(For simplifying our notation, we use c to denote c_{inc} in the following.)

Moreover, since game_1 until at least one player becomes bankrupt is regarded as a n-dimensional random walk, we can use Lemma 7 to show that the following event happens with probability bigger than $1/3$.

$$w_i = I_i + c\left(t_i - \frac{t}{n}\right) \leq I_i + c\left(\frac{t}{n} - \alpha\sqrt{\frac{t}{n}} - \frac{t}{n}\right) = I_i - c\alpha\sqrt{\frac{t}{n}}.$$

[3] We do not have a rigorous proof for this result, and for this reason we stated it as a claim.

Therefore, with probability more than $1/3$, the weight of player i becomes zero or negative if $c\alpha\sqrt{t/n} \geq I_i$, that is, $t \geq (I_i/c\alpha)^2 n$. Now sort players by their initial weights, and define P to be the set of the first (i.e., the smallest) $n/2$ players. Since the total weight is W_0 (at any step), all players in P have weight at most $2W_0/n$ and therefore,

$$\Pr\{\, w_i \leq 0 \text{ in } t_0 = n\left(\frac{2W_0}{nc\alpha}\right)^2 \text{ steps} \mid i \in P \,\} \;>\; \frac{1}{3}.$$

Moreover, *if* we can assume that each player in P become bankrupt independently, we also have the following probability:

$$\Pr\{\, \text{There exists } i \in P, \text{ such that } w_i \leq 0 \text{ in } t_0 \text{ steps} \,\} \;>\; 1 - \left(\frac{2}{3}\right)^{n/2}.$$

From this observation, it is reasonable to bound $\mathrm{E}[t_2(n, I_1, \ldots, I_n)]$ by $c_3 t_0$ for some constant c_3 since for most of the valid game sequences (a $1 - (2/3)^{n/2}$ fraction of them) this bound holds.

Now combining the above lemmas and the obtained bound, we have

$$\begin{aligned}
\mathrm{E}[T_1(n, I)] &\leq \mathrm{E}[t_1(n, I_1, \ldots, I_n)] + \mathrm{E}[T_1(n - c_2, I'_1, \ldots, I'_{n-c_2})] \\
&\leq c_3 n \left(\frac{2W_0}{nc\alpha}\right)^2 + \mathrm{E}[T_1(n - c_2, I'_1, \ldots, I'_{n-c_2})] \\
&\leq c_3 n \left(\frac{2W_0}{nc\alpha}\right)^2 + c_3(n-1)\left(\frac{2W_0}{(n-1)c\alpha}\right)^2 \\
&\quad + \mathrm{E}[T_1(n - c'_2, I'_1, \ldots, I'_{n-c'_2})] \cdots \\
&\leq c_3 \ln n \left(\frac{W_0}{c\alpha}\right)^2 \leq c_4 \ln n \left(\frac{In}{c}\right)^2, \quad \text{for some constant } c_4.
\end{aligned}$$

From this and Lemma 5 , we obtain the desired bound.

Acknowledgments

This research has been started while the second author visited the CRM, Centre de Recerca Matemàtica, Institut D'Estudis Catalans. We thank many researchers; in particular, to Shigeru Tanaka at the Institute of Physical and Chemical Research (RIKEN) and Masanobu Miyashita at NEC Fundamental Research Laboratories for their kind guidance to this field, to Paul Vitányi for suggesting Lemma 7 , to Shigeru Mase for suggesting [JS72], and to Jose Balcázar, Miklos Santha, and Carme Torras for their interest and discussion.

References

[DWY98] C. Domingo, O. Watanabe, T. Yamazaki, A role of constraint in self-organization, *Research Report C-124*, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 1998; `http://www.is.titech.ac.jp/research/technical-report/C/index.html`.

[Fel68] W. Feller, *An Introduction to Probability Theory and its Applications*, volume I, third edition. John Wiley, 1968.

[JS72] Japanese Standards Association ed., *Statistical Tables and Formulas with Computer Applications*, JSA-1972, Japanese Standards Association, Tokyo, 1972.

[Mal73] C. von der Malsburg, Self-organization of orientation sensitive cells in the striate cortex, *Kybernetik* 14 (1973), 85–100.

[MM94] K. Miller and D. MacKay, The role of constrains in Hebbian learning, *Neural Computation* 16 (1994), 100–126.

[Swi96] N. Swindale, The development of topography in the visual cortex: a review of models, *Network* 7 (1996), 161–247.

[Tan90] S. Tanaka, Theory of self-organization of cortical maps: mathematical framework, *Neural Networks*, 3 (1990), 625–640.

[WH63] T. Wiesel and D. Hubel, Effects of visual deprivation on morphology and physiology of cells in cat's lateral geniculate body, *J. Neurophysiology*, 26 (1963), 978–993.

This article was processed using the LaTeX macro package with LLNCS style

CONSTRUCTIVE BOUNDS AND EXACT EXPECTATIONS FOR THE RANDOM ASSIGNMENT PROBLEM

DON COPPERSMITH AND GREGORY B. SORKIN†

Abstract

The random assignment problem is to choose a minimum-cost perfect matching in a complete $n \times n$ bipartite graph, whose edge weights are chosen randomly from some distribution such as the exponential distribution with mean 1. In this case it is known that the expectation does not grow unboundedly with n, but approaches a limiting value $c^\star$ between 1.51 and 2. The limit is conjectured to be $c^\star = \pi^2/6$, while a recent conjecture has it that for finite n, the expected cost is $\mathbb{E}A^\star = \sum_{i=1}^{n} 1/i^2$.

By defining and analyzing a constructive algorithm, we show that the limiting expectation is $c^\star < 1.94$. In addition, we generalize the finite-n conjecture to partial assignments on complete $m \times n$ bipartite graphs, and prove it in some limited cases. A full version of our work is available as [CS98].

1. Introduction

The assignment problem, a special case of the transportation problem, is to find a minimum cost perfect matching in an edge-weighted bipartite graph. The expected cost of a minimum assignment, in a randomized setting, seems first to have been considered by Donath [Don69]. If A is a complete $n \times n$ bipartite graph whose edge weights are independent and identically distributed (i.i.d.) random variables (r.v.'s) in the interval $(0, 1)$, Donath observed through simulations that, as $n \to \infty$, the cost appeared to tend to some constant $c^\star$ between 1 and 2.

Walkup proved that $\limsup_{n\to\infty} \mathbb{E}A^\star \leqslant 3$, by showing that for large n a certain "2-out" subgraph containing only edges of small weight is likely to contain a perfect matching [Wal79]. The method was made constructive (and efficient) by Karp, Kan, and Vohra [KKV94]. Karp [Kar84, Kar87] showed that $\mathbb{E}A^\star \leqslant 2$, by analyzing the problem's linear program (LP) and its dual; the method was quickly generalized by Dyer, Frieze, and McDiarmid [DFM86].

Lazarus proved that $\liminf_{n\to\infty} \mathbb{E}A^\star \geqslant 1 + 1/e$ [Laz79, Laz93]. His method can be viewed as exploiting the dual LP, but is also easy to understand naively; we will develop the result later. Lazarus's methods were extended at about the same time by Goemans and Kodialam [GK93], who showed a lower bound of 1.41, and by Olin [Oli92], who showed a lower bound of 1.51.

On a different front, Aldous [Ald92] proved that as $n \to \infty$, $A^\star$ converges to a well-defined limit $c^\star$, in distribution as well as expectation. Aldous also observed that as $n \to \infty$, the uniform (0,1) distribution and the exponential distribution with mean 1 (probability density $p(x) = \exp(-x)$) are equivalent. Intuitively, a good assignment uses only small values, so only the distribution's density near 0 is relevant; these two distributions both have density 1 there. Use of the exponential distribution instead of uniform vastly simplifies the results of all the works cited, both because the minimum of n mean-1 exponentials is a mean-$1/n$ exponential, and because an exponential random

† Both authors are members of the IBM T.J. Watson Research Center, Department of Mathematical Sciences, Yorktown Heights NY 10598; copper@watson.ibm.com and sorkin@watson.ibm.com.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 319-330, 1998.

variable X conditioned by $X \geqslant c$ has the property that $X - c$ is an (unconditioned) exponential random variable.

Most tantalizingly, Mézard and Parisi [MP85, MP87] used the "replica method" of statistical physics to calculate that $c^\star = \pi^2/6$; the method is not mathematically formal, but its results have been borne out in a variety of cases. Parisi [Par98] made the further conjecture that for an $n \times n$ instance A with i.i.d. exponential costs (the distribution is critical in the finite case), $\mathbb{E}A^\star = \sum_{i=1}^{n} 1/i^2$. Since it is well known that $\sum_{i=1}^{\infty} 1/i^2 = \pi^2/6$, proof of this conjecture would immediately validate the replica calculation.

In this paper we describe an algorithm whose analysis yields an upper bound of $c^\star < 1.94$. Also, we generalize Parisi's conjecture, and prove it in a number of cases.

2. Lower bounds

We begin with a review of Lazarus's $1 + 1/e$ lower bound, both to introduce some basic principles and because its construction is the starting point for our assignment algorithm and upper bound. We will need two simple properties of the exponential distribution:

Property 1. *The minimum of a collection of n i.i.d. exponential r.v.'s with mean 1 is an exponential r.v. with mean $1/n$.*

Property 2. *For an exponential r.v. X with parameter λ (density $p(x) = \frac{1}{\lambda} e^{-\lambda x}$), for any $s, t \geqslant 0$, $\Pr\{X > s + t \mid X > t\} = \Pr\{X > s\}$. (That is, conditional upon $X \geqslant t$, the distribution of $X - t$ is again exponential, with the same parameter.)*

All existing lower bounds on expected assignment cost are based on feasible (not necessarily optimal) solutions to an assignment problem's dual LP. The key is the transformation of an instance A:

Lemma 3. *For real n-vectors $\mathbf{u}$ and $\mathbf{v}$, if $A - \mathbf{u}\mathbf{1}^T - \mathbf{1}\mathbf{v}^T = A'$ (that is, $a_{ij} - u_i - v_j = a'_{ij}$), then $A^\star = \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j + A'^\star$.*

Proof. Since any assignment σ selects precisely one value from each row and each column, $\mathrm{cost}(A, \sigma) = \mathrm{cost}(A', \sigma) + \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$. Thus the minimum-cost assignments for A and A' are achieved by the same σ, and $A^\star = A'^\star + \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$. □

Corollary 4. *If $A - \mathbf{u}\mathbf{1}^T - \mathbf{1}\mathbf{v}^T = A'$, and A' is elementwise non-negative, then $A^\star \geqslant \sum u_i + \sum v_j$.*

Lemma 5. $\mathbb{E}A^\star \geqslant 1$.

Proof. Let $\mathbf{u}$ be the vector of row minima, and $\mathbf{v} = \mathbf{0}$. By Property 1, $\mathbb{E}u_i = 1/n$. Applying Lemma 3 and Corollary 4, and taking expectations, $\mathbb{E}A^\star = \mathbb{E}A'^\star + \mathbb{E}\sum u_i \geqslant n \cdot 1/n = 1$. □

Theorem 6 (Lazarus). $\liminf_{n\to\infty} \mathbb{E}A^\star \geqslant 1 + 1/e$.

Proof. As above, subtract the row minima u_i in A to give A' and conclude that $\mathbb{E}A^\star = 1 + \mathbb{E}A'^\star$. By Property 2, subtracting u_i from values a_{ij} other than the row-i minimum produces a collection of i.i.d. exponential r.v.'s $a'_{ij} = a_{ij} - u_i$.[1] Thus A' is (like A) an i.i.d. exponential matrix, except that in each row it has a single zero, in a random column.

In the large-n limit, the probability that a column contains no zeros is $1/e$. Subtract the column minima v_j in A' to give A''. If column j contained a zero, $v_j = 0$; otherwise (by Property 1) v_j has

[1] This argument, based on the exponential distribution, is easier than Lazarus's for the uniform distribution. The memoryless property of the exponential means that when row and/or column minima are subtracted from an i.i.d. exponential matrix, the result is again i.i.d. exponential (outside of the zeros); this "self-reducibility" is most useful. (The matrix A and its "residue", after subtraction of the row and column minima, are far from independent of one another, but that is not a concern.)

exponential distribution with mean $1/n$. So in the large-n limit, $\mathbb{E}\sum_{j=1}^{n} v_j = n \cdot 1/e \cdot 1/n = 1/e$, and by Lemma 3,

$$\mathbb{E}A^\star \;=\; \mathbb{E}A'^\star + \mathbb{E}\sum u_i + \mathbb{E}\sum v_j \;\geqslant\; 1 + 1/e.$$

□

Olin's improved bound $\liminf_{n\to\infty} \mathbb{E}A^\star < 1.51$ takes off from inequality (2), replacing $\mathbb{E}A'^\star \geqslant 0$ with a stronger bound by reasoning about groups of rows whose zeros all lie in the same column.

3. Outline of Sections 4–6

The lower bounds are based on constructing feasible solutions to the assignment problem's dual LP; there is a certain satisfaction in the constructions, and the sense that their continuing incremental improvement might result in lower bounds arbitrarily close to the true value. By contrast, the upper bound has rested at exactly 2 for over a decade, and is not constructive.

We derive a smaller upper bound by defining and analyzing an assignment algorithm which, applied to a large i.i.d. exponential matrix A, produces an assignment with expected cost less than 1.94. There are three key points. **(1)** We begin with a partial assignment of size about $.81n$ implicit in Lazarus's lower-bound construction. **(2)** To complete even a partial assignment of size $n-1$ to a complete assignment in the naive manner would increase the assignment cost by 1, which when we are gunning for $\pi^2/6$ is intolerable. However, we show how an $(n-1)$-assignment can be completed to an n-assignment quite simply, with additional cost $o(1)$. Dogged extension of the technique allows us to complete an initial assignment of size about $.81n$ to an n-assignment with additional cost of only about 0.56 altogether. **(3)** In both the initial construction and the successive augmentations, the "working" matrix will consist entirely of zeros and i.i.d. exponential values.

4. Initial assignment

Begin with an i.i.d. exponential matrix A, and — following Lazarus's lead — form a "reduced" matrix A' according to the following algorithm:

Algorithm 0
Input: An $n \times n$ matrix A of i.i.d. exponential elements.

For $i = 1 \ldots n$:
    Subtract from row i its minimum value u_i.
For $j = 1 \ldots n$:
    Subtract from column j its minimum value v_j.
For $i = 1 \ldots n$:
    For $j = 1 \ldots n$:
        If element (i, j) is zero, and has other zeros in its row and in its column,
            replace it with an exponential r.v.

A typical reduced matrix is depicted in Figure 1, and is described by the following theorem:

Theorem 7. *For n asymptotically large, Algorithm 0 reduces an $n \times n$ matrix A of i.i.d. exponential r.v.'s to a matrix A' in which*

- *there is at least one zero in each row and in each column;*
- *a zero may share either its row or its column with other zeros, but not both;*
- *the collection of nonzero elements are i.i.d. exponential random variables;*

and, with probability $1 - o(1)$,

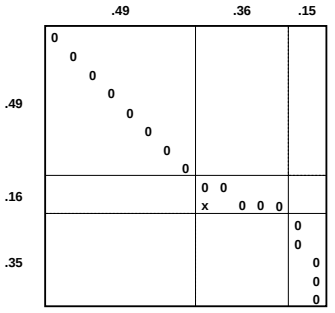


FIGURE 1. The zero structure of the matrix resulting from Algorithm 0: subtraction of row and column minima, and replacement of selected zeros (such as in the spot marked with an "x") by fresh random variables.

- *the fraction of columns having $k > 1$ zeros is asymptotically equal to*

$$\mathrm{col}(k) = \mathrm{Pois}_k(e^{-e^{-1}});$$

- *the fraction of rows having $k > 1$ zeros is asymptotically equal to*

$$\mathrm{row}(k) = (1 - p^\star)\,\mathrm{Pois}_k(1/e) + (p^\star)\,\mathrm{Pois}_{k-1}(1/e),$$

  where $p^\star = (e^{-e^{-e^{-1}}} - e^{-1})/(1 - e^{-e^{-1}})$; and
- *the fraction of zeros unique in their row and column is asymptotically equal to*

$$\mathrm{row}(1) = \mathrm{col}(1) = e^{-e^{-1}} \left\{ e^{-1} + e^{-e^{-e^{-1}}} + \frac{e^{-1}\left(e^{-1} - e^{-e^{-e^{-1}}}\right)}{1 - e^{-e^{-1}}} \right\} \approx .4914.$$

Proof. The first two assertions hold by construction: A zero is generated in every row and every column, and zeros are removed when they share their column and their row with others. The third assertion follows from Property 1.

When row minima are subtracted, the number of zeros contained in any column j is the sum of n Bernoulli$(1/n)$ random variables; this has binomial distribution $B(n, 1/n)$, which (as $n \to \infty$) converges in distribution to Pois(1). Immediately, the expected number of columns having j zeros is $n\,\mathrm{Pois}_j(1)$. The actual number is tightly concentrated about this expectation, by Azuma's inequality [McD89]: all statements in the remainder of the proof will implicitly be "almost surely" and "almost exactly".

From the above, $n\,\mathrm{Pois}_0(1) = n/e$ columns contain no row-induced zeros, so subtracting column minima introduces n/e new zeros. They fall in random rows, with a density of $1/e$ new zeros per row, so the number of column-induced zeros in each row is distributed as Pois$(1/e)$.

Consider the number of columns with $k > 1$ zeros. A column only acquires multiple zeros in the row phase; a zero is removed if any of the n/e column zeros falls into the same row *and* if the zero is not the last remaining in its column. Forgetting this exceptional "*and*" only results in miscounting columns with no or one zeros. Then the probability that a zero is allowed to remain, is the probability that no column zero falls into its row, which is $\mathrm{Pois}_0(1/e) = e^{-e^{-1}}$. Thus a column may be modeled as having n slots, into each of which a zero is placed w.p. $1/n$, and allowed to remain w.p. $e^{-e^{-1}}$; this is equivalent to just placing zeros w.p. $e^{-e^{-1}}/n$, which for n large is the Poisson process Pois$(e^{-e^{-1}})$. So the fraction of columns containing $k > 1$ zeros is $\mathrm{col}(k) = \mathrm{Pois}_k(e^{-e^{-1}})$.

Counting rows with $k > 1$ zeros is a little bit trickier but follows a similar line. (See [CS98] for details.)

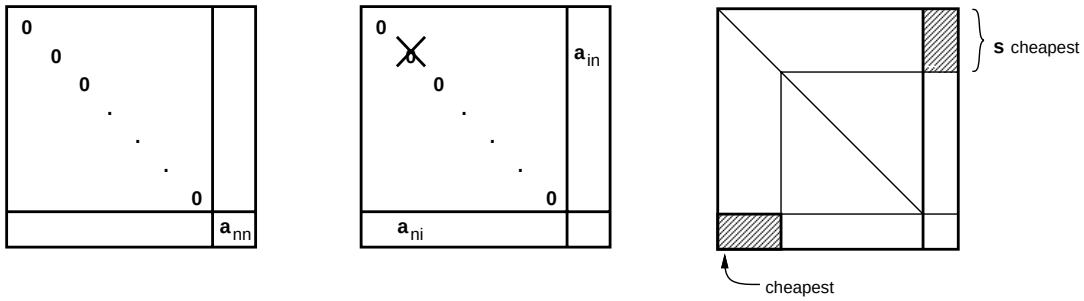


FIGURE 2. At left, the naive completion of an $(n-1)$-assignment adds expected cost $\mathbb{E}a_{nn} = 1$. At center, adding the cheapest pair $a_{in} + a_{ni}$ and deleting a_{ii} adds expected cost $O(\sqrt{1/n})$. At right, with $s = \lfloor\sqrt{2n}\rfloor$, finding the s smallest elements in column n (depicted without loss of generality as elements $1 \ldots s$), then choosing the smallest corresponding element $\hat{\jmath}$ of row n, gives a pair with $\mathbb{E}(a_{\hat{\jmath}n} + a_{n\hat{\jmath}}) \asymp \sqrt{2/n}$.

This leaves rows which neither have 2 or more zeros, nor are associated with a column having 2 or more zeros. These are counted by $\text{row}(1) = 1 - \sum_{k=2}^{\infty} k\,\text{col}(k) - \sum_{k=2}^{\infty} \text{row}(k)$, which is a straightforward calculation with Poisson distributions. □

Corollary 8. *For asymptotically large graphs, the size of a maximum assignment contained in the bipartite graph B_1 formed by edges which have minimum cost for either their row or their column is almost surely almost exactly n times $2 - e^{-e^{-1}} - e^{-e^{-e^{-1}}}$ (approximately .8073n).*

Pittel and Weishaar [PW98] compute the size of a maximum assignment in B_k, the graph formed of the k cheapest edges out of each row and column, using entirely different methods; for B_1 they find a definite integral whose numerical evaluation agrees with our closed-form expression.

Corollary 8 means that at a cost of $1 + 1/e$ (in row and column dual variables), Lazarus's construction produces an assignment of size about $.81n$ consisting entirely of zero-weight edges, with i.i.d. exponential edge weights elsewhere. How cheaply can this assignment be completed?

5. AUGMENTING AN m-ASSIGNMENT

First consider a simpler question. Suppose an $n \times n$ matrix A has $n-1$ zeros along its diagonal (a zero-cost assignment of size $n-1$), and i.i.d. exponential entries elsewhere. How cheaply can this assignment be completed? The simplest way to complete the assignment is to assign row n to column n. This adds 1 to the expected cost, since $\mathbb{E}a_{nn} = 1$; as our goal is to find an assignment of cost less than 2 (ideally, $\pi^2/6$), an additional cost of 1 is far too much.

A better way (see Figure 2) is to remove from the matching some edge a_{ii}, and add the two edges a_{in} and a_{ni}; this is augmentation by the 3-edge alternating path a_{ni}, a_{ii}, a_{in}. There are $n-1$ ways of doing this (choosing $i \in 1, \ldots, n-1$), and the cheapest will have expected cost $\mathbb{E}\min_i(a_{in} + a_{ni}) = \Theta(1/\sqrt{n})$.

We can implement the same principle a little differently, fixing a value s, choosing the s smallest elements in the unmatched (or "free") column, then selecting the cheapest corresponding element in the unmatched row. The s smallest "column elements" have average expectation $(1/n + \ldots + s/n)/s = (s+1)/(2sn)$, and the smallest of the corresponding s "row elements" has expectation $1/s$, so for $s \asymp \lfloor\sqrt{2n}\rfloor$, the expected added cost is asymptotically $\sqrt{2/n}$.

More generally, suppose only m of the n rows and columns are matched, each at cost 0. Again we can augment to an $(m+1)$-assignment by constructing a 3-edge alternating path from the first unmatched row to one of the unmatched columns, as per the following algorithm:

Algorithm 2
Input: matrix A; m-assignment of zero-weight edges, with $m > n/2$;
positive integer $s \leqslant m/(n-m)$.

For each unmatched column $k = m+1 \ldots n$ in turn:
  Choose the s cheapest elements a_{ik} among rows $i \in 1 \ldots m$.
  For each row i chosen:
    Define $\mathrm{parent}(i) = k$.
    Eliminate row i from future choices.
Let S be the set of row indices i chosen. ($|S| = s(n-m)$.)
Let $\hat{\jmath} = \mathrm{argmin}_{j \in S}\{a_{m+1,j}\}$, the index of the cheapest of the $s(n-m)$
  elements in row $m+1$ and columns S.
Add edges $a_{m+1,\hat{\jmath}}$ and $a_{\hat{\jmath},\mathrm{parent}(\hat{\jmath})}$ to the assignment, and remove edge $a_{\hat{\jmath},\hat{\jmath}}$.
Let $\mathbf{u} = \mathbf{0}$, except for $u_{m+1} = a_{m+1,\hat{\jmath}}$.
Let $\mathbf{v} = \mathbf{0}$, except for $v_{\mathrm{parent}(\hat{\jmath})} = a_{\hat{\jmath},\mathrm{parent}(\hat{\jmath})}$.
Let $A' = A - \mathbf{u}\mathbf{1}^T - \mathbf{1}\mathbf{v}^T$.
Replace any negative elements a'_{ij} with fresh exponential r.v.'s.
Replace element $a'_{\hat{\jmath},\hat{\jmath}}$ with a fresh exponential r.v.

Lemma 9. *Given an $n \times n$ matrix A with elements $a_{ii} = 0$ for $i = 1 \ldots m$, and i.i.d. exponential random elsewhere, with $n - m \gg 0$. Then Algorithm 2 produces A', $\mathbf{u}$, and $\mathbf{v}$ such that: $A \leqslant A' + \mathbf{u}\mathbf{1}^T + \mathbf{1}\mathbf{v}^T$;*

$$\mathbb{E}\sum_i u_i + \mathbb{E}\sum_j v_j \leqslant \frac{1}{s(n-m)} + \frac{s+1}{2s(n-m)} \ln\left(\frac{m}{m - s(n-m)}\right);$$

and the elements of A' are i.i.d. exponential except on an $(m+1)$-assignment all of whose edge weights are zero.

Proof. The first claim, that $A \leqslant A' + \mathbf{u}\mathbf{1}^T + \mathbf{1}\mathbf{v}^T$, is immediate by construction. For the second claim,

$$\mathbb{E}\sum u_i + \mathbb{E}\sum v_j = a_{m+1,\hat{\jmath}} + a_{\hat{\jmath},\mathrm{parent}(\hat{\jmath})}.$$

As the smallest of $s(n-m)$ elements, $\mathbb{E}a_{m+1,\hat{\jmath}} = 1/s(n-m)$. Furthermore, the smallest element occurs in random position: $\hat{\jmath} \in S$ is chosen uniformly at random, and so the expectation of the chosen column element $a_{\hat{\jmath},\mathrm{parent}(\hat{\jmath})}$ is merely the expected mean of all the elements $a_{j,\mathrm{parent}(j)}$ generated by the algorithm. The first s of these elements chosen, from amongst rows $1 \ldots m$, have expectations $1/m, \ldots, s/m$; the next s chosen (from amongst s fewer rows) have expectations $1/(m-s), \ldots, s/(m-s)$; and so forth, with the last column's elements having expectations $1/(m-(n-m-1)s), \ldots, s/(m-(n-m-1)s)$. The mean is

$$\mathbb{E}a_{\hat{\jmath},\mathrm{parent}(\hat{\jmath})} = \frac{1}{s(n-m)} \sum_{k=0}^{n-m-1} \frac{s(s+1)}{2} \frac{1}{m-ks} \leqslant \frac{s+1}{2(n-m)} \int_0^{n-m} \frac{1}{m-xs} dx.$$

Thus

$$\mathbb{E}\sum u_i + \mathbb{E}\sum v_j \leqslant \frac{1}{s(n-m)} + \frac{s+1}{2s(n-m)} \ln\left(\frac{m}{m - s(n-m)}\right).$$

Finally, we must show that apart from the $m+1$ zeros, the elements of A' are i.i.d. exponential r.v.'s. Before $a_{m+1,\hat{\jmath}}$ is chosen, row $m+1$ is unrevealed. Its revelation can be simulated by: choosing $\hat{\jmath} \in S$ uniformly at random; setting $a_{m+1,\hat{\jmath}}$ equal to an exponential random variable with mean $1/|S|$; for the remaining $j \in S - \hat{\jmath}$, setting $a_{m+1,j} = a_{m+1,\hat{\jmath}} + x_j$, where the x_j are independent mean 1 exponential r.v.'s; and for $j \notin S$, setting $a_{m+1,j} = x_j$, where the x_j are again independent mean-1 exponentials. Subtracting $a_{m+1,\hat{\jmath}}$ from each $a_{m+1,j}$, and replacing negative values with fresh exponentials, results in an i.i.d. exponential n-vector.

The above simulation begins by choosing $\hat{\jmath}$, which may be done by specifying in advance, at random, the time — from 1 to $s(n-m)$ — at which $\hat{\jmath}$ was added to S; this gives us the ability to stop generating the set S once $\hat{\jmath}$ is produced, say while choosing from column $\hat{k}$. It is crucial that all columns were equally represented, with s choices each, for this ensures that $\hat{k}$ is chosen uniformly at random from $m+1, m+2, \ldots, n$, and in turn that the columns $k \neq \hat{k}$ *not* chosen retain their unbiased i.i.d. exponential distribution. (Otherwise, for example, high-probability selection of a column $\hat{k}$ with particularly small entries would bias the untouched columns to have large entries.) In column $\hat{k}$, elements not probed (because their rows were unmatched, or were selected for an earlier column) remain random. Probed elements smaller than the chosen $a_{\hat{\jmath},\hat{k}}$ become negative when $a_{\hat{\jmath},\hat{k}}$ is subtracted, and are replaced by fresh r.v.'s. And probed elements larger than $a_{\hat{\jmath},\hat{k}}$ become unbiased exponentials when $a_{\hat{\jmath},\hat{k}}$ is subtracted, independent of one another and of the rest of the matrix. □

Corollary 10. *An $n \times n$ matrix A of i.i.d. exponential r.v.'s, with n large, has an assignment whose expected cost is less than 2.92. Therefore, $c^\star < 2.92$.*

Proof. Given a matrix which initially has an all-zero assignment of size m, apply Algorithm 2 repeatedly ($n-m$ times), producing vectors $\mathbf{u}(m+1)$ and $\mathbf{v}(m+1)$, $\mathbf{u}(m+2)$ and $\mathbf{v}(m+2)$, ..., up to $\mathbf{u}(n)$ and $\mathbf{v}(n)$. By Lemma 9, and choosing s at each iteration to minimize the cost bound added, the total weight of these vectors satisfies

$$\begin{aligned}
&\sum_{m'=m+1}^{n} \mathbb{E}\Big(\sum u_i(m') + \sum v_j(m')\Big) \\
&\qquad \leqslant \sum_{m'=m+1}^{n} \min_{s\in\mathbb{Z}^+} \left\{ \frac{1}{s(n-m')} + \frac{s+1}{2(n-m')s} \ln\left(\frac{m'}{m'-s(n-m')}\right)\right\} \\
&\qquad \asymp \int_{m}^{n} \min_{s\in\mathbb{Z}^+} \left\{ \frac{1}{s(n-m')} + \frac{s+1}{2(n-m')s} \ln\left(\frac{m'}{m'-s(n-m')}\right)\right\} dm'
\end{aligned}$$

and, changing variables to $x = 1 - m'/n$,

$$\begin{aligned}
&= \int_{1-m/n}^{0} \min_{s\in\mathbb{Z}^+} \left\{ \frac{1}{xns} + \frac{s+1}{2xns} \ln\left(\frac{1-x}{1-x-sx}\right)\right\} d(-xn). \\
&= \int_{x=0}^{1-m/n} \min_{s\in\mathbb{Z}^+} \left\{ \frac{1}{xs} + \frac{s+1}{2xs} \ln\left(\frac{1-x}{1-x-sx}\right)\right\} dx.
\end{aligned}$$

For $m/n = b_1 = 2 - e^{-e^{-1}} - e^{-e^{-e^{-1}}} \approx .81$, using a combination of numerical techniques (for x large, say $x > .01$) and algebraic ones (for x small), yields

$\sum_{m'=m+1}^{n} \mathbb{E}(\sum u_i(m') + \sum v_j(m')) < 1.55$. That is, a matrix A' which is i.i.d. exponential except for an all-zero assignment of size $(1+o(1))b_1 n$, has assignment cost $\mathbb{E}A'^\star < 1.55 + o(1)$. Applying Algorithm 0 to an i.i.d. exponential matrix A produces a matrix which includes an all-zero assignment of size $(1+o(1))b_1 n$, some additional zeros, and i.i.d. exponential r.v.'s. Replacing its non-assignment zeros with new exponential r.v.'s produces a matrix A' as required, and with $\mathbb{E}A^\star \leqslant \mathbb{E}A'^\star + 1 + 1/e$. Thus for a random i.i.d. exponential matrix A, $\mathbb{E}A^\star < 1 + 1/e + 1.55 < 2.92$. □

To our knowledge, this is the first proof of $c^\star < 3$ that is based on algorithmically constructing an assignment of such cost.

6. An upper bound less than 2

Theorem 11. *An $n \times n$ matrix of i.i.d. exponential r.v.'s, with n large, has an assignment whose expected cost is less than 1.94. Therefore $c^\star < 1.94$.*

Proof. The proof is again based on an algorithmic construction, which unfortunately we will only be able to sketch here. (See [CS98] for details.) In the previous algorithm, each unmatched "free" column generated s "child" columns; row $m+1$ was assigned to one child, as the first link of a 3-edge alternating path ending at the child's parent.

Instead, we now let each free column, in turn, generate a single child. (As before, the children must be distinct, and disjoint from the parents.) Add the list of children to the end of the list of parents, doubling the number of free columns. Repeat this "doubling" procedure a parametrized number of times, which will in fact be until a constant-bounded fraction of the columns are free. Assign row $m+1$ to the cheapest free column, as the first link of an alternating path, and continue the alternating path by reassigning that column to its parent, the parent to the grandparent, and so forth. (The path lengths are typically, and at most, logarithmic in the ratio of n to the number of initially free columns.)

As before, the first assigned edge's value $a_{m+1,i}$ is subtracted from row $m+1$, and the other newly assigned edges' values are subtracted from their respective columns. The analysis follows the familiar lines: estimating the values of the newly assigned elements, and proving that the elements of the residual matrix are (outside of its $m+1$ zeros) again i.i.d. exponential r.v.'s.

To push the expected cost below 2, two other tricks are needed. The first is to do with the initial assignment and the initial set of free columns. After Algorithm 0 (refer to Figure 1), assign all rows with unique zeros to their corresponding columns, and to each column with two or more zeros, assign the first corresponding row. The later rows for such columns need to be assigned, and there are about $.19n$ of them. The initial free columns are those corresponding to rows with several zeros and there are about $.36n$ of them; it is an advantage that they outnumber the rows needing assignment. A row with several zeros, while not explicitly assigned, is not a concern: when all but one of its columns has been consumed, we assign it to that last one.

For the second trick, suppose that a column j initially contained two zeros, in rows i_1 and i_2; that row i_1 has been assigned to column j, and that we are now trying to assign row i_2. Go through the column-doubling procedure as usual, but now instead of selecting as the first edge of the alternating path the smallest value in row i_2 among the free columns, take the smallest value in rows i_1 or i_2 among the free columns. If the value falls in row i_2 proceed as usual; if it falls in row i_1, go through the same alternating path construction for row i_1, and assign row i_2 to column j. The virtue of the trick is that because we are choosing from twice as many values, the expectation of the first element of the alternating path is halved. The price we pay is that whichever row did not contain the smallest element is now biased, must be removed from consideration in all the remaining assignment augmentations, and, by thus reducing the options slightly, increases future augmentation costs. The tradeoff turns out to be advantageous, and the trick is employed for the last zero-bearing row of any column. □

7. Exact optima

Parisi's conjecture [Par98] that $\mathbb{E}A^\star = \sum_{i=1}^{n} 1/i^2$ seems to offer the best "handle" on the assignment problem. To that end, we have verified it to the extent we could, and in the process generalized it further:

Conjecture 12. *Given an $m \times n$ array of i.i.d., exponentially distributed random variables a_{ij} with mean 1, select $k \leqslant \min(m,n)$ elements, no two in the same row or column, such as to minimize the sum of those variables. (A k-assignment.) The expected value of this sum is*

$$\sum_{i,j \geq 0,\ i+j<k} \frac{1}{(m-i)(n-j)}.$$

We will show that the conjecture is true in the following four cases: $k = 1$; $k = 2$; $k = m = 3$; and $k = m = n = 4$. We will also show that in the case $k = m = n$, the conjectured sum reduces to $\sum_{i=1}^{k} \frac{1}{i^2}$, agreeing with Parisi's conjecture. Unfortunately, it seems difficult to extend these results.

Notation. By $F(k, m, n)$ we denote the expected value of the minimal k-assignment on an $m \times n$ array of i.i.d. exponential random variables $a_{i,j}$ with mean 1. By $G(k, m, n)$ we denote the expected value of the same problem, except with $a_{11} = 0$.

A few lemmas will be useful.

Lemma 13. *We have* $F(k, m, n) = G(k, m, n) + k/mn$.

Proof. Start with an $m \times n$ array A of i.i.d. exponential variables. Its global minimum has expected value $1/mn$. Subtract this global minimum from all entries, to produce an $m \times n$ array B of i.i.d. exponential variables except for one entry of 0, which we may take to be $b_{11} = 0$. An optimal k-assignment on A differs from one on B by the addition of the global minimum to each of the k selected elements. Then $\mathbb{E}A^\star = \mathbb{E}B^\star + k/mn$. □

Lemma 14. *For integers $k \leqslant m \leqslant n$, consider a nonnegative $m \times n$ matrix A with $a_{11} > 0$, where row 1 has y zero entries and column 1 has z zero entries, with $y + z \geqslant k$. Form the matrix B which agrees with A except that $b_{11} = 0$. Then the optimal k-assignments have equal values: $A^\star = B^\star$.*

Proof. Suppose an optimal k-assignment of A uses element a_{11}. We claim that, among the remaining $k - 1$ elements of the assignment, exactly y are in columns j such that $a_{1j} = 0$, since otherwise we could replace a_{11} in the assignment by one of the $a_{1j} = 0$ from an unused column, obtaining an assignment with a smaller value. Similarly, z elements are in rows i such that $a_{i1} = 0$. Since $y + z > k - 1$, one of the $k - 1$ elements a_{ij} of the assignment satisfies both conditions: $a_{i1} = a_{1j} = 0$. Replacing $a_{11} + a_{ij}$ by $a_{i1} + a_{1j} = 0$, we would obtain another k-assignment with a smaller value. In either case optimality is violated. So any optimal k-assignment for A avoids a_{11}, and thus has the same value as an optimal k-assignment for B. □

We will use this lemma to insert zeros into matrices.

Theorem 15. *Conjecture 12 holds in the cases $k = 1$; $k = 2$; $k = m = 3$; and $k = m = n = 4$.*

Proof.
Case $k = 1$: The smallest among mn i.i.d. mean-1 exponential variables has expected value $1/mn$.

Case $k = 2$: We appeal to Lemma 13, and show that for an $m \times n$ matrix B in which $b_{11} = 0$ and all other b_{ij} are i.i.d. exponential with mean 1, the smallest 2-assignment is $G(2, m, n) = (mn - 1)/[mn(m - 1)(n - 1)]$.

Any 2-assignment of the B matrix must involve at least one entry from rows $i \neq 1$. The minimum such entry d has expected value $1/[(m - 1)n]$. With probability $(n - 1)/n$, this entry occurs outside the first column, so that $0 + d = d$ is the smallest assignment.

With probability $1/n$, the entry occurs in the first column. But some entry in the minimum 2-assignment must come from a column other than the first. So in this case we construct the $m \times (n-1)$ matrix $c = \{c_{ij}, 1 \leqslant i \leqslant m, 2 \leqslant j \leqslant m\}$, where

$$c_{ij} = \begin{cases} b_{ij} & \text{if } i = 1, j \in \{2, 3, \ldots, n\} \\ b_{ij} - d & \text{if } i \neq 1, j \in \{2, 3, \ldots, n\} \end{cases}$$

Again the entries c_{ij} are i.i.d. exponential with mean 1, so the minimum element $e = c_{ij}$ has expected value $1/[m(n - 1)]$. If this minimum occurs in row $k = 1$, then the minimal 2-assignment in the B matrix is $d + e$. (The two entries d and e are a valid 2-assignment, and no 2-assignment can be

smaller.) If the minimum occurs in row $k \neq 1$, then the minimum 2-assignment of the B matrix is $b_{11} + b_{ij} = (0) + (d+e)$. Summing,

$$G(2,m,n) = \mathbb{E}(d) + \frac{1}{n}\mathbb{E}(e) = \frac{1}{(m-1)n} + \frac{1}{nm(n-1)} = \frac{(mn-m)+(m-1)}{mn(m-1)(n-1)} = \frac{mn-1}{mn(m-1)(n-1)}$$

and

$$F(2,m,n) = G(2,m,n) + \frac{2}{mn} = \frac{3mn-2m-2n+1}{mn(m-1)(n-1)} = \frac{1}{mn} + \frac{1}{m(n-1)} + \frac{1}{(m-1)n}.$$

Case $k = m = 3$:

Let a_{i,h_i} be the least element in row i, for $i = 1,2,3$. We have three possibilities.

With probability $(n-1)(n-2)/n^2$, *all the columns* h_i *are distinct,* and the assignment is just $\sum a_{i,h_i}$. In this case the expected value is clearly $3/n$. (The values of the row minima are independent of their relative positions.)

With probability $3(n-1)/n^2$, *two of the columns agree and one is different.* Define a new array b_{ij} with $b_{ij} = a_{ij} - a_{i,h_i}$. Without loss of generality, assume $h_1 = h_2 = 1$ and $h_3 = 2$. The values b_{ij} are i.i.d. exponential, except $b_{11} = b_{21} = b_{32} = 0$. The least 3-assignment of A differs from that of B by $a_{11} + a_{21} + a_{32}$, whose expected value is $3/n$.

Since column 1 has two zeros and row 3 has one zero, we can use Lemma 14 to insert a zero, and set $b_{31} = 0$. An optimal 3-assignment on B is then obtained from an optimal 2-assignment on the $3 \times (n-1)$ array $\{b_{ij} | j = 2,3,\ldots,n\}$, augmented with a zero from the first column and the unique row not used in the 2-assignment. So

$$\begin{aligned}
\mathbb{E}B^\star &= G(2,3,n-1) \\
&= F(2,3,n-1) - \frac{2}{3(n-1)} \\
&= \frac{1}{3(n-1)} + \frac{1}{2(n-1)} + \frac{1}{3(n-2)} - \frac{2}{3(n-1)} \\
&= \frac{3n-4}{6(n-1)(n-2)},
\end{aligned}$$

and the optimal assignment on A has expected value $3/n$ larger:

$$\mathbb{E}A^\star = \frac{3}{n} + \frac{3n-4}{6(n-1)(n-2)}.$$

With probability $1/n^2$ *all three columns are the same.* Let this column be column 1. Again we subtract the row minima a_{i1} from each row to obtain a $3 \times n$ array $b_{ij} = a_{ij} - a_{i1}$. We want a minimal 2-assignment from the $3 \times (n-1)$ array $\{b_{ij} \mid j = 2,3,\ldots,n\}$; we will augment that with the value $b_{i1} = 0$ from the first column and the unique unused row. So, appealing to the previously proved case, the expected value of the 3-assignment of the b matrix is

$$F(2,3,n-1) = \frac{1}{3(n-1)} + \frac{1}{2(n-1)} + \frac{1}{3(n-2)} = \frac{7n-12}{6(n-1)(n-2)},$$

and that of the a matrix is

$$\frac{3}{n} + \frac{7n-12}{6(n-1)(n-2)}.$$

Summing over the three cases, the expected value is

$$\left(\frac{(n-1)(n-2)}{n^2}\right)\cdot\left(\frac{3}{n}\right) + \left(\frac{3(n-1)}{n^2}\right)\cdot\left(\frac{3}{n} + \frac{3n-4}{6(n-1)(n-2)}\right) + \left(\frac{1}{n^2}\right)\cdot\left(\frac{3}{n} + \frac{7n-12}{6(n-1)(n-2)}\right).$$

One can check algebraically that this is equal to

$$\frac{1}{3(n)} + \frac{1}{2(n)} + \frac{1}{1(n)} + \frac{1}{3(n-1)} + \frac{1}{2(n-1)} + \frac{1}{3(n-2)}.$$

Case $k = m = n = 4$:

Subtract the row minimum from each row, incurring an expected cost 4/4=1, and leaving an array b_{ij} of random variables, four of which are 0, and the rest of which are i.i.d. exponential. We will break into subcases, depending on the relative positions of the row minima. For each subcase we will develop three numbers: P, the probability of this case occurring (including symmetry); L, the expected cost of removing row minima and column minima; and R, the expected cost of the residual array b.

Subcase 1: All row minima in the same column. Say $b_{11} = b_{21} = b_{31} = b_{41} = 0$. The probability of such an arrangement is $P = (1/4)^3 = 1/64$. An optimal 4-assignment on b is obtained from an optimal 3-assignment on (b without its first column) by adding a 0 entry from the first column. We have $P = 1/64$, $L = 4/4$, and (by the previously proved case) $R = F(3,3,4) = 65/72$.

Subcase 2: Three row minima in one column, the fourth in another. Say $b_{11} = b_{21} = b_{31} = b_{42} = 0$. We calculate P: There are $\binom{4}{3} = 4$ ways to select the three rows whose columns agree. Each way has probability $(1/4)^2(3/4)$ because two specified rows agree with the first, and the last disagrees. So $P = 4(1/4)^2(3/4) = 12/64$. Use Lemma 14 to insert a zero: $b_{41} = 0$. Find an optimal 3-assignment on b without its first column. We have $P = 12/64$, $L = 4/4$, and $R = G(3,3,4) = 65/72 - 3/12 = 47/72$.

Subcase 3: Two row minima in one column, two in another. Say $b_{11} = b_{21} = b_{32} = b_{42} = 0$. The probability is $3(1/4)(3/4)(1/4) = 9/64$. We will subtract column minima from the third and fourth columns; this incurs a cost of 1/4 for each column, giving $L = 6/4$. We call the resulting array c. We further break into subcases depending on the positions of the column minima.

Subcase 3a: Two column minima on one row. Say $c_{13} = c_{14} = 0$. The net probability of this case is $P = (1/4)(9/64) = 9/256$. Use Lemma 14 to add zeros, successively: $c_{12} = 0$, then $c_{22} = 0$. The residual cost is that of a 2-assignment on the 3×3 array obtained by deleting the first row and second column from c; this array has one 0 element, so the cost is $G(2,3,3) = 2/9$. This subcase has $P = 9/256$, $L = 6/4$, and $R = 2/9$.

Subcase 3b: Two column minima in first and second rows. Say $c_{13} = c_{24} = 0$. (The third and fourth rows could also be used.) The probability of this case is $P = (1/4)(9/64) = 9/256$. Use Lemma 14 to insert zeros, successively, at positions c_{12}, c_{22}, c_{14} and c_{23}. The residual cost is the minimum 1-assignment (single element) from the 2×3 array gotten by deleting the first two rows and the second column from c. Thus we have $P = 9/256$, $L = 6/4$, and $R = F(1,2,3) = 1/6$.

Subcase 3c: Two column minima in first and third rows. Say $c_{13} = c_{34} = 0$. This case includes one minimum in either the first or second row and the other in the third or fourth row. Its probability is $P = (2/4)(9/64) = 18/256$, and $L = 6/4$. Its residual cost is 0, since $c_{13} = c_{21} = c_{34} = c_{42} = 0$ is an optimal assignment. So $P = 18/256$, $L = 6/4$, and $R = 0$.

Subcase 4: Two row minima in one column, the others in two different columns. Say $b_{11} = b_{21} = b_{32} = b_{43} = 0$. The probability of such an arrangement is $6(1/4)(2/4)(3/4) = 36/64$. Again we subtract the column minimum from the fourth column, and break into two subcases depending on its position.

Subcase 4a: Column minimum in first (or second) row. Say $c_{14} = 0$. This has probability $P = (2/4)(36/64) = 18/64$. The residual cost is 0 because $c_{14} = c_{21} = c_{32} = c_{43} = 0$. So $P = 18/64$, $L = 5/4$, $R = 0$.

Subcase 4b: Column minimum in third (or fourth) row. Say $c_{34} = 0$. This has probability $P = (2/4)(36/64) = 18/64$. Use Lemma 14 to insert zeros, successively, at c_{31}, c_{41}, and c_{33}. We

need a 2-assignment from the 3×3 array obtained by deleting the first column and third row from c; this has one 0 at $c_{43} = 0$, so its residual cost is $G(2,3,3) = 2/9$. So $P = 18/64$, $L = 5/4$, $R = 2/9$.

Subcase 5: All row minima in distinct columns. For example, $b_{11} = b_{22} = b_{33} = b_{44} = 0$. The optimal assignment is 0, and no further work is required. We have $P = (3/4)(2/4)(1/4) = 6/64$, $L = 4/4$, and $R = 0$.

The overall expected cost is computed as

$$F(4,4,4) = \sum P_i(L_i + R_i) = \frac{1}{1} + \frac{1}{4} + \frac{1}{9} + \frac{1}{16}.$$

This concludes the proof of Theorem 15. □

Theorem 16. *With $k = m = n$, Conjecture 12 reduces to Parisi's conjecture that $F(n) = \sum_{i=1}^{n} \frac{1}{i^2}$.*

Proof. With $S_n = \sum_{i,j \geq 0,\ i+j<n} \frac{1}{(n-i)(n-j)}$, the inductive proof centers on showing that $S_{n+1} - S_n = 1/(n+1)^2$. □

Acknowledgements

Greg Sorkin is very grateful to Boris Pittel: for introducing the problem to me, for being constantly enthusiastic, and for failing to disclose the extent of previous work before I was completely hooked. Both authors thank David Williamson for helpful discussions.

References

[Ald92] David Aldous. Asymptotics in the random assignment problem. *Pr. Th. Related Fields*, 93:507–534, 1992.

[CS98] Don Coppersmith and Gregory B. Sorkin. Constructive bounds and exact expectations for the random assignment problem. Technical Report RC21133 (94490), IBM Reseach Report, Yorktown Heights, NY, March 1998. Accessible at `http://domino.watson.ibm.com/library/CyberDig.nsf/home`.

[DFM86] Martin E. Dyer, Alan M. Frieze, and Colin J.H. McDiarmid. On linear programs with random costs. *Mathematical Programming*, 35:3–16, 1986.

[Don69] W.E. Donath. Algorithm and average-value bounds for assignment problems. *IBM Journal of Research and Development*, 13(4):380–386, July 1969.

[GK93] Michel X. Goemans and Muralidharan S. Kodialam. A lower bound on the expected cost of an optimal assignment. *Mathematics of Operations Research*, 18(2):267–274, May 1993.

[Kar84] R.M. Karp. An upper bound on the expected cost of an optimal assignment. Technical report, Computer Science Division, University of California, Berkeley CA, 1984.

[Kar87] Richard M. Karp. An upper bound on the expected cost of an optimal assignment. In David S. Johnson et al., editors, *Discrete Algorithms and Complexity: Proceedings of the Japan-US Joint Seminar*, volume 15 of *Perspectives in Computing*, pages 1–4. Academic Press, 1987.

[KKV94] Richard M. Karp, Alexander H.G. Rinnooy Kan, and Rakesh V. Vohra. Average case analysis of a heuristic for the assignment problem. *Mathematics of Operations Research*, 19(3):513–522, August 1994.

[Laz79] Andrew J. Lazarus. The assignment problem with uniform (0,1) cost matrix. Master's thesis, Department of Mathematics, Princeton University, Princeton, N.J., 1979.

[Laz93] Andrew J. Lazarus. Certain expected values in the random assignment problem. *Oper. Res. Lett.*, 14(4):207–214, 1993.

[McD89] Colin McDiarmid. On the method of bounded differences. In *London Mathematical Society Lecture Note Series*, volume 141, pages 148–188. Cambridge University Press, 1989.

[MP85] M. Mézard and G. Parisi. Replicas and optimization. *J. Physique Lettres*, 46:771–778, September 1985.

[MP87] M. Mézard and G. Parisi. On the solution of the random link matching problems. *J. Physique Lettres*, 48:1451–1459, September 1987.

[Oli92] Birgitta Olin. *Asymptotic Properties of Random Assignment Problems.* PhD thesis, Kungl Tekniska Högskolan, Stockholm, Sweden, 1992.

[Par98] Giorgio Parisi. A conjecture on random bipartite matching. Physics e-Print archive, http://xxx.lanl.gov/ps/-cond-mat/9801176, January 1998.

[PW98] Boris Pittel and Robert S. Weishaar. The random bipartite nearest neighbor graphs. Submitted for publication, 1998.

[Wal79] David W. Walkup. On the expected value of a random assignment problem. *SIAM J. Computing*, 8(3):440–442, August 1979.

The "Burnside Process" Converges Slowly*

Leslie Ann Goldberg[1] and Mark Jerrum[2]

[1] Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom,
leslie@dcs.warwick.ac.uk,
http://www.dcs.warwick.ac.uk/~leslie

[2] Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom,
mrj@dcs.ed.ac.uk,
http://www.dcs.ed.ac.uk/~mrj

Abstract. We consider the problem of sampling "unlabelled structures", i.e., sampling combinatorial structures modulo a group of symmetries. The main tool which has been used for this sampling problem is Burnside's lemma. In situations where a significant proportion of the structures have no non-trivial symmetries, it is already fairly well understood how to apply this tool. More generally, it is possible to obtain nearly uniform samples by simulating a Markov chain that we call the Burnside process; this is a random walk on a bipartite graph which essentially implements Burnside's lemma. For this approach to be feasible, the Markov chain ought to be "rapidly mixing", i.e., converge rapidly to equilibrium. The Burnside process was known to be rapidly mixing for some special groups, and it has even been implemented in some computational group theory algorithms. In this paper, we show that the Burnside process is not rapidly mixing in general. In particular, we construct an infinite family of permutation groups for which we show that the mixing time is exponential in the degree of the group.

1 Introduction

The computational task considered in this article is that of sampling "unlabelled structures", i.e., sampling combinatorial structures modulo a group of symmetries. We work within the framework of Pólya theory: "Structures" are taken to be length-m words over a finite alphabet Σ, and the group of symmetries is taken to be a permutation group G of degree m which acts on the words by permuting positions. (See Section 2 for precise definitions.) The image of $\alpha \in \Sigma^m$ under g is conventionally denoted α^g. Words α and β are in the same *orbit* if there is a permutation $g \in G$ which maps α to $\alpha^g = \beta$. The orbits partition the set of words into equivalence classes, and the computational problem is to sample words in such a way that each orbit is equally likely to be output.[1]

* This work was supported in part by ESPRIT Projects RAND-II (Project 21726) and ALCOM-IT (Project 20244), and by EPSRC grant GR/L60982.

[1] Here is a concrete example: Let Σ be a binary alphabet. Encode the adjacency matrix of an n-vertex graph as a word of length $m = \binom{n}{2}$. The relevant permutation group

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 331-345, 1998.

The main tool which has been used for sampling orbits is *Burnside's Lemma*,[2] which says that each orbit comes up $|G|$ times (as the first component) in the set of pairs

$$\Upsilon(\Sigma, G) := \{(\alpha, g) \mid \alpha \in \Sigma^m, g \in G \text{ and } \alpha^g = \alpha\}. \tag{1}$$

Thus, we are interested in the computational problem of sampling uniformly at random from $\Upsilon(\Sigma, G)$, given (an efficient representation of) G.

Wormald [17] has shown how to solve this sampling problem for *rigid* structures. That is, he has given an efficient random sampling algorithm that works whenever a high fraction of the pairs in $\Upsilon(\Sigma, G)$ have g equal to the identity permutation. Wormald's method does not extend to the case in which the identity permutation contributes only a small fraction[3] of the pairs in $\Upsilon(\Sigma, G)$. However, Jerrum proposed a natural approach based on Markov chain simulation which does extend to this case [8].

We give the details of the Markov chain simulation approach in Section 2. In brief, the idea is to consider the following bipartite graph: The vertices on the left-hand side are all words in Σ^m. The vertices on the right-hand side are all permutations in G. There is an edge from word α to permutation g if and only if $\alpha^g = \alpha$. This graph essentially implements Burnside's Lemma: The lemma shows that the stationary distribution of a random walk on the graph assigns equal weight to each orbit, i.e., to each unlabelled structure. The Markov chain that we consider, which we refer to as the "Burnside process", is the random walk on this graph observed on alternate steps.

We may obtain a nearly uniform unlabelled sample by simulating the Burnside process from a fixed initial state for sufficiently many steps, and returning the final state. The efficiency of this sampling method is dependent on the so-called mixing time of the Burnside process: in rough terms, how many steps is "sufficiently many"? The aim of this article is to show that the mixing time of the Burnside process is sometimes very large. We now make that statement precise.

For any two probability distributions π and π' on a finite set Ψ, define the *total variation distance* between π and π' to be

$$D_{\mathrm{tv}}(\pi, \pi') := \max_{A \subseteq \Psi} |\pi(A) - \pi'(A)| = \frac{1}{2} \sum_{x \in \Psi} |\pi(x) - \pi'(x)|.$$

Suppose M is an ergodic Markov chain with state space Ψ and stationary distribution π, and let the t-step distribution of M, when started in state x_0, be π_t. The

is the group (acting on words) which is induced by the group of all permutations of the n vertices. Note that two graphs are in the same orbit if and only if they are isomorphic.

[2] Although this lemma is commonly referred to as "Burnside's Lemma", it is really due to Cauchy and Frobenius [14].

[3] Specifically, Wormald's approach can be used when the fraction of pairs in $\Upsilon(\Sigma, G)$ which are due to the identity is at least the inverse of some polynomial in m.

mixing time of M, given initial state x_0, is a function $\tau_{x_0} : (0,1) \to \mathbf{N}$, from tolerances δ to simulation times, defined as follows: for each $\delta \in (0,1)$, let $\tau_{x_0}(\delta)$ be the smallest t such that $D_{\mathrm{tv}}(\pi_{t'}, \pi) \leq \delta$ for all $t' \geq t$. If the initial state is not significant or is unknown, it is appropriate to define $\tau(\delta) = \max_x \tau_x(\delta)$, where the maximum is over all $x \in \Psi$. By *rapid mixing*, we mean that $\tau(\delta) \leq \mathrm{poly}(m, \log \delta^{-1})$, where m is the input size—in our case the degree of the group G—and δ the tolerance. Stuart Anderson has suggested the phrase *torpid mixing* to describe the contrasting situation where mixing time is exponential in the input size.

The Burnside process was shown to be rapidly mixing for some very special groups G [8]. However, it was an open question whether it is rapidly mixing in general. The precise result of this article (Theorem 1) is a construction of an infinite family of permutation groups G for which we show that the mixing time $\tau(\frac{1}{3})$ is exponential in the degree of G. Thus, if we use the t-step distribution to estimate the probability $\pi(A)$ of some event $A \subset \Psi$ in the stationary distribution, the result may be out by as much as $\frac{1}{3}$, unless we take t exponentially large.

The main idea of the proof is to relate the mixing time of the Burnside process to the "Swendsen-Wang process", a particular dynamics for the Potts model in statistical physics. The Swendsen-Wang process was shown by Gore and Jerrum [6] to have exponential mixing time at a certain critical value of a parameter called "temperature". It turns out that the Swendsen-Wang process defined on a graph , at a different (lower, non-critical) temperature has exactly the same dynamics as the Burnside process on a derived permutation group $G_3(,)$. Thus we only have to relate the Swendsen-Wang process at the two different temperatures, which we do using the "l-stretch" construction used by other authors [7]. The dynamics of the Swendsen-Wang process is not perfectly preserved by the l-stretch construction, but the correspondence is close enough to yield the claimed result.

Sections 2 and 3 describe the Burnside and Swendsen-Wang processes; Section 4 describes the relationship between the two; Section 5 relates the Swendsen-Wang process at two different temperatures via the l-stretch construction, thus completing the "torpid mixing" proof; finally, Section 6 concludes with some open problems.

2 The Burnside process

Let $\Sigma = \{0, \ldots, k-1\}$ be a finite alphabet of cardinality k, and G a permutation group on $[m] = \{0, \ldots, m-1\}$. For $g \in G$ and $i \in [m]$, denote by i^g the image of i under g. The group G has a natural action on the set Σ^m of all words of length m over the alphabet Σ, induced by permutations of the "positions" $0, \ldots, m-1$. Under this induced action, the permutation $g \in G$ maps the word $\alpha = a_0 a_1 \ldots a_{m-1}$ to the word $\alpha^g = \beta = b_0 b_1 \ldots b_{m-1}$ defined by $b_j = a_i$ for all $i, j \in [m]$ satisfying $i^g = j$. The action of G partitions Σ^m into a number of *orbits*, these being the equivalence classes of Σ^m under the equivalence relation that identifies α and β whenever there exists $g \in G$ mapping α to β. The orbit $\{\alpha^g : g \in G\}$ containing the word $\alpha \in \Sigma^m$ is denoted α^G. As we indicated in

the introduction, Burnside's Lemma says that each orbit comes up $|G|$ times in the set $\Upsilon(\Sigma, G)$ defined in equation (1). Thus, we are interested in the problem of uniformly sampling elements of $\Upsilon(\Sigma, G)$.

A standard attack on combinatorial sampling problems [10] is to design a Markov chain whose states are the structures of interest (in this case the state space is G) and whose transition probabilities are chosen so that the stationary distribution is the required sampling distribution. The following natural Markov chain was proposed by Jerrum [8]. As we noted in the introduction, it is essentially a random walk on the bipartite graph which corresponds to Burnside's Lemma. The state space of the Markov chain $M_{\mathrm{B}} = M_{\mathrm{B}}(G, \Sigma)$ is just G. The transition probabilities from a state $g \in G$ are specified by the following conceptually simple two-step experiment:

(B1) Sample α uniformly at random (u.a.r.) from the set $\operatorname{Fix} g := \{\alpha \in \Sigma^m : \alpha^g = \alpha\}$.
(B2) Sample h u.a.r. from the point stabiliser $G_\alpha := \{h \in G : \alpha^h = \alpha\}$.

The new state is h. Algorithmically, it is not difficult to implement (B1). However, Step (B2) is apparently difficult in general. (It is equivalent under randomised polynomial-time reductions to the *Setwise Stabiliser* problem, which includes *Graph Isomorphism* as a special case.) Nevertheless, there are significant classes of groups G for which an efficient (polynomial time) implementation exists. Luks has shown that p-groups—groups in which every element has order a power of p for some prime p—is an example of such a class [11].

Returning to the Markov chain itself, we note immediately that M_{B} is ergodic, since every state (permutation) can be reached from every other in a single transition, by selecting the word $\alpha = 0^m$ in step (B1). Let $\pi : G \to [0, 1]$ denote the stationary distribution of M_{B}. Then $\pi(g)$ is proportional to the degree of vertex g in the bipartite graph corresponding to Burnside's Lemma, which is $|\operatorname{Fix} g| = k^{c(g)}$, where $c(g)$ denotes the number of cycles in the permutation g. We have therefore established the following Lemma from [9]:

Lemma 1. *Let π be the stationary distribution of the Markov chain $M_{\mathrm{B}}(G, \Sigma)$. Then $\pi(g) = k^{c(g)}/|\Upsilon(\Sigma, G)|$ for all $g \in G$.*

Although the Markov chain M_{B} on G is the most convenient one for us to work with, it is clear that we can invert the order of steps (B1) and (B2) to obtain a dual Markov chain $M'_{\mathrm{B}}(G, \Sigma)$ with state space Σ^m. The dual Markov chain[4] has greater practical appeal, as it gives a uniform sampler for orbits (i.e., unlabelled structures):

Lemma 2. *Let π' be the stationary distribution of the Markov chain $M'_{\mathrm{B}}(G, \Sigma)$. Then*

$$\pi'(\alpha) = \frac{|G|}{|\alpha^G|\,|\Upsilon(\Sigma, G)|}$$

for all $\alpha \in \Sigma^m$; in particular, π' assigns equal probability to each orbit α^G.

[4] In references [8] and [9], the primed and unprimed versions are reversed.

The result again follows from a consideration of the random walk on the bipartite graph, using the elementary group-theoretic fact that $|G_\alpha| \times |\alpha^G| = |G|$.

Peter Cameron has observed that a Markov chain similar to M'_{B} may be defined for any group action, not just the special case of a permutation group G acting on Σ^m by permutation of positions. In the general setting: given a point α, select u.a.r. a group element g that fixes α, and then select a point that is fixed by g. Thus, the generalisation of M'_{B} to arbitrary group actions provides a potentially efficient procedure for uniformly sampling unlabelled structures (i.e., sampling structures up to symmetry). This procedure has been implemented in certain algorithms for determining the conjugacy classes of a finite group [16].

Of course, the effectiveness of M'_{B} (equivalently M_{B}) as a basis for a general purpose sampling procedure for unlabelled structures depends on its mixing time. It was known that M_{B} mixes rapidly in some special cases (see Jerrum [8]), but it was not previously known whether M_{B} mixes rapidly for all groups G. Specifically, it was not known whether the mixing time of $M_{\mathrm{B}}(G, \Sigma)$ is uniformly bounded by a polynomial in m, the degree of G. The result in this article is a construction of an infinite family of permutation groups for which we show that the mixing time of M_{B} grows exponentially in the degree m.

3 The Swendsen-Wang process

As noted in the Introduction, our strategy is to relate the mixing time of the Burnside process to that of the Swendsen-Wang process. In this section we describe the latter process, which provides a particular dynamics for the q-state Potts model. In fact, we need only consider the special case $q = 3$. See Martin's book [13] for background on the Potts model.

A (3-state) Potts system is defined by a graph , $= (V, E)$ and a real number ("inverse temperature") β. For compactness, we will sometimes denote an edge $(i, j) \in E$ by ij. A *configuration* of the system is an assignment $\sigma : V \to \{0, 1, 2\}$ of "spins" or *colours* to the vertices of , . The set of all $3^{|V|}$ possible configurations is denoted by Ω. We associate each configuration $\sigma \in \Omega$ with an energy $H(\sigma) := \sum_{ij \in E} \big[1 - \delta(\sigma(i), \sigma(j))\big]$, where δ is the Kronecker-δ function which is 1 if its arguments are equal, and 0 otherwise. Thus the energy of a configuration is just the number of edges connecting unlike colours. The (Boltzmann) *weight* of a configuration σ is $\exp(-\beta\, H(\sigma))$. The *partition function* of the 3-state Potts model is

$$Z = Z(,\ ,\beta) := \sum_{\sigma \in \Omega} \exp\big(-\beta\, H(\sigma)\big); \tag{2}$$

it is the normalising factor in the *Gibbs distribution* on configurations, which assigns probability $\exp(-\beta\, H(\sigma))/Z$ to configuration σ. To avoid the exponentials, we will define the *edge weight* λ of the Potts system to be $\mathrm{e}^{-\beta}$, so the partition function (2) may be rewritten as

$$Z = Z(,\ ,\lambda) = \sum_{\sigma \in \Omega} \prod_{ij \in E} \lambda^{[1 - \delta(\sigma(i), \sigma(j))]}. \tag{3}$$

Thus the weight of a configuration is λ^b, where b is the number of bichromatic edges.

The Swendsen-Wang process specifies a Markov chain $M_{\mathrm{SW}}(\ ,\ ,\lambda)$ on Ω. Let the current Potts configuration be denoted by σ. The next configuration σ' is obtained as follows.

(SW1) Let $\overline{A} = \{ij \in E : \sigma(i) = \sigma(j)\}$ be the set of monochromatic edges. Select a subset $A \subseteq \overline{A}$ by retaining each edge in $\overline{A}$ independently with probability $p = 1 - \lambda$.

(SW2) The graph (V, A) consists of a number of connected components. For each connected component, a colour is chosen u.a.r. from $\{0, 1, 2\}$, and all vertices within the component are assigned that colour.

That the Markov chain with transitions defined by this experiment is ergodic is immediate; that it has the correct (i.e., Gibbs) distribution is not too difficult to show. (See, for example, Edwards and Sokal [4].) Both the Swendsen-Wang process ($M_{\mathrm{SW}}(\ ,\ ,\lambda)$) and the Burnside process ($M_{\mathrm{B}}(G, \Sigma)$) are examples of Markov chains using the "method of auxiliary variables" (see [4] and [1]).

4 The relationship between the Burnside process and the Swendsen-Wang process

Let Σ be a finite alphabet of size k, and let $\ ,\ = (V, E)$ be an undirected graph defining a 3-state Potts system with edge weight $\lambda = k^{-2}$. We will construct an associated permutation group $G_3(\ ,\ )$ such that the dynamics of the Burnside process on $(G_3(\ ,\ ), \Sigma)$ is essentially the same as the Swendsen-Wang dynamics on $(\ ,\ ,\lambda)$. This construction generalises a construction from [8], which deals with the case $k = 2$ (i.e., the binary alphabet case).

The permutation group $G_3(\ ,\ )$ acts on the set $\Delta = \bigcup_{e \in E} \Delta_e$, which is the disjoint union of three-element sets Δ_e. Arbitrarily orient the edges of $\ ,\ $, so that each edge $e \in E$ has a defined start-vertex e^- and end-vertex e^+. For $e \in E$ and $v \in V$, denote by h_e some fixed permutation that induces a 3-cycle on Δ_e and leaves everything else fixed, and denote by g_v the generator

$$g_v := \prod_{e:e^+=v} h_e \times \prod_{e:e^-=v} h_e^{-1}.$$

Finally, define $G_3(\ ,\ ) = \langle g_v : v \in V \rangle$, the group generated by $\{g_v\}$.

Observe that the generators of $G_3(\ ,\ )$ commute and have order three, so each permutation $g \in G_3(\ ,\ )$ can be expressed as

$$g = g(\sigma) := \prod_{v \in V} {g_v}^{\sigma(v)} = \prod_{e \in E} {h_e}^{\sigma(e^+) - \sigma(e^-)}, \tag{4}$$

where $\sigma : V \to \{0, 1, 2\}$. Provided the graph $\ ,\ $ is connected, this expression is essentially canonical, in that σ is uniquely determined up to addition (mod 3) of a constant function. To see this, note that g uniquely determines the exponent

of h_e in expansion (4), which in turn determines the difference between the colours (viewed as integers) at the endpoints of edge e. Note that all three of the configurations associated with g induce the same set $\overline{A}$ in (SW1). Thus, the transition probabilities from the three configurations are the same, and we can therefore think of g as being associated with all three configurations.

Lemma 3. *Suppose , is a graph, Σ a finite alphabet, and let $k = |\Sigma|$. Then*

$$M_{\mathrm{B}}(G_3(,\ ),\Sigma) \cong M_{\mathrm{SW}}(,\ ,k^{-2});$$

that is to say, each permutation g in the state space of $M_{\mathrm{B}}(G_3(,\ ),\Sigma)$ can be associated with exactly three configurations in the state space of $M_{\mathrm{SW}}(,\ ,k^{-2})$ in such a way that transition probabilities are preserved.

Proof. We associate each permutation $g \in G_3(,\ )$ with three configurations as described above. As we observed, the transition probabilities of the three configurations in SW are identical.

Perhaps the easiest way to show that these transition probabilities are the same as those in M_{B} is to combine the experiment defining the Burnside process (see (B1) and (B2)) with that defining the Swendsen-Wang process (see (SW1) and (SW2)) into a single coupled version. Start with the pair (g,σ_g), where σ_g is one of the three configurations associated with g.

(C1) Sample α u.a.r. from the set $\mathrm{Fix}\, g = \{\alpha \in \Sigma^m : \alpha^g = \alpha\}$ of words fixed by g. Let $A := \{e \in E : \alpha$ is *not* constant on $\Delta_e\}$. The pair (α, A) is the intermediate state.

(C2) Sample h u.a.r. from the point stabiliser $G_\alpha = \{h \in G : \alpha^h = \alpha\}$.

The new pair is (h,σ_h) (again, choose σ_h arbitrarily from the three configurations associated with h.).

By construction, the transitions $g \to \alpha \to h$ occur with the probabilities dictated by (B1) and (B2). We must check that the induced transitions $\sigma_g \to A \to \sigma_h$ match (SW1) and (SW2) in probability. Let $e = uv \in E$ be any edge, and consider the action of g on Δ_e. If $\sigma_g(u) = \sigma_g(v)$ then the action of g on Δ_e is the identity, and probability that α is constant on Δ_e is k^{-2}. Thus the probability that $e \in A$ is $1 - k^{-2}$, independent of the other edge choices, as required by (SW1), where $\lambda = k^{-2}$. Otherwise, $\sigma_g(u) \neq \sigma_g(v)$ and the action of g on Δ_e is a 3-cycle. Necessarily, α is constant on Δ_e, and $e \notin A$, again as required by (SW1). So the distribution of $A \subseteq E$ is correct.

To verify the second step, again let $e = uv \in E$ be any edge. If $e \in A$ then α is not constant on Δ_e, entailing that the action of h on Δ_e is the identity and $\sigma_h(u) = \sigma_h(v)$. Conversely, if $e \notin A$ then α is constant on Δ_e, and $\sigma_h(u) - \sigma_h(v)$ is unconstrained. Thus $h \mapsto \sigma_h$ is a bijection from G_α to configurations that are constant on connected components of (V, A), and the distribution of σ_h is as demanded by (SW2).

5 Torpid mixing

We have seen that the Burnside process is equivalent to the Swendsen-Wang process at a particular edge-weight λ; and it is known that the Swendsen-Wang process at a *different* edge weight (which is approximately $1-(4\ln 2)/|V|$, where V is the vertex set of ,) has exponential mixing time [6]. In this section we bridge the gap between the different edge weights.

Denote by P_l the path of length l or *l-path*, i.e., the graph with vertex set $[l+1]$ and edge set $\{\{i,i+1\} : 0 \le i < l\}$.

Lemma 4. *Consider a randomly sampled configuration of the 3-state Potts model on P_l with edge weight λ. The induced distribution of colours on the two end vertices of P_l is identical to the distribution of configurations of the 3-state Potts model on P_1 $(= K_2)$ with edge weight*

$$\hat{\lambda}(l) := \frac{(1+2\lambda)^l - (1-\lambda)^l}{(1+2\lambda)^l + 2(1-\lambda)^l}. \tag{5}$$

Proof. Define $w^{(l)} \in \mathbf{R}^2$ to be the vector whose first (respectively, second) component $w_0^{(l)}$ (respectively, $w_1^{(l)}$) is the total weight of those configurations on P_l whose (ordered) endpoints have colours $(0,0)$ (respectively, $(0,1)$). Clearly, there is nothing special in the particular choice of colours; the pair $(0,0)$ could be replaced by any pair of like colours, and $(0,1)$ by any pair of unlike ones. Introduce the matrix

$$T := \begin{pmatrix} 1 & 2\lambda \\ \lambda & 1+\lambda \end{pmatrix};$$

a straightforward induction on l establishes

$$w^{(l)} = T^l \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The matrix T has eigenvalues $1-\lambda$ and $1+2\lambda$. Introduce two further matrices

$$D := \begin{pmatrix} 1-\lambda & 0 \\ 0 & 1+2\lambda \end{pmatrix} \quad \text{and} \quad S := \begin{pmatrix} 2 & 1 \\ -1 & 1 \end{pmatrix}.$$

Then $T = SDS^{-1}$ and hence $T^l = SD^lS^{-1}$. Noting that

$$S^{-1} = \frac{1}{3}\begin{pmatrix} 1 & -1 \\ 1 & 2 \end{pmatrix},$$

we obtain

$$w^{(l)} = SD^lS^{-1}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} (1+2\lambda)^l + 2(1-\lambda)^l \\ (1+2\lambda)^l - (1-\lambda)^l \end{pmatrix}. \tag{6}$$

Since P_l is equivalent—in the sense of the statement of the lemma—to a single edge with effective weight $w_1^{(l)}/w_0^{(l)}$, Lemma 4 follows immediately.

Denote by $K_n \otimes P_l$ the graph obtained from the complete graph on n vertices by subdividing each edge by $l-1$ intermediate vertices of degree two. Thus each edge of K_n becomes in $K_n \otimes P_l$ a copy of the l-path P_l. We refer to the vertices of degree $n-1$ as *exterior* vertices and those of degree two as *interior*. (Assume $n > 3$ to avoid trivialities.) We remark that this construction is just the "l-stretch", used in related situations by Jaeger, Vertigan and Welsh [7]. The l-stretch operation allows us to move between different edge weights, at least if we forget for a moment the specific dynamics imposed by the Swendsen-Wang process.

Lemma 5. *Consider a randomly sampled configuration of the 3-state Potts model on $K_n \otimes P_l$ with edge weight λ. The induced distribution of colours on the exterior vertices of $K_n \otimes P_l$ is identical to the distribution of configurations of the 3-state Potts model on K_n with edge weight $\hat{\lambda}$, where $\hat{\lambda} = \hat{\lambda}(l)$ is as in (5).*

Proof. Suppose σ is any Potts configuration on the graph $K_n \otimes P_l$, and S is any subset of its vertices. Denote by $\sigma|_S \in \{0,1,2\}^{|S|}$ the restriction of σ to the set S. Through some elementary algebraic manipulation, we may express the partition function of a Potts system on $K_n \otimes P_l$ in terms of the partition function of a Potts system on K_n with edge weight closer to 1. In the following manipulation, we assume that the vertices of $K_n \otimes P_l$ are numbered $0, \ldots, N-1$ and that the exterior vertices receive numbers in the range $0, \ldots, n-1$. Furthermore, $U_{ij} \subset [N]$ denotes the set of $l-1$ interior vertices lying on the l-path between exterior vertices i and j, and E_{ij} denotes the set of edges on that path.

$$
\begin{aligned}
&Z(K_n \otimes P_l, \lambda) \\
&= \sum_{\sigma} \prod_{uv \in E} \lambda^{[1-\delta(\sigma(u),\sigma(v))]} \\
&= \sum_{\sigma|_{[n]}} \sum_{\sigma|_{U_{0,1}}} \cdots \sum_{\sigma|_{U_{n-2,n-1}}} \left(\prod_{uv \in E_{0,1}} \lambda^{[1-\delta(\sigma(u),\sigma(v))]} \cdots \prod_{uv \in E_{n-2,n-1}} \lambda^{[1-\delta(\sigma(u),\sigma(v))]} \right) \\
&= \sum_{\sigma|_{[n]}} \left(\sum_{\sigma|_{U_{0,1}}} \prod_{uv \in E_{0,1}} \lambda^{[1-\delta(\sigma(u),\sigma(v))]} \right) \cdots \left(\sum_{\sigma|_{U_{n-2,n-1}}} \prod_{uv \in E_{n-2,n-1}} \lambda^{[1-\delta(\sigma(u),\sigma(v))]} \right) \\
&= \sum_{\sigma|_{[n]}} \prod_{0 \le i < j \le n-1} C\, \hat{\lambda}^{[1-\delta(\sigma(i),\sigma(j))]} \\
&= C^{n(n-1)/2}\, Z(K_n, \hat{\lambda}),
\end{aligned}
$$

where C is a constant (actually $w_0^{(l)}$). The penultimate equality above uses Lemma 4.

Let $\hat{\sigma} \in \{0,1,2\}^n$ be any configuration on K_n. From the above manipulation, we see that the weight of the configuration $\hat{\sigma}$ on K_n is equal—modulo the constant factor $C^{n(n-1)/2}$—to the sum of the weights of configurations σ of $K_n \otimes P_l$ that agree with $\hat{\sigma}$ on the exterior vertices or, symbolically, $\sigma|_{[n]} = \hat{\sigma}$. This proves Lemma 5.

Lemma 6. *There exists an infinite sequence of pairs* $(n,l) = \{(n(l),l) : l = 1,2,\ldots\}$ *such that*

$$\left|\left(1-\hat{\lambda}(l)\right) - \frac{4\ln 2}{n(l)}\right| \leq \frac{3}{n(l)^2}$$

for all pairs, where $\hat{\lambda}(l)$ *is defined as in (5).*

Proof. The function $1-\hat{\lambda}(l)$ decreases monotonically to 0, as $l \to \infty$. Given l, choose n to be the unique natural number satisfying

$$\frac{4\ln 2}{n(l)+1} < 1-\hat{\lambda}(l) \leq \frac{4\ln 2}{n(l)}.$$

The upper and lower bounds differ by less than $3n(l)^{-2}$. Thus, we have proved Lemma 6.

Let Ω be the set of configurations of the 3-state Potts model on $K_n \otimes P_l$. For each configuration $\sigma \in \Omega$, define $\gamma(\sigma) \in \mathbf{R}^3$ be the 3-vector whose ith component is the proportion of exterior vertices of $K_n \otimes P_l$ given colour i by σ. Then let $\Omega_{1:1:1}(\varepsilon)$ (respectively, $\Omega_{4:1:1}(\varepsilon)$) denote the set of configurations σ such that $\gamma(\sigma)$ lies within an ε-ball centred at $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ (respectively, one of the three ε-balls centred at $(\frac{2}{3},\frac{1}{6},\frac{1}{6})$, $(\frac{1}{6},\frac{2}{3},\frac{1}{6})$, or $(\frac{1}{6},\frac{1}{6},\frac{2}{3})$).

Lemma 7. *Let a configuration* σ *be sampled from the 3-state Potts model on* $K_n \otimes P_l$ *with edge weight* λ*, and suppose that* $1-\hat{\lambda}(l) = (4\ln 2)/n + O(n^{-2})$*. Then, for any* $\varepsilon > 0$*:*

(i) $\Pr(\sigma \in \Omega_{1:1:1}(\varepsilon)) = \Omega(n^{-2})$*;*
(ii) $\Pr(\sigma \in \Omega_{4:1:1}(\varepsilon)) = \Omega(n^{-2})$*; and*
(iii) $\Pr(\sigma \notin \Omega_{1:1:1}(\varepsilon) \cup \Omega_{4:1:1}(\varepsilon)) = \mathrm{e}^{-\Omega(n)}$.

The implicit constants depend only on ε*.*

Proof. By Lemma 4, we may equivalently work with the Potts model on K_n with edge weight $\hat{\lambda}(l)$.

When $1-\hat{\lambda}(l) = (4\ln 2)/n$, i.e., the error term is 0, this is precisely the result of Gore and Jerrum [6, Prop 3]. See also Bollobás, Grimmett and Janson [2]. The validity of the proof given in [6] is unaffected by the error term: an additive error $O(n^{-2})$ in $\hat{\lambda}(l)$ translates to an additive perturbation $O(n^{-1})$ in the function f in [6, eq. (2)]. This perturbation may be absorbed into the error term Δ appearing in that equation, which is $\Omega(1)$. Thus, we have proved Lemma 7.

We now need to compare the dynamics of the Swendsen-Wang processes on $K_n \otimes P_l$ and K_n, more precisely, the Markov chains $M_{\mathrm{SW}}(K_n \otimes P_l, \lambda)$ and $M_{\mathrm{SW}}(K_n, \hat{\lambda})$. The correspondence will not be exact, as in Lemma 3, but it will be close enough for our purposes.

Let $\mathcal{G}_{\nu,p}$ denote the standard random graph model in which an undirected ν-vertex graph is formed by adding, independently with probability p, for each

unordered pair of vertices (i, j), an edge connecting i and j. Suppose that $p < d/\nu$, with $d < 1$ a constant, and , is selected according to the model $\mathcal{G}_{\nu,p}$. It is a classical result that, with probability tending to 1 as $\nu \to \infty$, the connected components of , all have size $O(\log \nu)$. We require a (fairly crude) large deviation version of this result.

Lemma 8. *Let , be selected according to the model $\mathcal{G}_{\nu,p}$, where $p < d/\nu$ and $0 < d < 1$ is a constant. Then the probability that , contains a component of size exceeding $\sqrt{\nu}$ is $\exp(-\Omega(\sqrt{\nu}))$.*

Proof. This result in exactly this form appears as [6, Lemma 4]. See O'Connell [15, Thm 3.1] for a much more precise large-deviation result for the "giant component" of a sparse random graph.

We also need:

Lemma 9 (Hoeffding). *Let $Z_1, \ldots, Z_s$ be independent r.v's with $a_i \le Z_i \le b_i$, for suitable constants a_i, b_i, and all $1 \le i \le s$. Also let $\widehat{Z} = \sum_{i=1}^{s} Z_i$. Then for any $t > 0$,*

$$\Pr\left(|\widehat{Z} - \operatorname{Exp} \widehat{Z}| \ge t\right) \le \exp\Big(-2t^2 \Big/ \sum_{i=1}^{s} (b_i - a_i)^2\Big)$$

Proof. See McDiarmid [12, Thm 5.7].

Lemma 10. *Let a configuration $\sigma \in \Omega$ be sampled from the 3-state Potts model on $K_n \otimes P_l$ with edge weight λ, and suppose that $1 - \hat{\lambda}(l) = (4 \ln 2)/n + O(n^{-2})$. Let $\sigma' \in \Omega$ be the result of applying one step of the Swendsen-Wang process, starting at σ. Then, for any $\varepsilon > 0$,*

$$\Pr(\sigma' \in \Omega_{1:1:1}(\varepsilon) \mid \sigma \in \Omega_{1:1:1}(\varepsilon)) = 1 - \mathrm{e}^{-\Omega(\sqrt{n})},$$

and

$$\Pr(\sigma' \in \Omega_{4:1:1}(\varepsilon) \mid \sigma \in \Omega_{4:1:1}(\varepsilon)) = 1 - \mathrm{e}^{-\Omega(\sqrt{n})}.$$

The implicit constants depend only on ε.

Proof. For i, j exterior vertices of $K_n \otimes P_l$ satisfying $\sigma(i) = \sigma(j)$.

$$\Pr(\text{Path } i \leftrightarrow j \text{ is monochromatic}) = \frac{1}{w_0^{(l)}} = \frac{3}{(1+2\lambda)^l + 2(1-\lambda)^l},$$

where the second equality is from (6). After step (SW1),

$$\begin{aligned}
&\Pr(\text{Path } i \leftrightarrow j \text{ is contained in } A) \\
&\quad = \Pr(\text{Path } i \leftrightarrow j \text{ is monochromatic}) \times (1-\lambda)^l \\
&\quad = \frac{3(1-\lambda)^l}{(1+2\lambda)^l + 2(1-\lambda)^l} \\
&\quad = 1 - \hat{\lambda}(l).
\end{aligned}$$

For convenience, set $\hat{p} = 1 - \hat{\lambda}(l)$. Consider the set of exterior vertices of some given colour, and let $\nu \leq (\frac{1}{3} + \varepsilon)n$ be the size of that set. Provided ε is small enough ($\varepsilon = 1/40$ will do), $\hat{p}\nu \leq d < 1$. By Lemma 8, with probability $1 - \exp(-\Omega(\sqrt{\nu}))$, the maximum number of exterior vertices in any connected component of the graph $([N], A)$ restricted to this colour-class is at most $\sqrt{\nu}$. (Recall that $[N]$ is the vertex set of $K_n \otimes P_l$.) Combining this observation for all three colours, and noting $\nu = \Theta(n)$, we obtain the following: with probability $1 - \exp(-\Omega(\sqrt{n}))$, the number of external vertices in any connected component of $([N], A)$ is at most $\sqrt{n}$.

Let s be the number of such components, and $n_1, \ldots, n_s$ be their respective sizes. The expected size of a colour-class constructed in step (SW2) is $n/3$, and because there are many components (at least $\sqrt{n}$) we expect the actual size of each colour-class to be close to the expectation. We quantify this intuition by appealing to the Hoeffding bound. Fix a colour, say 0, and define the random variables $Y_1, \ldots, Y_s$ and $\widehat{Y}$ by

$$Y_i = \begin{cases} n_i, & \text{if the } i\text{th component receives colour 0 in step (SW2);} \\ 0, & \text{otherwise,} \end{cases}$$

and $\widehat{Y} = \sum_{i=1}^{s} Y_i$. Then $\operatorname{Exp} \widehat{Y} = n/3$ and, by Lemma 9, for any $t > 0$,

$$\begin{aligned} \Pr\left(|\widehat{Y} - \operatorname{Exp} \widehat{Y}| \geq t\right) &\leq \exp\Big(-2t^2 \Big/ \sum_{i=1}^{s} n_i^2\Big) \\ &\leq \exp(-2t^2 n^{-3/2}), \end{aligned}$$

since

$$\sum_{i=1}^{s} n_i^2 \leq \sum_{i=1}^{s} n_i \sqrt{n} = n^{3/2}.$$

Similar bounds apply, of course, to the other colours. Choosing $t = \varepsilon n/\sqrt{3}$ we see that, with probability $1 - \exp(-\Omega(\sqrt{n}))$, the size of every colour class in σ' lies in the range $\big((\frac{1}{3} - \varepsilon/\sqrt{3})n, (\frac{1}{3} + \varepsilon/\sqrt{3})n\big)$; but this condition implies $\sigma' \in \Omega_{1:1:1}(\varepsilon)$.

This proves the first part of Lemma 10, concerning $\Omega_{1:1:1}(\varepsilon)$; the second part of the lemma follows from the first by Lemma 7 and time-reversibility. In particular, it follows from the fact that M_{SW} satisfies the *detailed balance* condition:

$$\Pr(\sigma = \sigma_1 \wedge \sigma' = \sigma_2) = \Pr(\sigma = \sigma_2 \wedge \sigma' = \sigma_1),$$

for all configurations σ_1 and σ_2, where σ is sampled from the stationary distribution.

It is now a short step to the main theorem. Recall that $\tau(\frac{1}{3})$ denotes the number of steps t before the t-step distribution is within variation distance $\frac{1}{3}$ of the stationary distribution (maximised over the choice of starting state).

Theorem 1. *Let Σ be a finite alphabet of size at least two. There exists an infinite family of permutation groups G such that the mixing time of the Burnside process $M_{\mathrm{B}}(G, \Sigma)$ is exponential in the degree m of G; specifically $\tau(1/3) = \Omega(\exp(m^{1/(4+\varepsilon)}))$ for any $\varepsilon > 0$.*

Proof. By Lemma 3, it is enough to exhibit an infinite family of graphs , such that $M_{\mathrm{SW}}(, , \lambda)$ has exponential mixing time, where $\lambda = k^{-2}$. This family of graphs will of course be $(K_{n(l)} \otimes P_l : l \in \mathbf{N})$ where $n(l)$ is as defined in lemma 6. The family of permutation groups promised by the theorem will then be $(G_3(K_{n(l)} \otimes P_l) : l \in \mathbf{N})$.

Consider a trajectory $(\sigma_t : t \in \mathbf{N})$ of $M_{\mathrm{SW}}(K_n \otimes P_l, \lambda)$ starting in the stationary distribution. We say that the trajectory *escapes* at step t if

$$(\sigma_t \in \Omega_{1:1:1}(\varepsilon) \wedge \sigma_{t+1} \notin \Omega_{1:1:1}(\varepsilon)) \vee (\sigma_t \in \Omega_{4:1:1}(\varepsilon) \wedge \sigma_{t+1} \notin \Omega_{4:1:1}(\varepsilon)).$$

For each t, by Lemma 10, the probability of escape at time t is bounded by $\exp(-\Omega(\sqrt{n}))$. Furthermore, by Lemma 7 the probability of the event

$$\sigma_0 \notin \Omega_{1:1:1}(\varepsilon) \cup \Omega_{4:1:1}(\varepsilon)$$

is also bounded by $\exp(-\Omega(\sqrt{n}))$.

Thus there is a function $T = T(n) = \exp(\Omega(\sqrt{n}))$ such that, with probability at least $\frac{9}{10}$, the initial segment of the trajectory $(\sigma_t : 0 \le t \le T)$ lies either entirely within $\Omega_{1:1:1}(\varepsilon)$ or entirely within $\Omega_{4:1:1}(\varepsilon)$. Hence there is an initial state $s \in \Omega_{1:1:1}(\varepsilon)$ such that $\Pr(\sigma_T \notin \Omega_{1:1:1}(\varepsilon) \mid \sigma_0 = s) \le \frac{1}{10}$, and similarly for $s \in \Omega_{4:1:1}(\varepsilon)$. Choose such an initial state s from whichever of $\Omega_{1:1:1}(\varepsilon)$ or $\Omega_{4:1:1}(\varepsilon)$ has the smaller total weight in the stationary distribution. Then the variation distance of the T-step distribution from the stationary distribution is at least $\frac{1}{2} - e^{-\Omega(n)} - \frac{1}{10} \ge \frac{1}{3}$. Finally note that $m = O(n^2 l) = O(n^2 \log n)$. (It is straightforward to see from Lemma 6 that $l = O(\log n)$.) Thus, we have proved Theorem 1.

Although the definition of τ contains an existential quantification over initial states, it will be seen that Theorem 1 is not very sensitive to the initial state: $\tau(\frac{1}{3})$ can be replaced by $\tau_s(\frac{1}{3})$, where s ranges over almost every state in $\Omega_{1:1:1}(\varepsilon)$ or $\Omega_{4:1:1}(\varepsilon)$, as appropriate ("almost every" being interpreted with respect to the stationary distribution).

6 Open problems

In this paper, we have shown that the Burnside process is not rapidly mixing in general. It remains an open question whether there is some *other* polynomial-time method which achieves the same distribution as the Burnside process, either on permutations (as in Lemma 1) or on words (as in Lemma 2). Since (B1) is easy to implement in polynomial-time, a polynomial-time sampling algorithm for the stationary distribution π of Lemma 1 would yield a polynomial-time sampler for the stationary distribution π' of Lemma 2 (i.e., the uniform distribution on

orbits). If there is a polynomial-time sampling algorithm for the distribution π this will imply [9] that there is a *fully polynomial randomised approximation scheme* for the single-variable *cycle index polynomial* for every integer k (see [3]). Such a result would be a striking contrast to the result of the authors (see [5]) which shows that, unless NP = RP, no such approximation algorithm exists for any fixed rational non-integer k.

References

1. J. Besag and P. Green, Spatial statistics and Bayesian computation, *J. Royal Statistical Society* **B 55(1)** 25–37 (1993). (See also the "Discussion on the Gibbs Sampler and Other MCMC Methods" on pages 53–102.)
2. B. Bollobás, G. Grimmett and S. Janson, The random-cluster model on the complete graph, *Probability Theory and Related Fields* **104** (1996), 283–317.
3. N. G. de Bruijn, Pólya's theory of counting, in *Applied Combinatorial Mathematics* (E. F. Beckenbach, ed.), John Wiley and Sons, 1964, 144–184.
4. R. G. Edwards and A. D. Sokal, Generalizations of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm, *Physical Review D* **38** (1988), 2009–2012.
5. Leslie Ann Goldberg, Automating Pólya theory: the computational complexity of the cycle index polynomial, *Information and Computation* **105** (1993), 268–288.
6. Vivek Gore and Mark Jerrum, The Swendsen-Wang process does not always mix rapidly, *Proceedings of the 29th ACM Symposium on Theory of Computation* (STOC), ACM Press, 1997, 674–681. (Theorem numbers cited here are from the proceedings version, not from the Edinburgh University technical report version (ECS-LFCS-96-349, 1996).)
7. F. Jaeger, D. L. Vertigan, and D. J. A. Welsh, On the computational complexity of the Jones and Tutte polynomials, *Mathematical Proceedings of the Cambridge Philosophical Society* **108** (1990), 35–53.
8. Mark Jerrum, Uniform sampling modulo a group of symmetries using Markov chain simulation. In "Expanding Graphs" (Joel Friedman, ed.), *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **10**, American Mathematical Society, 1993, 37–47.
9. Mark Jerrum, Computational Pólya theory. In "Surveys in Combinatorics 1995," *London Mathematical Society Lecture Note Series* **218**, Cambridge University Press, 1995, 103–118.
10. Mark Jerrum and Alistair Sinclair, The Markov chain Monte Carlo method: an approach to approximate counting and integration. In *Approximation Algorithms for NP-hard Problems* (Dorit Hochbaum, ed.), PWS, 1996, 482–520.
11. Eugene M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *Journal of Computer and System Sciences* **25** (1982), 42–65.
12. Colin McDiarmid, On the method of bounded differences, *London Mathematical Society Lecture Note Series* **141**, Cambridge University Press, 1989, 148–188.
13. Paul Martin, *Potts Models and Related Problems in Statistical Mechanics*, World Scientific, Singapore, 1991.
14. P. M. Neumann, A lemma that is not Burnside's, *Mathematical Scientist* **4** (1979), 133–141.
15. Neil O'Connell, Some large deviation results for sparse random graphs, *Probability Theory and Related Fields* **110** (1998), 277–285.

16. Leonard SOICHER, personal communication.
17. N.C. WORMALD, Generating random unlabelled graphs, *SIAM Journal of Computing* **16** (1987) 717–727.

Quicksort Again Revisited*

Charles Knessl[1] and Wojciech Szpankowski[2]

[1] Dept. Mathematics, Statistics & Computer Science, University of Illinois, Chicago, Illinois 60607-7045, USA
[2] Dept. Computer Science, Purdue University, W. Lafayette, IN 47907, USA

Abstract. We consider the standard Quicksort algorithm that sorts n distinct keys with all possible $n!$ orderings of keys being equally likely. Equivalently, we analyze the total path length $\mathcal{L}_n$ in a randomly built *binary search tree*. Obtaining the limiting distribution of $\mathcal{L}_n$ is still an outstanding open problem. In this paper, we establish an integral equation for the probability density of the number of comparisons $\mathcal{L}_n$. Then, we investigate the large deviations of $\mathcal{L}_n$. We shall show that the left tail of the limiting distribution is much "thinner" (i.e., double exponential) than the right tail (which is only exponential). We use formal asymptotic methods of applied mathematics such as the WKB method and matched asymptotics.

1 Introduction

Hoare's *Quicksort* algorithm [9] is the most popular sorting algorithm due to its good performance in practise. The basic algorithm can be briefly described as follows [9, 12, 14]:

> A pivotal key is selected at random from the unsorted list of keys, and used to partition the keys into two sublists to which the same algorithm is called recursively until the sublists have size one or zero.

To justify the algorithm's good performance in practise, a body of theory was built. First of all, every undergraduate learns in a data structures course that the algorithm sorts "on average" n keys in $\Theta(n \log n)$ steps. To be more precise, one assumes that all $n!$ possible orderings of keys are equally likely. It is, however, also known that in the worst case the algorithm needs $O(n^2)$ steps (e.g., think of an input that is given in a decreasing order when the output is printed in an increasing order). Thus, one needs a more detailed probabilistic analysis to understand better the Quicksort behavior. In particular, one wants to know how likely (or rather unlikely) it is for such pathological behavior to occur. Our goal is to answer precisely this question.

A large body of literature is devoted to analyzing the Quicksort algorithm [3, 4, 5, 7, 12, 13, 14, 15, 16, 18]. However, many aspects of this problem are

* The work was supported by NSF Grant DMS-93-00136 and DOE Grant DE-FG02-93ER25168, as well as by NSF Grants NCR-9415491, NCR-9804760.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 346-356, 1998.

still largely unsolved. To review what is known and what is still unsolved, we introduce some notation. Let $\mathcal{L}_n$ denote the number of comparisons needed to sort a random list of length n. It is known that after selecting randomly a key, the two sublists are still "random" (cf. [12]). Clearly, the sorting time depends only on the keys' ranking, so we assume that the input consists of the first n integers $\{1, 2, \ldots, n\}$, and key k is chosen with probability $1/n$. Then, the following recurrence holds

$$\mathcal{L}_n = n - 1 + \mathcal{L}_k + \mathcal{L}_{n-1-k} \ .$$

Now, let $L_n(u) = \mathbf{E}u^{\mathcal{L}_n} = \sum_{k\geq 0} \mathsf{Pr}[\mathcal{L}_n = k]u^k$ be the probability generating function of $\mathcal{L}_n$. The above recurrence implies that

$$L_n(u) = \frac{u^{n-1}}{n} \sum_{i=0}^{n-1} L_i(u) L_{n-1-i}(u) \tag{1}$$

with $L_0(u) = 1$. Observe that the same recurrences are obtained when analyzing the total path length $\mathcal{L}_n$ of a binary search tree built over a random set of n keys (cf. [12, 14]). Finally, let us define a bivariate generating function $L(z, u) = \sum_{n\geq 0} L_n(u) z^n$. Then (1) leads to the following partial-differential functional equation

$$\frac{\partial L(z,u)}{\partial z} = L^2(zu, u) \ , \qquad \frac{\partial L(0,u)}{\partial z} = 1. \tag{2}$$

Observe also that $L(z, 1) = (1 - z)^{-1}$.

The moments of $\mathcal{L}_n$ are relatively easy to compute since they are related to derivatives of $L_n(u)$ at $u = 1$. Hennequin [7] analyzed these carefully and computed exactly the first five cumulants, and obtained an asymptotic formula for all cumulants as $n \to \infty$.

The main open problem is to find the limiting distribution of $\mathcal{L}_n$. Régnier [15] proved that the limiting distribution of $(\mathcal{L}_n - \mathbf{E}[\mathcal{L}_n])/n$ exists, while Rösler [16, 17] characterized this limiting distribution as a fixed point of a contraction satisfying a recurrence equation. A partial-differential functional equation seemingly similar to (2) was studied recently by Jacquet and Szpankowski [10]. They analyzed a *digital* search tree for which the bivariate generating function $L(z, u)$ (in the so–called symmetric case) satisfies

$$\frac{\partial L(z,u)}{\partial z} = L^2(\frac{1}{2}zu, u)$$

with $L(z, 0) = 1$. The above equation was solved asymptotically in [10], and this led to a limiting normal distribution of the path length $\mathcal{L}_n$ in digital search trees. While the above equation and (2) look similar, there are crucial differences. Among them, the most important is the contracting factor $\frac{1}{2}$ in the right-hand side of the above. Needless to say, we know that (2) does *not* lead to a normal distribution since the third moment of $(\mathcal{L}_n - \mathbf{E}[L_n])/n$ does not tend to zero (cf. [14]).

In view of the above discussion, a less ambitious goal was set, namely that of computing the large deviations of $\mathcal{L}_n$, i.e., $\mathsf{Pr}[|\mathcal{L}_n - \mathbf{E}[\mathcal{L}_n]| \geq \varepsilon \mathbf{E}[\mathcal{L}_n]]$ for $\varepsilon > 0$. Hennequin [7] used Chebyshev's inequality to show that the above probability is $O(1/(\varepsilon \log^2 n))$. Recently, Rösler [16] showed that this probability is in fact $O(n^{-k})$ for any fixed k, and soon after McDiarmid and Hayward [13] used the powerful method of *bounded differences*, known also as Azuma's inequality, to obtain an even better estimate, namely that the tail is $O(n^{-2\varepsilon \log\log n})$ (see the comment after Theorem 1 of Section 2).

In this paper, we improve on some of the previous results. First of all, we establish an integral equation for the probability density of $\mathcal{L}_n$, and using this we derive a left tail and a right tail of the large deviations of $\mathcal{L}_n$. We demonstrate that the left tail is much thinner (i.e., double exponential) than the right tail, which is roughly exponential. We establish these results using formal asymptotic methods of applied mathematics such as the WKB method and matched asymptotics (cf. [2, 6]).

This paper is a conference version of the full paper [11] which contains all proofs while here we rather sketch some of the derivations. The full version can be found on `http://www.cs.purdue.edu/homes/spa/current.html`.

2 Formulation and Summary of Results

As before, we let $\mathcal{L}_n$ be the number of key comparisons made when Quicksort sorts n keys. The probability generating function of $\mathcal{L}_n$ becomes

$$L_n(u) = \sum_{k=0}^{\infty} \mathsf{Pr}[\mathcal{L}_n = k] u^k = \mathbf{E}[u^{\mathcal{L}_n}]. \tag{3}$$

The upper limit in this sum may be truncated at $k = \binom{n}{2}$, since this is clearly an upper bound on the number of comparisons needed to sort n keys.

The generating function $L_n(u)$ satisfies (1) which we repeat below (cf. also [5, 14, 15, 16])

$$L_{n+1}(u) = \frac{u^n}{n+1} \sum_{i=0}^{n} L_i(u) L_{n-i}(u), \quad L_0(u) = 1. \tag{4}$$

Note that $L_n(1) = 1$ for all $n \geq 0$, and that the probability $\mathsf{Pr}[\mathcal{L}_n = k]$ may be recovered from the Cauchy integral

$$\mathsf{Pr}[\mathcal{L}_n = k] = \frac{1}{2\pi i} \int_C u^{-k-1} L_n(u) du. \tag{5}$$

Here C is any closed loop about the origin.

In Section 3, we analyze (4) asymptotically for $n \to \infty$ and for various ranges of u. The most important scale is where $n \to \infty$ with $u - 1 = O(n^{-1})$, which corresponds to $k = \mathbf{E}[\mathcal{L}_n] + O(n) = 2n \log n + O(n)$. Most of the probability mass is concentrated in this range of k. As mentioned before, the existence of a

limiting distribution of $(\mathcal{L}_n - \mathbf{E}[\mathcal{L}_n])/n$ as $n \to \infty$ was established in [15, 16], though there seems to be little known about this distribution (cf. [3, 5, 7, 13, 18]). Numerical and simulation results in [3, 5] show that the distribution is highly asymmetric and that the right tail seems much thicker than the left tail. It is also of interest to estimate these tails (cf. [13, 16]), as they give the probabilities that the number of key comparisons will deviate significantly from $\mathbf{E}[\mathcal{L}_n]$, which is well known to be asymptotically equal to $2n \log n$ as $n \to \infty$ (cf. [7, 14]).

For $u - 1 = w/n = O(n^{-1})$ and $n \to \infty$, we shall show that

$$L_n(u) = \exp(A_n w/n) \left(G_0(w) + \frac{\log n}{n} G_1(w) + \frac{1}{n} G_2(w) + o(n^{-1}) \right) \tag{6}$$

where $A_n = \mathbf{E}[\mathcal{L}_n]$. The leading term $G_0(w)$ satisfies a non-linear integral equation. Indeed, we find that (cf. (39))

$$e^{-w} G_0(w) = \int_0^1 e^{2\phi(x)w} G_0(wx) G_0(w - wx) dx \tag{7}$$

$$G_0(0) = 1; \quad G_0'(0) = 0 \tag{8}$$

where

$$\phi(x) = x \log x + (1 - x) \log(1 - x) \tag{9}$$

is the entropy of the Bernoulli(x) distribution. Furthermore, the correction terms $G_1(\cdot)$ and $G_2(\cdot)$ satisfy linear integral equations (cf. (40)–(41)).

By using (6) in (5) and asymptotically approximating the Cauchy integral we obtain

$$\Pr[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) = ny] \sim \frac{1}{n} P(y) \tag{10}$$

where

$$P(y) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^{-yw} G_0(w) dw, \tag{11}$$

$$G_0(w) = \int_{-\infty}^{\infty} e^{yw} P(y) dy \tag{12}$$

and c is a constant. Hence, $G_0(w)$ is the moment generating function of the density $P(y)$.

Now, we can summarize our main findings:

Theorem 1. *Consider the Quicksort algorithm that sorts n randomly selected keys.*

(i) *The limiting density $P(y)$ satisfies*

$$P(y+1) = \int_0^1 \int_{-\infty}^{\infty} P\left(xt + \frac{y - 2\phi(x)}{2(1-x)} \right) P\left(-(1-x)t + \frac{y - 2\phi(x)}{2x} \right) dt dx \tag{13}$$

and

$$\int_{-\infty}^{\infty} P(y)dy = 1, \quad \int_{-\infty}^{\infty} yP(y)dy = 0. \tag{14}$$

(ii) *The left tail of the distribution satisfies*

$$\mathsf{Pr}[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) \le nz] \sim \frac{2}{\pi}\frac{1}{\sqrt{2\log 2 - 1}} \exp\left(-\alpha \exp\left(\frac{\beta - z}{2 - \log^{-1} 2}\right)\right) \tag{15}$$

for $n \to \infty$, $z \to -\infty$ where $\alpha = \frac{2\log 2 - 1}{e \log 2} = 0.205021\ldots$ and β is a constant.

(iii) *The right tail becomes*

$$\begin{aligned}\mathsf{Pr}[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) \ge ny] \sim{}& \frac{C_*}{2\sqrt{\pi}}\frac{1}{w_*\sqrt{w_* - 1}} e^{-w_*(w_* - \frac{1}{2} + 2\gamma + \log 2)} \\ &\times \exp[-yw_* + \int_1^{w_*} \frac{2e^u}{u} du]\end{aligned} \tag{16}$$

for $n \to \infty$, $y \to +\infty$. Here C_* is a constant, γ is Euler's constant, while $w_* = w_*(y)$ is the solution to

$$y = \frac{2}{w_*}\exp(w_*) \tag{17}$$

that asymptotically becomes

$$w_* = \log\left(\frac{y}{2}\right) + \log\log\left(\frac{y}{2}\right) + \frac{\log\log(y/2)}{\log(y/2)}(1 + o(1)) \tag{18}$$

for $y \to \infty$ (cf. (51)).

Finally, we relate our results for the tails to those of McDiarmid and Hayward [13]. These authors showed that

$$\mathsf{Pr}[|\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n)| > \varepsilon \mathbf{E}(\mathcal{L}_n)] = \exp[-2\varepsilon \log n(\log\log n - \log(1/\varepsilon) + O(\log\log\log n))], \tag{19}$$

which holds for $n \to \infty$ and ε in the range

$$\frac{1}{\log n} < \varepsilon \le 1. \tag{20}$$

As pointed out in [13], this estimate is not very precise if, say, $\varepsilon = O(\log\log n/\log n)$. From Theorem 1 we conclude that (since the right tail is much thicker than the left tail)

$$\mathsf{Pr}[|\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n)| \ge ny] \sim Ck(y)e^{\phi(y)}, \quad y \to \infty \tag{21}$$

where C is a constant and

$$\begin{aligned}\phi(y) &= -yw_* + \int_1^{w_*} \frac{2e^u}{u} du, \qquad y = \frac{2e^{w_*}}{w_*}, \\ k(y) &= \frac{e^{-w_*^2} e^{w_*(\frac{1}{2} - 2\gamma - \log 2)}}{w_*\sqrt{w_* - 1}}.\end{aligned} \tag{22}$$

We have not been able to determine the upper limit on y for the validity of (21). However, it is easy to see that (21) reduces to (19) if we set $y = \varepsilon \mathbf{E}(\mathcal{L}_n)/n = 2\varepsilon \log n + +\varepsilon(2\gamma - 4) + O(\varepsilon(\log n)/n)$ and use (18) to approximate w_* as $y \to \infty$. This yields

$$\begin{aligned} -yw_* + \int_1^{w_*} \frac{2e^u}{u} du &= y\left[-\log\left(\frac{y}{2}\right) - \log\log\left(\frac{y}{2}\right) + 1 + o(1)\right] \\ &= -2\varepsilon \log n\,[\log\log n - \log(1/\varepsilon) + \log\log(\varepsilon \log n) - 1] + o(\log n) \end{aligned} \tag{23}$$

which agrees precisely with the estimate (19), and also explicitly identifies the $O(\log\log\log n)$ error term. This suggests that (21) applies for y as large as $2\log n$, though it cannot hold for y as large as $n/2$ in view of the fact that $\mathsf{Pr}[\mathcal{L}_n = k] = 0$ for $k > \binom{n}{2}$. An important open problem is obtaining an accurate numerical approximation to the constant C. This would likely involve the numerical solution of the integral equation for $G_0(w)$.

3 Analysis

We study (4) asymptotically, for various ranges of n and u, and then we evaluate the tails of the distribution.

First we consider the limit $u \to 1$ with n fixed. Using the Taylor expansion

$$\begin{aligned} L_n(u) &= 1 + A_n(u-1) + B_n(u-1)^2 + O((u-1)^3) \\ &= e^{A_n(u-1)}[1 + (B_n - \frac{1}{2}A_n^2)(u-1)^2 + O((u-1)^3)] \end{aligned} \tag{24}$$

we find from (4) that $A_n = L_n'(1)$ and $B_n = L_n''(1)/2$ satisfy the linear recurrence equations

$$A_{n+1} = n + \frac{1}{n+1}\sum_{i=0}^{n}[A_i + A_{n-i}] = n + \frac{2}{n+1}\sum_{i=0}^{n} A_i; \quad A_0 = 0, \tag{25}$$

$$B_{n+1} = \binom{n}{2} + \frac{2n}{n+1}\sum_{i=0}^{n} A_i \frac{1}{n+1} + \sum_{i=0}^{n}[2B_i + A_i A_{n-i}]; \quad B_0 = 0. \tag{26}$$

These are easily solved using either generating functions, or by multiplying (25) and (26) by $n+1$ and then differencing with respect to n. The final result is (cf. [12, 14])

$$A_n = 2(n+1)H_n - 4n \tag{27}$$

$$B_n = 2(n+1)^2 H_n^2 - (8n+2)(n+1)H_n + \frac{n}{2}(23n+17) - 2(n+1)^2 H_n^{(2)}. \tag{28}$$

Here $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the harmonic number, and $H_n^{(2)} = \sum_{k=1}^{n} k^{-2}$ is the harmonic number of second order. In terms of A_n and B_n, the mean and

variance of $\mathcal{L}_n$ are given by

$$\mathbf{E}[\mathcal{L}_n] = A_n = 2(n+1)H_n - 4n \tag{29}$$

$$\begin{aligned}\mathbf{Var}\,[\mathcal{L}_n] &= L_n''(1) + L_n'(1) - [L_n'(1)]^2 = 2B_n + A_n - A_n^2 \\ &= 7n^2 - 2(n+1)H_n + 13n - 4(n+1)^2 H_n^{(2)}.\end{aligned} \tag{30}$$

Asymptotically, for $n \to \infty$, we also obtain

$$A_n = 2n\log n + (2\gamma - 4)n + 2\log n + 2\gamma + 1 + \frac{5}{6}n^{-1} + O(n^{-2}) \tag{31}$$

$$B_n - \frac{1}{2}A_n^2 \equiv C_n = \left(\frac{7}{2} - \frac{\pi^2}{3}\right)n^2 - 2n\log n + n\left(\frac{21}{2} - 2\gamma - \frac{2}{3}\pi^2\right) + o(n). \tag{32}$$

These expressions will be used in order to asymptotically match the expansion for $u \to 1$ and n fixed, to those that will apply for other ranges of n and u.

Next we consider the limit $u \to 1$, $n \to \infty$ with $w \equiv n(u-1)$ held fixed. We define $G(\cdot)$ by

$$L_n(u) = \exp\left(A_n w/n\right) G(w;n) = e^{A_n(u-1)} G(n(u-1);n). \tag{33}$$

From (33) and the identity $L_n'(1) = A_n = \mathbf{E}[\mathcal{L}_n]$ we find that for all n

$$G(0;n) = 1 \quad \text{and} \quad G'(0;n) = 0. \tag{34}$$

We assume that for $n \to \infty$, $G(w;n)$ has an asymptotic expansion of the form

$$G(w;n) = G_0(w) + a_1(n)G_1(w) + a_2(n)G_2(w) + \cdots \tag{35}$$

where $a_j(n)$ is an asymptotic sequence as $n \to \infty$, i.e., $a_{j+1}(n)/a_j(n) \to 0$ as $n \to \infty$. We will eventually obtain

$$a_1(n) = \frac{\log n}{n}; \qquad a_2(n) = \frac{1}{n}, \tag{36}$$

so we use this form from the beginning. Note that $G_0(w)$ is the moment generating function of the limiting density of $(\mathcal{L}_n - \mathbf{E}[\mathcal{L}_n])/n$, which is discussed in [16]. The conditions (34) imply that

$$G_0(0) = 1; \quad G_1(0) = G_2(0) = \cdots = 0 \tag{37}$$

$$G_0'(0) = G_1'(0) = G_2'(0) = \cdots = 0. \tag{38}$$

In fact, in the full version of this paper [11] we prove that $G_i(w)$ satisfy the following integral equations:

$$e^{-w}G_0(w) = \int_0^1 e^{2\phi(x)w} G_0(wx)G_0(w - wx)dx, \tag{39}$$

$$e^{-w}G_1(w) = 2\int_0^1 \frac{1}{x} e^{2\phi(x)w} G_0(w - wx)G_1(wx)dx, \tag{40}$$

$$e^{-w}[G_0(w) + \frac{1}{2}w^2 G_0(w) + wG_0'(w) + G_2(w)] = G_0(w) \tag{41}$$
$$+ w\int_0^1 [2\psi(x)+3]e^{2\phi(x)w}G_0(wx)G_0(w-wx)dx$$
$$+ 2\int_0^1 \frac{\log x}{x}e^{2\phi(x)w}G_0(w-wx)G_1(wx)dx$$
$$+ 2\int_0^1 \frac{1}{x}e^{2\phi(x)w}G_0(w-wx)G_2(wx)dx.$$

Equations (39)-(41), along with (37) and (38) are integral equations for the first three terms in the series (35). The leading order equation (39) was previously obtained in [5], using probabilistic arguments.

Now, we shall examine the leading term $G_0(w)$, and we first consider the limit $w \to -\infty$. As $w \to -\infty$, the "kernel" $\exp[2w\phi(x)]$ in (39) is sharply concentrated near $x = 1/2$, and behaves as a multiple of $|w|^{-1/2}\delta(x-1/2)$. Thus we treat (39) as a Laplace type integral (cf. [8]). The case $w \to \infty$ can be treated in a similar manner. We prove the following asymptotic expansions (see [11] for details)

$$G_0(w) \sim \frac{2\sqrt{2}}{\sqrt{\pi\log 2}}\sqrt{-w}e^{\beta w}\exp\left[\left(\frac{1}{\log 2}-2\right)w\log(-w)\right], \quad w \to -\infty, \tag{42}$$

$$G_0(w) \sim C_*\exp\left(\int_1^w \frac{2e^u}{u}du\right)e^{-w^2}e^{(1-2\gamma-2\log 2)w}\frac{1}{w}, \qquad w \to \infty. \tag{43}$$

Finally, we analyze the tails of the limiting distribution. Using the approximation (33) for $u - 1 = O(n^{-1})$, we obtain

$$\mathsf{Pr}[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) = ny] = \frac{1}{2\pi i}\int_C z^{-[ny+A_n+1]}L_n(z)dz \tag{44}$$
$$\sim \frac{1}{n}\frac{1}{2\pi i}\int_{Br} e^{-yw}G_0(w)dw = \frac{1}{n}P(y)$$

where $Br = (c - i\infty, c + i\infty)$ for some constant c, is any vertical contour in the w-plane. Here we have set $z = 1 + w/n$ in the integral. It follows that

$$G_0(w) = \int_{-\infty}^{\infty} e^{wy}P(y)dy \tag{45}$$

so that $G_0(w)$ is the moment generating function of the density $P(y)$. In view of (37) and (38) we have $\int_{-\infty}^{\infty} P(y)dy = 1$ and $\int_{-\infty}^{\infty} yP(y)dy = 0$.

Observe that, using (33), (35), and (36), we can refine the approximation (44) to

$$\mathsf{Pr}[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) = ny] = \frac{1}{n}\left(P(y) + \frac{\log n}{n}[P_1(y) + P''(y)]\right.$$
$$\left. + \frac{1}{n}[P_2(y) + P'(y) + \frac{1}{2}yP''(y) + (\gamma-2)P''(y)] + o(n^{-1})\right)$$

where

$$P_k(y) = \frac{1}{n}\frac{1}{2\pi i}\int_{Br} e^{-yw} G_k(w)dw$$

for $k = 1, 2$.

An integral equation for $P(y)$ can easily be derived from (39). We multiply (39) by $e^{-wy}/(2\pi i)$ and integrate over a contour Br in the w-plane:

$$\begin{aligned} P(y+1) &= \frac{1}{2\pi i}\int_{Br} e^{-w(y+1)} G_0(w)dw \qquad (46)\\ &= \int_0^1 \int_{-\infty}^{\infty} P\left(xt + \frac{y/2-\phi(x)}{1-x}\right) P\left(-(1-x)t + \frac{y/2-\phi(x)}{x}\right) dtdx, \end{aligned}$$

where we used the well known identity $\frac{1}{2\pi i}\int_{Br} e^{-yw}dw = \delta(y)$, where $\delta(y)$ is Dirac's delta function (cf. [1]). The last expression is precisely (13). The solution to this integral equation is not unique: if $P(y)$ is a solution, so is $P(y+c)$ for any c.

We now study $P(y) = (2\pi i)^{-1}\int_{Br} e^{-yw} G_0(w)dw$ as $y \to \pm\infty$. We argue that the asymptotic expansion of the integral will be determined by a saddle point $w = s(y)$, which satisfies $s(y) \to \pm\infty$ as $y \to \pm\infty$. Thus for $y \to -\infty$, we can use the approximation (42) for $G_0(w)$, which yields

$$P(y) \sim \frac{1}{2\pi i}\int_{Br} \frac{2\sqrt{2}}{\sqrt{\pi \log 2}}\sqrt{-w}\, e^{(\beta-y)w} \exp\left[\left(\frac{1}{\log 2} - 2\right) w\log(-w)\right] dw. \qquad (47)$$

This integrand has a saddle point where

$$\frac{d}{dw}\left[-(y-\beta)w + \left(\frac{1}{\log 2} - 2\right) w\log(-w)\right] = 0,$$

so that

$$w = -\frac{1}{e}\exp\left[-\frac{y-\beta}{2 - 1/\log 2}\right] \equiv \tilde{w}(y)$$

which satisfies $\tilde{w}(y) \to -\infty$ as $y \to -\infty$. Then the standard saddle point approximation to (47) yields

$$\begin{aligned} P(y) &\sim \frac{2\sqrt{2}}{\sqrt{\pi \log 2}}\sqrt{-\tilde{w}}\, e^{-(y-\beta)\tilde{w}} \exp\left[\left(\frac{1}{\log 2} - 2\right)\tilde{w}\log(-\tilde{w})\right]\\ &\quad \times \frac{1}{2\pi i}\int_{Br} \exp\left[\frac{1}{2\tilde{w}}\left(\frac{1}{\log 2} - 2\right)(w-\tilde{w})^2\right] dw \qquad (48)\\ &= \frac{2}{\pi e}\frac{1}{\sqrt{2\log 2 - 1}} \exp\left[\frac{\beta - y}{2 - 1/\log 2} - \frac{2 - 1/\log 2}{e}\exp\left(\frac{\beta - y}{2 - 1/\log 2}\right)\right] \end{aligned}$$

for $y \to -\infty$. Thus, the left tail is very small and the behavior of $P(y)$ as $y \to -\infty$ is similar to that of an extreme value distribution (i.e., double-exponential distribution).

Now take $y \to +\infty$ and use (43) to get

$$P(y) \sim \frac{1}{2\pi i} \int_{Br} e^{-yw} C_* \exp\left(\int_1^w \frac{2e^u}{u} du\right) \frac{e^{-w^2}}{w} e^{(1-2\gamma-2\log 2)w} dw. \tag{49}$$

The saddle point now satisfies

$$\frac{d}{dw}\left[-yw + \int_1^w \frac{2e^u}{u} du\right] = 0$$

or $y = 2e^w/w$. Let $w_* = w_*(y)$ be the solution to (17) that satisfies $w_* \to \infty$ as $y \to \infty$. Then expanding the integrand in (49) about $w = w_*(y)$ and using again the saddle point approximation yields

$$P(y) \sim \frac{C_*}{2\sqrt{2\pi}} \frac{\sqrt{y}}{\sqrt{1-1/w_*}} \exp\left[-yw_* + \int_1^{w_*} \frac{2e^u}{u} du\right] \exp[-w_*^2 - (2\gamma + 2\log 2)w_*] \tag{50}$$

as $y \to \infty$, from which (16) easily follows. Thus for $y \to \infty$ we have $P(y) = \exp[O(-y \log y)]$ and hence the right tail is *thinner* than that of an extreme value distribution. From (17) it is easy to show that

$$w_* = \log\left(\frac{y}{2}\right) + \log\log\left(\frac{y}{2}\right) + \frac{\log\log(y/2)}{\log(y/2)}(1 + o(1)), \qquad y \to \infty. \tag{51}$$

For fixed z and y we have, as $n \to \infty$,

$$\mathsf{Pr}[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) \le nz] \sim \int_{-\infty}^{z} P(y) dy \tag{52}$$

$$\mathsf{Pr}[\mathcal{L}_n - \mathbf{E}(\mathcal{L}_n) \ge ny] \sim \int_{y}^{\infty} P(u) du. \tag{53}$$

If $z \to -\infty$ or $y \to +\infty$, then these integrals may be evaluated asymptotically using (48) and (50), and we obtain the results (15) and (16), respectively.

References

1. Abramowitz, M. and Stegun, I., eds. *Handbook of Mathematical Functions*. Wiley, New York, 1962.
2. C. Bender and S. Orszag, *Advanced Mathematical Methods for Scientists and Engineers*, McGrew-Hill, New York 1978.
3. M. Cramer, A Note Concerning the Limit Distribution of the Quicksort Algorithm, *Theoretical Informatics and Applications*, **30**, 195–207, 1996.
4. L. Devroye, A Note on the Height of binary Search Trees, *J. ACM*, **33**, 489–498, 1986.
5. W. Eddy and M. Schervish, How Many Comparisons Does Quicksort Use, *J. Algorithms*, **19**, 402–431, 1995.
6. N. Froman and P. Froman, *JWKB Approximation*, North-Holland, Amsterdam 1965.

7. P. Hennequin, Combinatorial Analysis of Quicksort Algorithm, *Theoretical Informatics and Applications*, **23**, 317–333, 1989.
8. P. Henrici, *Applied and Computational Complex Analysis*, John Wiley&Sons, New York 1977.
9. C.A.R. Hoare, Quicksort, *Comput. J.*, **5**, 10–15, 1962.
10. P. Jacquet and W. Szpankowski, Asymptotic Behavior of the Lempel-Ziv Parsing Scheme and Digital Search Trees, *Theoretical Computer Science*, **144**, 161-197, 1995.
11. C. Knessl and W. Szpankowski, Quicksort Algorithm Again Revisited, Purdue University CSD-TR-97-015, 1997.
12. D. Knuth, *The Art of Computer Programming. Sorting and Searching,* Addison-Wesley (1973).
13. C.J. McDiarmid and R. Hayward, Large Deviations for Quicksort, *J. Algorithms*, **21**, 476–507, 1996.
14. H. Mahmoud, *Evolution of Random Search Trees*, John Wiley & Sons, New York (1992).
15. M. Régnier, A Limiting Distribution for Quicksort, *Theoretical Informatics and Applications*, **23**, 335–343, 1989.
16. U. Rösler, A Limit Theorem for Quicksort, *Theoretical Informatics and Applications*, **25**, 85–100, 1991.
17. U. Rösler, A Fixed Point Theorem for Distributions, *Stochastic Processes and Their Applications*, **42**, 195–214, 1992.
18. K.H. Tan and P. Hadjicostas, Some Properties of a Limiting Distribution in Quicksort, *Statistics & Probability Letters*, **25**, 87–94, 1995.

Sampling Methods Applied to Dense Instances of Non-Boolean Optimization Problems

Gunnar Andersson and Lars Engebretsen

Royal Institute of Technology
Department of Numerical Analysis and Computing Science
SE-100 44 Stockholm, SWEDEN
Fax: +46 8 790 09 30
E-mail: {gunnar,enge}@nada.kth.se

Abstract. We study dense instances of optimization problems with variables taking values in $\boldsymbol{Z}_p$. Specifically, we study systems of functions from $\boldsymbol{Z}_p^k$ to $\boldsymbol{Z}_p$ where the objective is to make as many functions as possible attain the value zero. We generalize earlier sampling methods and thereby construct a randomized polynomial time approximation scheme for instances with $\Theta(n^k)$ functions where n is the number of variables occurring in the functions.

1 Introduction

Arora, Karger and Karpinski [1] have constructed a randomized polynomial time approximation scheme for dense instances of a number of **Max-SNP** problems, including Max Cut. They formulate the problems as integer programs with certain properties, and then construct an algorithm finding, in probabilistic polynomial time, a solution accurate enough to give a relative error of ε, for any $\varepsilon > 0$. Fernandez de la Vega [3] has also constructed a randomized polynomial time approximation scheme for dense instances of Max Cut, independently of Arora, Karger and Karpinski. It is natural to look for generalizations of these ideas to other problems; for instance one can turn to problems with variables taking values in $\boldsymbol{Z}_p$ rather than $\boldsymbol{Z}_2$. The method of Arora, Karger and Karpinski [1] does not seem to apply in this case since the integer programs used to express such generalizations do not have the properties required by the method.

The algorithm of Fernandez de la Vega [3] selects a random subset W of the vertices in the graph $G = (V, E)$. This subset has constant size. Then, $V \setminus W$ is partitioned randomly into smaller sets. These sets are used to construct a cut in G by exhaustive search. Finally, it turns out that the randomly selected subset W has, with high probability, the property that the cut found by the exhaustive search is close to the optimum cut in dense graphs. Goldreich, Goldwasser and Ron [5] generalize these ideas to several other problems, and express the key idea somewhat differently. In their randomized polynomial time approximation scheme for Max Cut, they partition the vertices of the graph $G = (V, E)$ into a constant number of disjoint sets V^i. For each i they find a cut in V^i by

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 357-368, 1998.

selecting a small subset U^i of the vertices in $V \setminus V^i$. Then they try all possible partitions π of U^i into two parts. Each partition π induces a cut in V^i. Finally, when π is exactly the partition from the maximum cut restricted to the subset in question, the weight of the induced cut should, with high probability, be close to the weight of the maximum cut. In this paper, we continue this line of research by generalizing the above ideas to arbitrary non-boolean optimization problems.

Frieze and Kannan [4] have constructed a polynomial time approximation scheme for all dense **Max-SNP** problems. Their algorithm is a polynomial time approximation scheme for every problem that can be described as an instance of Max k-Function Sat with $\Theta(n^k)$ functions. Dense instances of Max k-Function Sat mod p do not seem to be describable in this manner, and on top of that, the algorithm proposed in this paper has a simpler structure and shorter running time than their algorithm.

2 Preliminaries

Definition 1. *We denote by Max Ek-Lin mod p the problem in which the input consists of a system of linear equations mod p in n variables. Each equation contains exactly k variables. The objective is to find the assignment maximizing the number of satisfied equations.*

Definition 2. *We denote by Max k-Function Sat the problem in which the input consists of a number of boolean functions in n boolean variables. Each function depends on k variables. The objective is to find the assignment maximizing the number of satisfied functions.*

Definition 3. *We denote by Max k-Function Sat mod p the problem in which the input consists of a number of functions $\boldsymbol{Z}_p^k \mapsto \boldsymbol{Z}_p$ in n variables. A function is satisfied if it evaluates to zero. The objective is to find the assignment maximizing the number of satisfied functions.*

Definition 4. *The class **Max-SNP** is the class of optimization problems which can be written on the form*

$$\max_S \big|\{x : \Phi(I, S, x)\}\big|, \tag{1}$$

where Φ is a quantifier-free formula, I an instance and S a solution. (This class is called $\boldsymbol{Max\text{-}SNP}_0$ in [9].)

For instance, the Max Cut problem is in **Max-SNP** since it can be described as

$$\max_{S \subseteq V} |\{(x, y) : E(x, y) \wedge S(x) \wedge \neg S(y)\}|, \tag{2}$$

where $E(x, y)$ is true if there is an edge (x, y) in the graph.

Definition 5. *A polynomial time approximation scheme for a maximization problem P with objective function $m(\cdot)$ is a family $\{A_\varepsilon\}$, $\varepsilon > 0$, of algorithms with polynomial running time (for fixed ε) such that $m(A_\varepsilon(I)) \geq (1 - \varepsilon)\,\mathrm{opt}(I)$ for all instances I of P, where $\mathrm{opt}(I)$ is the optimal value of the instance.*

In this paper we consider *randomized* polynomial time approximation schemes where $m(A_{\varepsilon,\delta}(I)) \geq (1-\varepsilon)\,\mathrm{opt}(I)$ holds with probability at least $1-\delta$.

3 Systems with two variables per equation

Max E2-Lin mod p is the most natural subfamily of Max k-Function Sat mod p, and for clarity we will first describe the polynomial time approximation scheme and prove the results in this setting.

We consider an unweighted system of linear equations modulo some prime p. There are n different variables in the system. The equations are of the form

$$ax_i + bx_{i'} = c \tag{3}$$

where $i \neq i'$, $a, b \in \boldsymbol{Z}_p^*$, and $c \in \boldsymbol{Z}_p$. We assume that there are no equivalent equations in the system. I.e., if the two equations

$$ax_i + bx_{i'} = c \tag{4}$$

$$a'x_i + b'x_{i'} = c' \tag{5}$$

both are in the system, we assume that there is no $d \in \boldsymbol{Z}_p$ such that $a = da'$, $b = db'$ and $c = dc'$. We think of variable assignments as functions from the set of variables to $\boldsymbol{Z}_p$.

Definition 6. *Denote by $S(X, \tau, x \leftarrow r)$ the number of satisfied equations with one variable from the set X and x as the other variable, given that the variables in X are assigned values according to the function τ and x is assigned the value r.*

The algorithm we use is based on the Max Cut algorithm by Goldreich, Goldwasser and Ron [5], and we use their terminology and notation. The parameters ℓ and t are independent of n. They will be determined during the analysis of the algorithm.

Algorithm 1 *A randomized polynomial time approximation scheme for Max E2-Lin mod p with running time $O(n^2)$.*

1. *Partition the variable set V into ℓ parts $V^1, \ldots, V^\ell$, each of size n/ℓ.*
2. *Choose ℓ sets $U^1, \ldots, U^\ell$ such that U^i is a set of cardinality t chosen uniformly at random from $V \setminus V^i$. Let $U = \bigcup_{i=1}^{\ell} U^i$.*
3. *For each of the (at most) $p^{\ell t}$ assignments $\pi \colon U \mapsto \boldsymbol{Z}_p$, form an assignment $\Pi_\pi \colon V \mapsto \boldsymbol{Z}_p$ in the following way:*
   (a) *Let $\pi' = \pi$.*
   (b) *For $i \in \{1, \ldots, \ell\}$,*
   (c) *For each $v \in V^i$,*
   (d) *Let $j^*(v)$ be the $j \in \boldsymbol{Z}_p$ which maximizes $S(U^i, \pi', v \leftarrow j)$.*
   (e) *Define $\Pi_\pi(v) = j^*(v)$.*
   (f) *Modify π' such that $\pi'|_{V^i} = \Pi_\pi^i$.*
4. *Let Π be the variable assignment Π_π which maximizes the number of satisfied equations.*

5. Return Π.

Our overall goal is to show that it is possible to choose the constants ℓ and t in such a way that Algorithm 1 produces, with probability at least $1 - \delta$, an assignment with weight at least $1 - \varepsilon/c$ times the weight of the optimal assignment for instances with cn^2 equations. In the analysis we will use the constants ε_1, ε_2 and ε_3. They are all linear in ε, and will be determined later.

The intuition behind the algorithm is as follows: Since the graph is dense, the sets U^i should in some sense represent the structure of $V \setminus V^i$. If we pick some variable v from V^i and some assignment to the variables in $V \setminus V^i$ we will, for each assignment $v \leftarrow j$, satisfy some fraction ϕ_j of the equations containing v and one variable from $V \setminus V^i$. We then expect U^i to have the property that the fraction of the satisfied equations containing v and one variable from U^i should be close to ϕ_j. It turns out that the decrease in the number of satisfied equations due to the sampling is $O(n^2)$ which implies that the algorithm is a polynomial approximation scheme only for dense instances of the problem.

Let us now formalize the intuition. From now on, we fix an assignment H to the variables in V and a partition of V into the ℓ parts $V^1, \ldots, V^\ell$. All definitions and results proven below will be relative to these fixed properties, unless stated otherwise.

Definition 7. *We say that the set U^i is* good *if for all except a fraction of at most ε_1 of the variables $v \in V^i$*

$$\left| \frac{S(U^i, H, v \leftarrow j)}{t} - \frac{S(V \setminus V^i, H, v \leftarrow j)}{n - |V^i|} \right| \leq \varepsilon_2 \quad \textit{for all } j \in \boldsymbol{Z}_p. \tag{6}$$

Remark 1. What we call a good set is essentially what Fernandez de la Vega [3] calls a *representative* set.

Lemma 1. *For any $\delta > 0$, it is possible to choose the constant t in such a way that the probability of a set U^i being good is at least $1 - \delta/\ell$ for a fixed i.*

Proof. Fix a variable $v \in V^i$ and some $j \in \boldsymbol{Z}_p$. Note that the assignment H is fixed; the underlying probability space is the possible choices of U^i. We now introduce, for each $w \in V \setminus V^i$, a Bernoulli random variable $\xi_{i,j,v,w}$ with the property that

$$\xi_{i,j,v,w} = \begin{cases} 1 & \text{if } w \in U^i, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

We can use these random variables to express the number of satisfied equations containing v and one variable from U^i.

$$S(U^i, H, v \leftarrow j) = \sum_{w \in V \setminus V^i} S(\{w\}, H, v \leftarrow j) \xi_{i,j,v,w}. \tag{8}$$

Since U^i is chosen uniformly at random from $V \setminus V^i$,

$$\Pr[\xi_{i,j,v,w} = 1] = \frac{t}{n - |V^i|}\,. \tag{9}$$

Now we construct the random variable

$$X_{i,j,v} = \frac{S(U^i, H, v \leftarrow j)}{t}\,. \tag{10}$$

From Eqs. 8 and 9 it follows that

$$E[X_{i,j,v}] = \sum_{w \in V \setminus V^i} \frac{S(\{w\}, H, v \leftarrow j)}{n - |V^i|} = \frac{S(V \setminus V^i, H, v \leftarrow j)}{n - |V^i|}\,, \tag{11}$$

which means that we are in good shape if we can bound the probability that $X_{i,j,v}$ deviates more than ε_2 from its mean. At a first glance, this seems hard to do. For $X_{i,j,v}$ is a linear combination of dependent random variables, and the coefficients in the linear combination depend on the assignment H and the instance. Since there are, for each $w \in V \setminus V^i$, at most $p(p-1)$ equations containing v and w, $S(\{w\}, H, v \leftarrow j)$ can be anywhere between 0 and $p-1$, which complicates things even worse. Fortunately, we can use martingale tail bounds to reach our goal in spite of our limited knowledge of the probability distribution of $X_{i,j,v}$. Since the sets U^i have cardinality t, exactly t different $\xi_{i,j,v,w}$ are non-zero, which means that the sum in Eq. 8 can be written as

$$S(U^i, H, v \leftarrow j) = \sum_{k=1}^{t} Z_k. \tag{12}$$

Each random variable Z_k corresponds to $S(\{w_k\}, H, v \leftarrow j)$ for some $w_k \in V \setminus V^i$, which implies that $0 \le Z_k \le p-1$. Thus, the sequence

$$X^m_{i,j,v} = \mathrm{E}\left[\frac{\sum_{k=1}^{t} Z_k}{t}\middle| Z_1, Z_2, \ldots, Z_m\right] \quad \text{for } m = 0, 1, \ldots, t \tag{13}$$

is a martingale [8] with the properties that

$$\left|X_{i,j,v} - \mathrm{E}[X_{i,j,v}]\right| = \left|X^t_{i,j,v} - X^0_{i,j,v}\right|, \tag{14}$$

$$\left|X^m_{i,j,v} - X^{m-1}_{i,j,v}\right| \le (p-1)/t \quad \text{for all } m \in \{1, \ldots, t\}, \tag{15}$$

which enables us to apply Azuma's inequality [2, 7, 8]:

$$\Pr\left[\left|X_{i,j,v} - \mathrm{E}[X_{i,j,v}]\right| > \varepsilon_2\right] < 2e^{-\varepsilon_2^2 t/2(p-1)^2}. \tag{16}$$

The above inequality is valid for fixed i, j and v. A set U^i is good if the above inequality holds for all j and for all but ε_1 of the vertices in V^i. Thus we keep i and j fixed and construct a new family of Bernoulli random variables

$$\eta_{i,j,v} = \begin{cases} 1 & \text{if } \left|X_{i,j,v} - \mathrm{E}[X_{i,j,v}]\right| > \varepsilon_2, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

and set

$$Y_{i,j} = \frac{1}{|V^i|} \sum_{v \in V^i} \eta_{i,j,v}. \tag{18}$$

By Eq 16,

$$\Pr[\eta_{i,j,v} = 1] < 2e^{-\varepsilon_2^2 t/2(p-1)^2}, \tag{19}$$

and thus

$$\mathrm{E}[Y_{i,j}] < 2e^{-\varepsilon_2^2 t/2(p-1)^2}. \tag{20}$$

We can now use Markov's inequality to bound

$$\Pr[Y_{i,j} > \varepsilon_1] \leq \frac{\mathrm{E}[Y_{i,j}]}{\varepsilon_1} < \frac{2e^{-\varepsilon_2^2 t/2(p-1)^2}}{\varepsilon_1}. \tag{21}$$

The above inequality is valid for a fixed j, and it can be used to obtain the probability that U^i is good:

$$\Pr[U^i \text{ is good}] = \Pr\left[\bigcap_{j=1}^{p} Y_{i,j} \leq \varepsilon_1\right] \geq 1 - \frac{2pe^{-\varepsilon_2^2 t/2(p-1)^2}}{\varepsilon_1}. \tag{22}$$

Finally, we are to determine a suitable value for t, in order to make this probability large enough. If we choose

$$t \geq \frac{2(p-1)^2}{\varepsilon_2^2} \ln \frac{2\ell p}{\delta \varepsilon_1}, \tag{23}$$

the probability that U^i is good will be at least $1 - \delta/\ell$.

Corollary 1. *For any $\delta > 0$ it is possible to choose the constant t in such a way that the probability of all U^i being good is at least $1 - \delta$.*

Proof. There are ℓ different U^i.

We construct an assignment Π to the variables in V^i by step 3 in Algorithm 1. If U^i is good, we expect the number of equations satisfied by H to be close to the number of equations satisfied by Π. To formalize this intuition, we need the following definitions.

Definition 8.

$$\mu(\tau) = \frac{\#\textit{ equations satisfied by the assignment } \tau}{n^2}. \tag{24}$$

Definition 9. *We say that a variable $v \in V^i$ is* unbalanced *if there exists a $j^* \in \boldsymbol{Z}_p$ such that for all $j' \in \boldsymbol{Z}_p \setminus \{j^*\}$*

$$\frac{S(V \setminus V^i, H, v \leftarrow j^*)}{n - |V^i|} \geq \frac{S(V \setminus V^i, H, v \leftarrow j')}{n - |V^i|} + \varepsilon_3. \tag{25}$$

Lemma 2. *Let $\pi = H|_U$ and Π be the assignment produced with this choice of π in step 3 of Algorithm 1. Denote by H' the assignment which assigns to a variable v the value $H(v)$ if $v \in V \setminus V^i$ and $\Pi(v)$ if $v \in V^i$. Then, if U^i is good, it is for any $\varepsilon > 0$ possible to choose the constant ℓ in such a way that*

$$\mu(H') \geq \mu(H) - \varepsilon/p\ell. \tag{26}$$

Proof. We want to compare the number of equations satisfied by the assignment H to the number satisfied by H'. In particular, we want to bound the decrease in the number of satisfied equations. As only the values assigned to variables in V^i can differ between the two assignments, the possible sources of aberrations are the equations where variables in V^i are involved. We have four different cases:

1. Equations of the type $av_1 + bv_2 = c$, where $v_1, v_2 \in V^i$. There are less than $p(p-1)n^2/2\ell^2$ such equations, and at most $(p-1)n^2/2\ell^2$ can be satisfied by any given assignment. The decrease is thus less than $pn^2/2\ell^2$.
2. Equations of the form $av + bw = c$ where $v \in V^i$ is unbalanced and satisfies Eq. 6, and $w \notin V^i$. If we combine Eq. 6 and Eq. 25 we obtain the inequality

$$\frac{S(U^i, \pi, v \leftarrow j^*)}{t} \geq \frac{S(U^i, \pi, v \leftarrow j')}{t} + \varepsilon_3 - 2\varepsilon_2 \quad \text{for all } j'. \tag{27}$$

   By the construction of Algorithm 1, the value chosen for v will thus be the correct value, provided that $\varepsilon_3 > 2\varepsilon_2$. It follows that the number of satisfied equations of this type cannot decrease.
3. Equations of the form $av + bw = c$ where $v \in V^i$ is *not* unbalanced, but still satisfies Eq. 6, and $w \notin V^i$. In this case, Algorithm 1 may select the wrong assignment to v. However, that cannot decrease the optimum value by much. For, suppose that $v = j$ in the optimal solution, but the algorithm happens to set $v = j'$. The reason for that can only be that $S(U^i, \pi, v \leftarrow j') \geq S(U^i, \pi, v \leftarrow j)$. By Eq. 6, this implies that

$$\left| \frac{S(V \setminus V^i, H, v \leftarrow j')}{n - |V^i|} - \frac{S(V \setminus V^i, H, v \leftarrow j)}{n - |V^i|} \right| \leq 2\varepsilon_2. \tag{28}$$

   Since there are at most $|V^i|$ different v that are not unbalanced, we can bound the decrease in the number of satisfied equations by

$$|V^i|\big(S(V \setminus V^i, H, v \leftarrow j') - S(V \setminus V^i, H, v \leftarrow j)\big) \leq 2\varepsilon_2 n^2/\ell. \tag{29}$$

4. Equations of the form $av + bw = c$ where $v \in V^i$ does not satisfy Eq. 6, and $w \notin V^i$. By construction there are at most $\varepsilon_1|V^i|$ such variables in V^i. The number of equations of this type is thus less than $\varepsilon_1 p^2|V^i|n$. Only $\varepsilon_1 p|V^i|n$ of these can be satisfied at the same time, and hence a bound on the decrease is $\varepsilon_1 p|V^i|n = \varepsilon_1 pn^2\ell$.

Summing up, the total decrease is at most

$$pn^2/2\ell^2 + 2\varepsilon_2 n^2/\ell + \varepsilon_1 pn^2\ell. \tag{30}$$

If we select $\ell = p^2/\varepsilon$, $\varepsilon_1 = \varepsilon/4p$, $\varepsilon_2 = \varepsilon/8$, and $\varepsilon_3 = \varepsilon/3$, the total decrease is at most

$$\varepsilon n^2/2p\ell + \varepsilon n^2/4p\ell + \varepsilon n^2/4p\ell = \varepsilon n^2/p\ell, \tag{31}$$

which concludes the proof.

Corollary 2. *If all U^i are good and we construct from an assignment $\pi = H|_U$ a new assignment Π as in step 3 of Algorithm 1, it is for all $\varepsilon > 0$ possible to choose the constant ℓ in such a way that $\mu(\Pi) \geq \mu(H) - \varepsilon/p$.*

Proof. Let $H_0 = H$ and H_i, $i = 1, 2, \ldots, \ell$, satisfy $H_i|_{V^i} = \Pi|_{V^i}$ and $H_i|_{V \setminus V^i} = H_{i-1}|_{V \setminus V^i}$. Apply Lemma 2 ℓ times, once for each H_i. This corresponds to the way Algorithm 1 works.

The only observation needed now is that since all results above are valid for *any* choice of the assignment H, they are in particular valid when H is the assignment producing the optimum number of satisfied equations.

Theorem 1. *For instances of Max E2-Lin mod p where the number of equations is $\Theta(n^2)$, Algorithm 1 is a randomized polynomial time approximation scheme.*

Proof. Consider the case when the assignment H is the optimal assignment. Since all possible assignments π to the variables in the set U are tried by the algorithm, the optimal assignment H restricted to U will eventually be tried. Corollaries 1 and 2 show that the algorithm produces, with probability at least $1-\delta$, an assignment Π such that $\mu(\Pi) > \mu(H) - \varepsilon/p$. An additive error of ε/p in $\mu(\tau)$ translates into an additive error of $\varepsilon n^2/p$ for the equation problem. Since the optimum of a Max E2-Lin mod p instance with cn^2 equations is at least cn^2/p, this gives a relative error of at most ε/c.

4 The general case

The algorithm described in the previous section is easily generalized to handle instances of Max k-Function Sat mod p as well — it does not use any special feature of the Max E2-Lin mod p problem. As for Max E2-Lin mod p, we assume that the set of functions does not contain any redundancy — all functions are assumed to be different. This is actually a weaker constraint than the one imposed on Max E2-Lin mod p; in the context of Max k-Function Sat mod p problems, $ax_i + bx_{i'} - c$ and $adx_i + dbx_{i'} - dc$ (for $d \notin \{0, 1\}$) are considered distinct functions whereas the corresponding Max E2-Lin mod p equations would be considered identical.

The analysis assumes that all functions in the instance are satisfiable. This is needed to ensure that the optimum of an instance with cn^k functions is at least cn^k/p^k.

We can adopt the techniques used in the proofs for the Max E2-Lin mod p case to this more general case with some minor modifications.

Definition 10. *We extend the notation $S(X, \tau, x \leftarrow r)$ to mean the number of satisfied functions with $k-1$ variables from the set X and one variable $x \notin X$, given that the variables in X are assigned values according to the function τ and x is assigned the value r.*

Definition 11. *We say that the set U^i is* good *if for all except a fraction of at most ε_1 of the variables $v \in V^i$*

$$\left| \frac{S(U^i, H, v \leftarrow j)}{\binom{t}{k-1}} - \frac{S(V \setminus V^i, H, v \leftarrow j)}{\binom{n-|V^i|}{k-1}} \right| \leq \varepsilon_2 \quad \text{for all } j \in \boldsymbol{Z}_p. \tag{32}$$

Lemma 3. *For any $\delta > 0$, it is possible to choose the constant t in such a way that the probability of a set U^i being good is at least $1 - \delta/\ell$ for a fixed i.*

Proof (sketch). Fix a variable $v \in V^i$ and some $j \in \boldsymbol{Z}_p$. If we introduce, for each $\boldsymbol{w} \subseteq V \setminus V^i$ such that $|\boldsymbol{w}| = k-1$, a Bernoulli random variable $\xi_{i,j,v,\boldsymbol{w}}$ with the property that

$$\xi_{i,j,v,\boldsymbol{w}} = \begin{cases} 1 & \text{if } \boldsymbol{w} \subseteq U^i, \\ 0 & \text{otherwise,} \end{cases} \tag{33}$$

we can write

$$S(U^i, H, v \leftarrow j) = \sum_{\substack{\boldsymbol{w} \subseteq V \setminus V^i \\ |\boldsymbol{w}| = k-1}} S(\boldsymbol{w}, H, v \leftarrow j) \xi_{i,j,v,\boldsymbol{w}}. \tag{34}$$

There are p^{p^k} functions from $\boldsymbol{Z}_p^k$ to $\boldsymbol{Z}_p$ and a fraction $1/p$ of these are satisfied simultaneously, which implies that

$$0 \leq S(\boldsymbol{w}, H, v \leftarrow j) \leq p^{p^k - 1}. \tag{35}$$

To simplify the notation, we put

$$T = \binom{t}{k-1}. \tag{36}$$

As in the proof of Lemma 1, we define

$$X_{i,j,v} = \frac{S(U^i, H, v \leftarrow j)}{T}. \tag{37}$$

The sum in Eq. 34 contains T non-zero terms. We construct a martingale $X_{i,j,v}^0, \ldots, X_{i,j,v}^T$ by conditioning on these terms, as in the proof of Lemma 1. The Lipschitz condition in Eq. 15 then changes to

$$|X_{i,j,v}^m - X_{i,j,v}^{m-1}| \leq p^{p^k - 1}/T \quad \text{for all } m \in \{1, \ldots, T\}. \tag{38}$$

By choosing

$$t \geq k - 2 + \left(\frac{2(k-1)!p^{2(p^k-1)}}{\varepsilon_2^2} \ln \frac{2\ell p}{\delta \varepsilon_1} \right)^{\frac{1}{k-1}}. \tag{39}$$

it can be verified that the probability that U^i is good is at least $1 - \delta/\ell$.

Lemma 4. *Let $\pi = H|_U$ and Π be the assignment produced with this choice π in step 3 of Algorithm 1. Denote by H' the assignment which assigns to a variable v the value $H(v)$ if $v \in V \setminus V^i$ and $\Pi(v)$ if $v \in V^i$. Then, if U^i is good, it is for any $\varepsilon > 0$ possible to choose the constant ℓ in such a way that*

$$\mu(H') \geq \mu(H) - \varepsilon/p^k\ell. \tag{40}$$

Proof. To bound the decrease in the number of satisfied functions, we perform a case analysis similar to that in the proof of Lemma 2.

1. Functions which depend on more than one variable from V^i. At most
$$\frac{p^{p^k-1}n^k}{\ell^2} \tag{41}$$
such functions can be satisfied by any given assignment.
2. Functions depending on $v \in V^i$ but not on any other variable in V^i where v is unbalanced and satisfies Eq. 32. As in the proof of Lemma 2, the number of satisfied functions of this type does not decrease if $\varepsilon_3 \geq 2\varepsilon_2$.
3. Functions depending on $v \in V^i$ but not on any other variable in V^i where v is not unbalanced but satisfies Eq. 32. In this case Algorithm 1 can select the wrong assignment to v. Suppose that $v = j$ in the optimal solution but that the algorithm sets $v = j'$. This implies that $S(U^i, \pi, v \leftarrow j') \geq S(U^i, \pi, v \leftarrow j)$ and by Eq. 32
$$\left| \frac{S(V \setminus V^i, H, v \leftarrow j')}{\binom{n-|V^i|}{k-1}} - \frac{S(V \setminus V^i, H, v \leftarrow j)}{\binom{n-|V^i|}{k-1}} \right| \leq 2\varepsilon_2. \tag{42}$$
Since there are at most $|V^i|$ different v that are not unbalanced, we can bound the decrease in the number of satisfied functions by
$$|V^i|\big(S(V \setminus V^i, H, v \leftarrow j') - S(V \setminus V^i, H, v \leftarrow j)\big) \leq \frac{2\varepsilon_2 n^k}{(k-1)!\ell}. \tag{43}$$
4. Functions depending on $v \in V^i$ but not on any other variable in V^i where v does not satisfy Eq. 32. By construction there are at most $\varepsilon_1|V^i|$ such variables in V^i. The number of functions of this type is thus less than $\varepsilon_1|V^i|p^{p^k}n^{k-1}/(k-1)!$. Only $\varepsilon_1|V^i|p^{p^k-1}n^{k-1}/(k-1)! = \varepsilon_1 p^{p^k-1}n^k/\ell(k-1)!$ of these can be satisfied at the same time, which gives us a bound on the decrease.

Summing up, the total decrease is

$$\frac{p^{p^k-1}n^k}{\ell^2} + \frac{2\varepsilon_2 n^k}{(k-1)!\ell} + \frac{\varepsilon_1 p^{p^k-1}n^k}{(k-1)!\ell}. \tag{44}$$

By choosing

$$\ell = 2p^{p^k+k-1}/\varepsilon, \tag{45}$$

$$\varepsilon_1 = \varepsilon(k-1)!/4p^{p^k+k-1}, \tag{46}$$

$$\varepsilon_2 = \varepsilon(k-1)!/8p^k, \tag{47}$$

$$\varepsilon_3 = \varepsilon(k-1)!/3p^k, \tag{48}$$

the total decrease becomes at most $\varepsilon n^k/p^k\ell$.

Theorem 2. *For instances of Max k-Function Sat mod p where the number of satisfiable functions is $\Theta(n^k)$, Algorithm 1 is a randomized polynomial time approximation scheme.*

Proof. Follows that of Theorem 1 with the only difference being that the optimum of a Max k-Function Sat mod p instance with cn^k satisfiable functions is at least cn^k/p^k.

5 Conclusions

We have shown how to construct a randomized polynomial time approximation scheme for dense instances of Max k-Function Sat mod p. The algorithm is intuitive, and shows the power of exhaustive sampling. The running time of the algorithm is quadratic in the number of variables, albeit with large constants. Using methods similar to those of Goldreich, Goldwasser and Ron [5], we can convert our algorithm into a randomized constant time approximation scheme. The algorithms in this scheme only compute numerical approximations to the optimum, they do not construct assignments achieving this optimum.

As a special case, Theorem 2 implies the existence of a polynomial time approximation scheme for dense instances of Max E3-Lin mod p. This result is interesting when compared to the lower bounds found by Håstad [6] for systems of equations with at least 3 variables in each equation: In the general case, there is no polynomial time approximation algorithm achieving a performance ratio of $p-\varepsilon$ for any $\varepsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$. Zwick [10] studied the problem of finding almost-satisfying assignments for almost-satisfiable instances of some constraint satisfaction problems. Also for this restricted problem, approximating Max Ek-Lin mod 2 (in the sense defined by Zwick) is hard by the results of Håstad [6].

It is well known [9, Theorem 13.8] that a **Max-SNP** problem can be viewed as a Max k-Function Sat problem for some fixed integer k. Arora, Karger and Karpinski [1] call an instance of a **Max-SNP** problem dense if the instance of Max k-Function Sat produced using it has $\Omega(n^k)$ functions. We have shown in this paper that there are natural extensions of these concepts to functions mod p. This extends the notion of denseness also to non-boolean optimization problems.

6 Acknowledgment

We would like to thank Alessandro Panconesi for pointing out an oversight in an earlier version of this manuscript and for valuable comments regarding the exposition of the material.

References

1. Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proc. Twenty-seventh Ann. ACM Symp. on Theory of Comp.*, pages 284–293. ACM, New York, 1995.
2. Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19:357–367, 1967.
3. Wenceslas Fernandez de la Vega. MAX-CUT has a randomized approximation scheme in dense graphs. *Random Structures and Algorithms*, 9:93–97, 1996.
4. Alan Frieze and Ravi Kannan. Quick approximation to matrices and applications. Manuscript, July 1997.
5. Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. In *Proc. of 37th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 339–348. IEEE Computer Society, Los Alamitos, 1996.
6. Johan Håstad. Some optimal inapproximability results. In *Proc. Twenty-ninth Ann. ACM Symp. on Theory of Comp.*, pages 1–10. ACM, New York, 1997.
7. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
8. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, 1995.
9. Christos H. Papadimitriou. *Computational Complexity.* Addison Westley, Reading, 1994.
10. Uri Zwick. Finding almost-satisfying assignments. In *Proc. Thirtieth Ann. ACM Symp. on Theory of Comp.*, pages 551–560. ACM, New York, 1998.

Second-Order Methods for Distributed Approximate Single- and Multicommodity Flow

S. Muthukrishnan[1] and **Torsten Suel**[2,3]

[1] Bell Laboratories, 700 Mountain Avenue, Murray Hill, NJ 07974.
muthu@research.bell-labs.com.
[2] Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201.
suel@photon.poly.edu.
[3] This work was done while the author was at Bell Labs.

Abstract. We study local-control algorithms for maximum flow and multicommodity flow problems in distributed networks. We propose a second-order method for accelerating the convergence of the "first-order" distributed algorithms recently proposed by Awerbuch and Leighton. Our experimental study shows that second-order methods are significantly faster than the first-order methods for approximate single- and multicommodity flow problems. Furthermore, our experimental study gives valuable insights into the diffusive processes that underly these local-control algorithms; this leads us to identify many open technical problems for theoretical study.

1 Introduction

The *multicommodity flow problem* is the problem of simultaneously shipping multiple commodities through a capacitated network such that the total amount of flow on each edge is no more than the capacity of the edge. Each commodity i has a source node, a sink node, and an associated *demand* d_i, which is the amount of that commodity that must be shipped from its source to its sink. The objective is to find a flow that meets the individual demands of all the commodities without exceeding any edge capacity (finding a *feasible flow*)[4]. The case when there is only a single commodity and the goal is to maximize the feasible flow is the well known *maximum flow problem*. The importance of the single- and multicommodity flow problems need hardly be stressed – a substantial body of work in Algorithms and Operations Research is devoted to these problems.

In this paper, we focus on local-control (or distributed) algorithms for the single- and multicommodity flow problems. Besides their inherent interest, local-control algorithms for these problems are relevant because of the following reasons:

(1) Many routing, communication, and flow-control problems between multiple senders and receivers, including various uni/broad/multicasts, can be modeled as multicommodity flow problems on networks (e.g., see the references in [BG91, BT89, AL93, AL94, AAB97]). These applications typically require online, local-control (distributed) algorithms, since global communication and control is expensive and cumbersome. Local algorithms for

[4] An alternate objective is to maximize z such that the flow satisfies a percentage z of every demand without exceeding any edge capacity (called the *concurrent flow problem* [SM86]); we do not consider this version here.

M. Luby, J. Rolim, and M. Serna (Eds.): RANDOM'98, LNCS 1518, pp. 369-384, 1998.

multicommodity flow not only provide a generic solution to these problems, but they also give valuable insights for the centralized/global solution of these problems.

(2) The best currently known algorithms for maximum flow and multicommodity flow problems are fairly sophisticated (see, e.g., [GR97, GT88, K97, LM+91, V89]), and typically rely on augmenting paths, blocking flows, min-cost flows, or linear programming. In contrast, local-control algorithms are appealingly simple, as they rely on simple "edge balancing" strategies of appropriately balancing commodities between adjacent nodes (details below). Thus, they are easy to implement, understand, and experiment with.

(3) Local-control algorithms have several other attractive features. For example, they adjust gracefully to dynamic changes in the topology (e.g., link failures) and the traffic demands (e.g., bursty multicasts) in communication networks. They are *iterative*, that is, running them longer gives progressively better approximations to the optimal solution. Hence, one can use them either for rapid coarse solutions or for slow refined solutions. Finally, they may expose alternate structure in the problem, as the convergence of such local-control algorithms is typically related to the eigenstructure of the network (for intuition, see [C89]).

1.1 First-Order Algorithms

Local-control algorithms for the multicommodity flow problem were recently designed by Awerbuch and Leighton [AL93, AL94]. Their algorithms proceed in parallel rounds. At the start of a round, (approximately) d_i units of commodity i are added to the source node of that commodity, where d_i is the demand of commodity i. The commodities accumulated in each node are then distributed equally among the local endpoints of the incident edges, and flow is pushed across each edge of the network so as to "balance" each commodity between the two endpoints of the edge (subject to edge capacity constraints). Finally, any commodity that has arrived at the appropriate sink is removed from the network. How to trade off the flow between different commodities that compete for the capacity of an edge is nontrivial. Awerbuch and Leighton proved in [AL93, AL94] that this simple "edge balancing" algorithm (and some of its variants) converges and, maybe somewhat surprisingly, that it provides a provably approximate solution to the multicommodity flow problem in a small number of rounds.

We refer to such edge-balancing algorithms as *first-order algorithms*. The first-order algorithms in [AL93, AL94] can clearly be implemented on a distributed network in which each node communicates only with neighboring nodes and has no global knowledge of the network.[5] Similar local-control algorithms have been designed for several other problems [LW95], including distributed load balancing [C89, AA+93, MGS98] and end-to-end communication [AMS89].

A particularly simple local-control algorithm can be obtained for the case of the maximum flow problem by specializing the first-order algorithm in [AL93, AL94] for the single-commodity case. There are many other algorithms for the maximum flow problem, but none that is a distributed first-order algorithm. The algorithm most closely related in spirit is the algorithm of Goldberg and Tarjan in [GT88], where a "preflow" is adjusted into a flow by pushing excess local flow towards the sink along estimated shortest paths. However, this algorithm needs to maintain estimated shortest-path information and is thus less amenable to a distributed, local-control implementation in dynamic networks.

[5] In contrast, other approximation algorithms for the multicommodity flow problem rely on global computations [V89, LM+91].

1.2 Second-Order Algorithms

In this paper, we initiate a new direction in distributed flow algorithms aimed at speeding up the first-order algorithms of [AL93, AL94] for the multicommodity flow problem. The basic idea is that in any round, we use the knowledge of the amount of flow that was sent across the edge in the previous round in order to appropriately adjust the flow sent in the current round. Specifically, for a parameter β, the flow sent across an edge is chosen as β times what would be sent by the first-order algorithm, plus $\beta - 1$ times what was actually sent across the edge in the previous round. (A more detailed description of these methods is given in Sections 3 and 4.)

We call algorithms derived in this manner *second-order algorithms*. Perhaps surprisingly, the main conclusion of this paper is that second-order algorithms appear to substantially outperform their first-order counterparts for maximum flow and multicommodity flow problems, as shown by our experiments.

1.3 Background and Related Work

First-Order methods. The first-order algorithm of Awerbuch and Leighton for the maximum flow problem is conceptually similar to the probabilistic phenomena of diffusion and random walks. The algorithm works based on diffusion since the excess flow always flows down the gradient along each edge. For simpler problems such as distributed load balancing, if one considers the vector of flows accumulated at the nodes as iterations progress, they can be modeled as transitions of a Markov Chain, or a suitable random walk [C89]. However, for the general multicommodity flow problem, these conceptual similarities have not yet been formalized. The analysis of Awerbuch and Leighton is sophisticated even for the case of the maximum flow problem. It does not rely on Markov Chain methods, and is entirely combinatorial.

First-order algorithms for flow problems are also related to matrix-iterative methods for solving linear systems, and in particular, the Gauss-Seidel iterations. This connection is made explicit in [BT89]. Also, there is a way to interpret the first-order algorithms as iteratively solving a dual network optimization problem involving a single variable per node. At each iteration, the dual variables of a single node or its incident edge flows are changed in an attempt to improve the dual cost. This process is also explained in [BT89].

Thus, there are intriguing connections between the first-order methods for flow problems and classical techniques such as matrix-iterative methods, diffusion, random walks and primal-dual relaxations. These techniques have been studied in different areas with somewhat different emphasis, but seem directly relevant to the work in [AL93, AL94].

Second-Order methods. Second-order algorithms, as described above, may seem ad-hoc, and further explanation is needed to motivate them. Our second-order algorithms are motivated by the observation that the first-order flow algorithms in [AL93] are iterative methods reminiscent of the matrix-iterative methods used for solving systems of linear equations. There is already a mature body of knowledge about speeding up these first-order methods (see, e.g., [A94, BB+93, HY81, Var62]). Very recently, these methods were explored for speeding up diffusive load-balancing schemes [MGS98]. Of the many known iterative techniques, the authors in [MGS98] identified a specific second-order scheme best suited for distributed implementations, and our second-order scheme for the multicommodity flow problem is inspired by that method.

There are fundamental similarities between our work here and the work in [MGS98] for distributed load balancing, but there are fundamental differences as well. The basic similarity is that our algorithmic strategy for second-order methods relies on the same stationary acceleration of the first-order method determined by a parameter β (fixed throughout all iterations) as that in [MGS98]. The main difference arises in the fact that the problem of multicommodity flow is much more general than the distributed load-balancing problem considered in [MGS98]. First, the edges in our problem have capacity constraints, while the edge capacities are unbounded in the load-balancing problem. Second, our algorithms are *dynamic* in that they introduce new flow in each round as described in Section 3; in contrast, the total load remains unchanged in [MGS98]. There are other differences (such as the fact that we do not use *IOUs* as in [MGS98]), but we omit these details.

The similarity of iterative flow algorithms to matrix-iterative methods and distributed load balancing is helpful. In particular, known results [Var62] show that $0 < \beta < 2$ is the *only* suitable range for the convergence of that iterative method. Furthermore, from the results in [MGS98], we would expect that the second-order method will be outperformed by the first-order method for $0 < \beta < 1$, and thus the fruitful range for β is $(1, 2)$; as we will see, this also holds for distributed flow problems.[6] However, the above mentioned differences explain the considerable difficulty in analyzing the first-order and second-order method for the multicommodity flow problem [AL93, AL94]. The first-order method for distributed load balancing can be analyzed fairly easily based on stationary Markov Chain methods [C89], and known second-order analyses for matrix-iterative methods can be fairly easily adopted to load balancing [MGS98]. However, standard approaches (e.g., based on Dirichlet boundary conditions [C97] for analyzing dynamic situations) do not seem to apply if edges have capacity constraints.

1.4 Contents of this Paper

In this paper, we propose second-order methods for accelerating the distributed flow algorithms proposed by Awerbuch and Leighton [AL93, AL94]. We perform an experimental study and show that the second-order algorithms are significantly faster than the first-order ones of [AL93, AL94] both for the maximum flow and the multicommodity flow problems. This is of possible applied interest as an online distributed solution for many routing problems arising in communication networks. Surprisingly, our algorithms seem to be of interest in the off-line, centralized context as well. While our algorithms are not as fast as the best known algorithms for the maximum flow problem, they seem to be at least competitive with (and possibly much faster than) the best known algorithms for the approximate multicommodity flow problem. This is a bit surprising since the best known centralized algorithms for the multicommodity flow problem [LM+91] use sophisticated techniques; in contrast, the first-order and second-order algorithms are exceedingly simple.

Our experimental study also leads to a number of observations and conjectures about the behavior of the diffusive processes used in the first- and second-order flow algorithms. We describe some of these as open problems for theoretical study.

[6] See [MGS98, Var62] for results on choosing the "best" value of β, and [DMN97] for choosing the best β for distributed load-balancing as a function of the graph structure. We plan to perform an experimental study of the best choices of β for flow problems on different classes of input graphs in the near future.

The remainder of this paper is organized as follows. The next section provides some definitions and notations used throughout the paper. Section 3 describes the first-order and second-order methods for the maximum flow problem, and presents a variety of experimental results. These results also give intuition to the reader about the behavior of first- and second-order algorithms for flow problems. Section 4 describes the algorithms and experimental results for the case of multicommodity flow, and they are more interesting in terms of comparative performance. A few open questions appear in Section 5.

We have a fully functional implementation with a graphical interface for vizualizing the behavior of our algorithms. Some additional information about our implementation and the input instances used in our experiments is contained in the appendix.

2 Preliminaries

Throughout this paper, we assume a network (or graph) $G = (V, E)$ with n nodes and m edges. We assume a model of the graph in which each edge e in the network has one capacity $c_1(e) \geq 0$ in one direction, and another capacity $c_2(e) \geq 0$ in the other direction.[7] Each node v has one queue for each incident edge. This queue can hold an unbounded amount of flow (or commodity), and should be considered as being located at the endpoint v of the edge.

In the case of the maximum flow problem, we are given a source node s and a sink node t, and our goal is to maximize the flow between s and t. In the multicommodity flow problem with k commodities, we are given k source/sink pairs (s_i, t_i) and corresponding demands d_i, and we are interested in finding a flow that satisfies the demands of all commodities, if such a flow exists. In the description of the algorithms, we use $\Delta_i(e)$ (or $\Delta(e)$ in the single-commodity case) to denote the difference between the amounts of commodity i located in the queues at the two endpoints of edge e.

3 Maximum Flow

In this section, we focus on the maximum flow problem. This special case of the multicommodity flow problem leads to particularly simple and efficient versions of the first-order and second-order methods. In the first subsection, we describe the first-order local-control algorithm for maximum flow. In Subsection 3.2 we explain our new second-order method, while Subsection 3.3 presents and discusses our experimental results.

3.1 First-Order Distributed Maximum Flow

We now describe the first-order algorithm for maximum flow. The algorithm proceeds in a number of synchronous parallel rounds (or iterations), where in each round, a small set of elementary operations is performed in each node and each edge of the network. In particular, each round consists of the following steps.

[7] Thus, each edge is equivalent to two directed edges with their own capacities $c_1(e)$ and $c_2(e)$. However, our algorithms and implementations also extend to a graph model where the capacity of each edge is shared between the two directions.

(1) Add d units of flow to the source node, where d is chosen as the sum of the capacities of the outgoing edges (or some other upper bound on the value of the maximum flow).
(2) In each node v, partition the flow that is currently in the node evenly among the $\delta(v)$ local queues of the $\delta(v)$ incident edges.
(3) In each edge e, attempt to balance the amount of commodity between the two queues at the endpoints of the edge, by routing $min\{\frac{\Delta(e)}{2}, c(e)\}$ units of flow across the edge, where $\Delta(e)$ is the difference in the amount of flow between the two queues, and $c(e)$ is the capacity of the edge in the direction from the fuller to the emptier queue.
(4) Remove all flow that has reached the sink from the network.

We point out that this algorithm is a simplified version of the algorithm in [AL93] for the single-commodity case; the simplification results from the fact that we do not have to resolve any contention between different commodities. One consequence is that the algorithm correctly finds the maximum flow even if d is much larger than the value of that flow, that is, the algorithm does not rely on the existence of a feasible flow of value d.

3.2 Second-Order Distributed Maximum Flow

We now describe how to obtain a second-order method for distributed maximum flow. As already mentioned in the introduction, the second-order method computes the flow to be sent across an edge in the current round as a linear combination of the flow that would be sent according to the first-order method and the flow that was sent in the previous iteration. The second-order method has an additional parameter β, with the case $\beta = 1.0$ being identical to the first-order method. More precisely, Step (3) of the above algorithm becomes:

(3a) In each edge e, compute the desired flow across the edge as

$$f = \beta \cdot \frac{\Delta(e)}{2} + (\beta - 1) \cdot f',$$

where $\Delta(e)$ is defined as before, and f' is the (possibly negative) amount of flow that was sent in the direction of the imbalance, in the previous iteration.
(3b) Obtain the amount of flow actually sent across the edge by adjusting f for the capacity of the edge, and for the amount of commodity available at the sending queue.

Note that the value of f computed in Step (3a) can not only exceed the available edge capacity, but may also be larger than the amount of commodity available at the sending queue.

Idealized and Realistic Versions. We distinguish two cases depending on how Step (3b) is handled if the amount of commodity available at the sending queue is smaller than the flow to be sent across that edge as calculated in Step (3a). In the *idealized* algorithm, we treat the flow accumulated at each node as just some (possibly negative) number, and we send out as much flow as the capacity constraint permits even if the amount of commodity stored at a sending queue becomes negative as a result. In the *realistic* algorithm, we treat the flows as physical flows and therefore, flows at nodes may only be non-negative. Thus, we send out the minimum of the flow calculated in Step (3a), the capacity of the edge, and the flow in the sending queue.

We expect the idealistic algorithm to converge faster, and in general, have smoother convergence properties than the realistic algorithm. In order to solve the standard sequential maximum flow problem, it suffices to implement the idealized case. However, if we want to solve

the flow problem online in a distributed environment as flow continuously enters the source, the realistic algorithm must be employed. In what follows, our experimental results are for the realistic algorithm unless stated otherwise.

3.3 Experimental Evaluation

In this subsection, we present a number of experimental results on the behavior of the first-order and second-order methods. Due to space constraints, we cannot hope to provide a detailed study of the behavior of the methods on different classes of input graphs. Instead, we present a few selected results that illustrate the most interesting aspects of the behavior of the algorithm, and provide a brief summary of other results at the end. Some information about our implementation, and about the graphs used in the experiments, can be found in the appendix.

Dependence on β We first look at the performance of the second-order method for different values of the parameter β. Figure 3.3 shows the flow arriving at the sink in each time step, for several choices of β ranging from 1.0 to 1.95, using a 20-level mesh graph with 402 nodes and 1180 edges. The results in Figure 3.3 show that the rate of convergence increases significantly as we increase β from 1.0 to 1.95. In particular, after 1500 iterations, the first-order method ($\beta = 1.0$) is still more than 10% away from the exact solution. In contrast, the second-order method with $\beta = 1.95$ has already converged to within 0.001%, and with a few thousand more iterations it reaches essentially floating point precision.

Figure 3.3 shows the behavior of the algorithms for very small and very large values of β. In particular, we see that for $\beta = 0.5$ the performance of the algorithm becomes even worse than in the first-order method, while for $\beta = 2.5$, the method becomes unstable, and does not converge to a final value. We point out that we observed a similar overall behavior on all the graphs that we tested, with very rapid convergence for the best values of β (usually, but not always, around 1.9), slower convergence for smaller values of β, and instability as we increase β beyond 2.0.

In general, the "optimal" β, namely, the one that gives the fastest convergence is probably a complex function of the eigenstructure of the underlying graph. This is provably the case in second-order methods for the distributed load balancing problem [MGS98]. Although in many of the examples we show here, the optimal β is large (around 1.95), there are cases when a smaller value of β is preferable; see Section 4.2 for one such example.

Convergence of Edge Flows The results in Figure 3.3 indicate a very rapid convergence of the amount of flow that arrives at the sink. However, this does not directly imply that all the flows inside the network converge to a steady state. To investigate whether this is the case, we define the *flow change norm* as the sum, over all edges, of the absolute value of the change in flow between the current and the previous iteration. Thus, if this norm converges to zero, then the network converges to a steady flow state.

Figure 3.3 shows the behavior of this norm for β equal to 1.0, 1.5, and 1.95, for the mesh graph considered before. As can be seen, the flow change norm converges to zero. Convergence is again most rapid for values of β around 1.9. Note that for the first 150 or so iterations, the flow change norm for $\beta = 1.95$ is actually larger than that of the other curves, indicating a

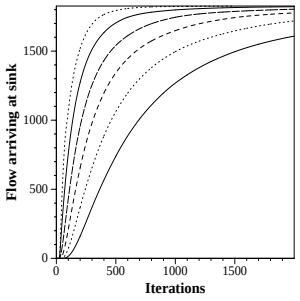


Fig. 1. Convergence of the second-order method with β set to 1.0 (lower curve), 1.2, 1.4, 1.6, 1.8, and 1.95 (upper curve).

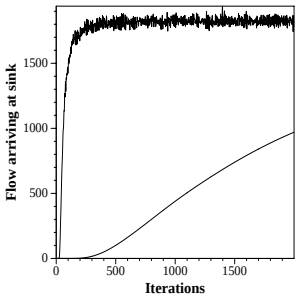


Fig. 2. Behavior for $\beta = 2.5$ (upper curve) and $\beta = 0.5$ (lower curve).

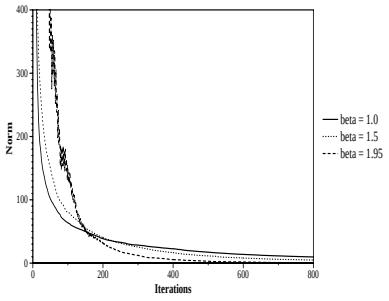


Fig. 3. Convergence of the flow change norm for β equal to 1.0, 1.5, and 1.95.

faster initial response to the injected flow. A similar rapid convergence behavior of the flows was observed in all our experiments.

The convergence of the flows is significant because it allows us to directly use the stabilized flow in the network as an approximate solution for the standard offline maximum flow problem, instead of computing the flow by averaging out the history of the edge flows, as suggested in [AL93]. Averaging the history implies the algorithm must be run for a much longer period to obtain a good approximation since the approximation ratio is then given by the ratio of the area under the curve and the area under the horizontal line at the height of the maximum flow.

Idealized Second-Order Method Recall that in Step (3b) of the second-order method, we may have to adjust the amount of flow sent across an edge in order to avoid getting a negative amount of commodity in the sending queue. In the following, we investigate how the behavior of the algorithm changes if we allow negative amounts of commodity at the nodes, that is, we consider the idealized second-order method described in Subsection 3.2, which does not adjust the flow for the amount of available commodity.

Figure 3.3 shows the convergence of the idealized and realistic methods for different values of β, for the mesh graph considered before. Note that for $\beta = 1.95$, the flow converges to more than 15 digits of accuracy in less than 1000 iterations. If we increase β further towards 2.0 we notice that the flow starts oscillating more extremely, and for values beyond 2.0 the method does not converge anymore. Figure 3.3 shows the behavior of the idealized method for the case of $\beta = 2.0$. (For the realistic method, this effect appears to be slightly less abrupt in that the method becomes instable more slowly as we increase β beyond 2.0.)

Note that whether allowing negative amounts of commodity at the nodes is appropriate or not depends on the particular application. If the goal is just to find a solution to the maximum flow problem, and the actual routing of the commodities is done in a separate phase afterwards, then the idealized version is fine. On the other hand, a major advantage of the distributed methods is that they overlap the process of finding the flow paths with that of routing the commodities, in which case the idealized version is not appropriate.

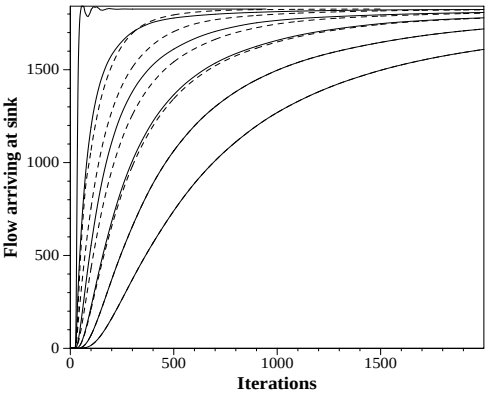


Fig. 4. Convergence of the idealized (solid lines) and realistic (dashed lines) second-order method with β equal to 1.0 (lower curve), 1.2, 1.4, 1.6, 1.8, and 1.95 (upper curve). Note that the two lowest dashed curves are hidden by the corresponding solid curves.

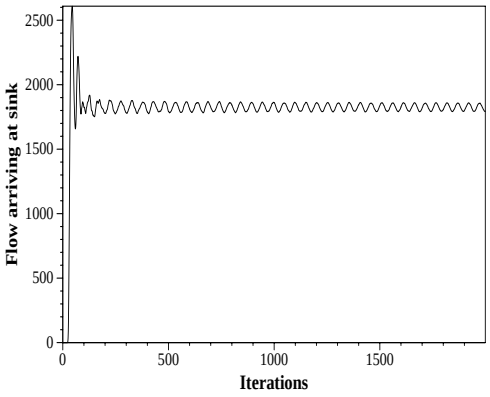


Fig. 5. Behavior of the idealized second-order method with $\beta = 2.0$.

4 Multicommodity Flow

In this section, we consider the case of multiple commodities. We first outline the first-order algorithm, which is a slightly simplified version[8] of the algorithm proposed by Awerbuch and Leighton [AL93], and describe the modifications needed for the second-order method. We then present our experimental results.

4.1 Description of the Algorithms

As in the single-commodity case, the algorithm proceeds in parallel rounds (or iterations). In our first-order implementation, the following operations are performed in each round.

(1) Add d_i units of commodity i to source node s_i, for $0 \leq i < k$.
(2) For each node v and each commodity i, partition the amount of commodity i that is currently in node v evenly among the $\delta(v)$ local queues of the $\delta(v)$ incident edges.
(3) In each edge e, attempt to balance the amount of each commodity between the two queues at the endpoints of the edge, subject to the capacity constraint of the edge. Several commodities may be contending for the capacity of the edge; this contention is resolved in the following way:
Let $\Delta_i(e)$ be the difference in the amount of commodity i between the two queues at the endpoints of edge e. The flow f_i for commodity i is computed from the d_i, $\Delta_i(e)$, and the edge capacity by using the algorithm described in Section 2.4.1 of [AL93], the details of which are omitted here.
(4) Remove from the network any commodity that has reached the appropriate sink.

[8] In particular, we get rid of the ϵ terms needed for the analysis in [AL93].

The second-order method can again be obtained with only a minor change in the algorithm. In particular, we compute

$$\Delta_i'(e) = \beta \cdot \Delta_i(e) + 2.0 \cdot (\beta - 1) \cdot f_i',$$

where f_i' is the amount of commodity i sent across the edge in the previous iteration. In the non-idealized version of the algorithm, where we do not allow negative amounts of commodity, we also have to adjust $\Delta_i'(e)$ if $\frac{\Delta_i'(e)}{2}$ is larger than the amount of commodity i available in the sending queue; this leads to the idealized and realistic case as with the maximum flow problem. We then apply the same algorithm as in the first-order method to resolve contention between the different commodities, but use the $\Delta_i'(e)$ in place of the $\Delta_i(e)$.

4.2 Experimental Results

We now present experimental results on the performance of the second-order method. Due to space constraints, we can only give a few selected results.

Sample Performance Results. Figure 4.2 shows the behavior of the idealized second-order method with $\beta = 1.95$ on a $5 \times 5 \times 20$ RMF graph with 5 sources and sinks selected at random from the nodes in the first and last level of the graph, respectively. The demands for the flows were chosen such that the flow is feasible, but within about 2% of the upper bound given by the maximum concurrent flow. Figure 4.2 shows the 5 flows converging to their respective demands. After about 4500 iterations, all flows have converged to within 16 digits of precision. In contrast, if we use the first-order method on this problem, then we need more than 10000 iterations to converge to within 10% of the demands.

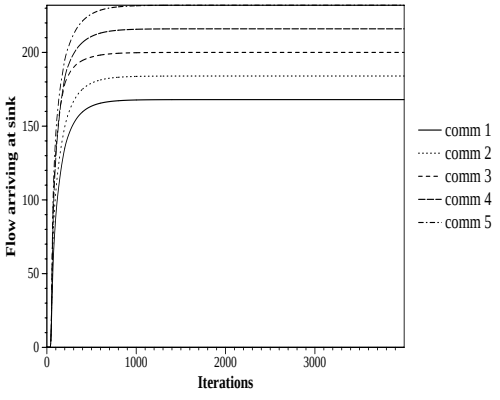


Fig. 6. Convergence of the idealized second-order method with $\beta = 1.95$ on an RMF graph with five commodities.

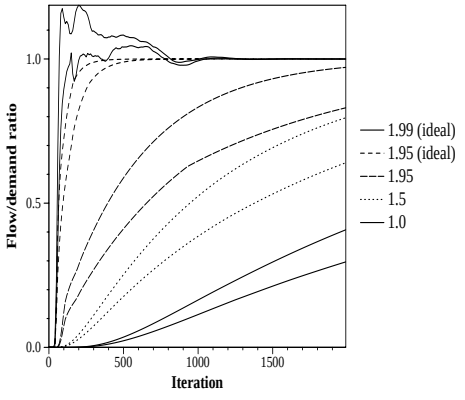


Fig. 7. Convergence of the realistic and idealized second-order methods with different values of β, on a 500 node RMF graph with 25 commodities. For each case, we plot the maximum and minimum flow/demand ratios over all commodities.

Figure 4.2 shows the behavior of the second-order method for a $5 \times 5 \times 20$ RMF graph with 25 commodities routed between the first and the last layer of the graph, with demands

chosen at random and then scaled such that they are within 1% of the maximum concurrent flow. The values measured on the y-axis are the minimum and maximum fractions z, over all commodities, such that z times the demand of a commodity arrives at its sink in a given step. Figure 4.2 shows the convergence behavior for the realistic second-order method with $\beta = 1.0$, 1.5, and 1.95, and for the idealized second-order method with $\beta = 1.95$ and 1.99. The figure shows a clear advantage of the second-order over the first-order method, and of the idealized over the realistic method.

Dependence on β. The behavior of the second-order multicommodity flow algorithms for varying values of β turned out to be similar to that of the second-order maximum flow algorithm. While for most of our input graphs the optimal value of β was between 1.95 and 1.99, there are other classes of graphs where the optimal value is significantly smaller; see Figures 4.2 and 4.2 for an example.

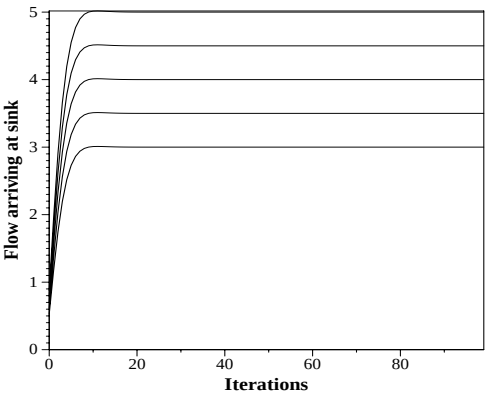


Fig. 8. Behavior of the idealized second-order method on a 5 node clique graph with 5 commodities and $\beta = 1.4$.

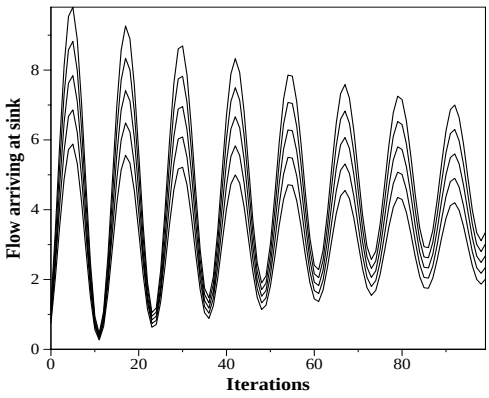


Fig. 9. Behavior of the idealized second-order method on a 5 node clique graph with 5 commodities and $\beta = 1.98$.

Running Times. In Table 1, we provide some very preliminary timing results. All timings were performed on a Sun Ultra 30 workstation with 300 Mhz UltraSPARCII processor and 256 MB of RAM, and the codes were compiled with the `-O` option using the vendor-supplied C compiler.

As input graph, we used a $5 \times 5 \times 20$ RMF graph, with 25, 50, and 100 commodities. All demands had the same value, while the capacities of the forward edges in the RMF graph were chosen at random. The sources and sinks were chosen from the first and last panels, respectively, of the graph.[9]

We give running times for four different methods: (1) the basic first-order method, as described by Awerbuch and Leighton [AL93], (2) the realistic second-order method with $\beta =$

[9] Thus, since the number of nodes in the first panel is 25, the number of "commodity groups" (see [LSS93]) in the implementation of Leong, Shor, and Stein [LSS93] is at most 25, independent of the number of commodities.

1.99, (3) the idealistic second-order method with $\beta = 1.97$, and (4) the Maximum Concurrent Flow code of Leong, Shor, and Stein [LSS93], referred to as LSS.

Algorithm	25 commodities	50 commodities	100 commodities
Leong-Shor-Stein (LSS)	519.77	456.10	501.72
First-order (Awerbuch-Leighton)	642.99	1233.32	2836.62
Realistic second-order, $\beta = 1.99$	149.01	304.64	645.16
Idealistic second-order, $\beta = 1.97$	9.54	27.70	70.41

Table 1. Running times (in seconds) of the different algorithms on a 500 node RMF graph. For LSS, we chose $\epsilon = 0.05$, while for the other codes, we terminated the runs after every commodity was within a 0.01 factor (first-order) or 0.001 factor (second-order) of its demand.

When looking at these numbers, the reader should keep the following points in mind:

(1) The code of Leong, Shor, and Stein [LSS93] solves the more general problem of maximizing the ratio of feasible flow, while our code only finds a feasible flow. However, we are not aware of any code for feasible flow that outperforms LSS. Following the suggestion in [LSS93], all our runs are performed with demands very close to the maximum feasible, by scaling the demands using the maximum edge congestion returned by LSS.
(2) The results for LSS are most likely not optimal, as we were unsure about the best setting of parameters for the code. Given the results reported in [LSS93] and the increases in CPU speed over the last few years, we would have expected slightly better numbers.
(3) We have not yet implemented a good termination condition for our code. Instead, we chose to measure the time until all flows at the sinks have converged to within a factor of at most 0.001 (second-order method) or 0.01 (first-order method) of the demands.
(4) We limit the reported numbers to RMF graphs due to differences in the graph formats used in LSS and in our code, which did not allow a direct comparison on other types of graphs.

We point out that the behavior of the LSS algorithm is fairly complex, while the performance of our second-order methods is dependent on the precise choice of β. Thus, one should be careful when trying to infer general performance trends from the few numbers provided above. However, our experiments with other graphs also showed a similar behavior. Thus, we believe that our implementation is at least competitive with the best previous codes, and may in fact significantly outperform them. We plan to perform a more thorough study in the future. We also see significant room for further improvements in the running times of our codes.

Sensitivity Analysis. An attractive feature of local algorithms is that they are, in general, robust. That is, they are expected to scale gracefully when edges appear or disappear, or traffic patterns change [AL93]. We will not try to formalize this intuition here. In Figure 4.2, we present an illustrative example of the behavior of local flow algorithms under dynamic situations, which shows how the resulting flows adapt quickly as we change the demands of commodities.

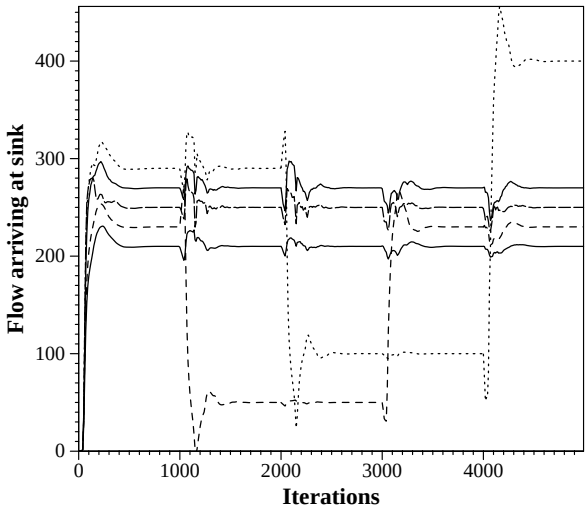


Fig. 10. Sensitivity of the algorithm to changes in demands, for the idealized method with $\beta = 1.98$ on a 500 node RMF graph with 5 commodities. We show the amounts of flow ariving at the sinks as we repeatedly change the demands, and thus the amounts of commodity injected into the network in each step.

5 Concluding Remarks

In this paper, we have proposed second-order methods for distributed, approximate maximum flow and multicommodity flow based on the first-order algorithms recently proposed by Awerbuch and Leighton [AL93, AL94]. We have presented experimental results that illustrate several interesting aspects of the behavior of these algorithms, and that provide strong evidence that the second-order methods significantly outperform their first-order counterparts.

The main open problem raised by our results is to give a formal analysis of the performance of the second-order methods for multicommodity flow, or to at least show a separation between first-order and second-order methods. We believe that this is a very challenging technical problem. Our experimental results also raise, and leave open, a number of other intriguing questions concerning the behavior of such distributed flow algorithms, and the diffusive processes underlying them. We list a few below.

Question 1. It would be very interesting to show that not only the amount of flow reaching the sinks, but in fact the entire "flow pattern" in the network converges to a stable state.[10] This was the case in all our experiments. If true, this will simplify the process of stopping the iteration in a distributed manner when the flows have converged; furthermore, it may improve the analytical bounds on the performance of the algorithm, since we do not have to average the flows over several steps as suggested in [AL93].

Question 2. For the case of the maximum flow problem, it would be interesting to show bounds that are tighter than those implied by the analysis for multicommodity flow in [AL93]. In

[10] As far as we know, this question is still open even in the first-order maximum flow case.

particular, it appears from our experiments that the convergence behavior of the maximum flow algorithms may be significantly better than $1/\epsilon$.

Question 3. Suppose the flow injected into the sources at each iteration consists of a collection of packets. Can we analyze or bound the delays of the packets, given an appropriate scheduling principle for packets at each node (such as first-in-first-out), if only for the first-order methods? This would correspond to providing certain quality-of-service guarantees to the sessions in communication networks. Such analysis was recently done for load balancing [MR98] and packet routing [AK+98] under adversarial models of traffic injection, but assuming unit edge capacities.

Question 4. As mentioned earlier, random walks can be modeled as a matrix iteration which is identical to the behavior of first-order algorithms for distributed load balancing [MGS98]. Can we design random walks that correspond to second-order algorithms? This may lead to improved bounds for mixing times of random walks. Some progress has been made recently for special graphs [S98]. Another question that arises is whether random walks can be set up to yield the first/second-order behavior in the presence of edge capacities. □

We are working on several extensions of our experimental results. In particular, we are working on an implementation of the improved first-order algorithm presented in [AL94], and on dynamic acceleration schemes for the second-order method such as those using Chebyshev polynomials with a β that varies from iteration to iteration. We are also in the process of carrying out a thorough comparison of our distributed implementations to that of the existing sequential multicommodity codes (see [LSS93] and the references therein).

6 Acknowledgments

Sincere thanks to Stephen Rutherford for experimenting with a preliminary version of the second-order method for the maximum flow problem. We also thank Fan Chung for valuable discussions, and Cliff Stein for graciously providing us with his multicommodity flow code.

References

[A94] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994.

[AAB97] B. Awerbuch, Y. Azar and Y. Bartal. Local multicast rate control with globally optimum throughput. *Manuscript*, 1997.

[AA+93] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and asynchronous networks. *Proc. of 25th ACM Symp. on Theory of Computing*, 632-641, 1993.

[AK+98] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen. Adaptive packet routing for bursty adversarial traffic. *Proc. ACM STOC*, 1998.

[AL93] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. *Proc. 34th IEEE FOCS*, 459-468, 1993.

[AL94] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multicommodity flow problem and local competitive routing in dynamic networks. *Proc. 26th ACM Symp. on Theory of Computing*, 487-496, 1994.

[AMS89] B. Awerbuch, Y. Mansour and N. Shavit. End-to-end communication with polynomial overhead. *Proc. 30th IEEE FOCS*, 358-363, 1989.

[BB+93] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst. Templates for the solution of linear systems: Building blocks for iterative methods, SIAM, Philadelphia, Penn, 1993. `http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html`.

[BG91] D. Bertsekas and R. Gallagher. *Data Networks*. Prentice-Hall Publ. 1991.

[BT89] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[C97] F. Chung. *Spectral Graph Theory*, Chapter 8, CBMS Regional conference series in Mathematics, 1997.

[C89] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 279–301, 1989.

[DMN97] R. Diekmann, S. Muthukrishnan and M. Nayakkankuppam. Engineering diffusive load balancing algorithms using experiments. *Proc. IRREGULAR 97*, LNCS Vol 1253, 111-122, 1997.

[GR97] A. Goldberg and S. Rao. Beyond the Flow Decomposition Barrier. *Proc. 38th IEEE FOCS*, 1997.

[GT88] A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 1988, 921-940.

[HY81] L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

[K97] D. Karger. Using random sampling to find maximum flows in uncapacitated undirected graphs. *Proc. 30th ACM STOC*, 1997.

[LM+91] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Proc. 24th ACM STOC*, 101-111, 1991.

[LSS93] T. Leong, P. Shor and C. Stein. Implementation of a Combinatorial Multicommodity Flow Algorithm. *First DIMACS Implementation Challenge: Network Flows and Matchings*, 1993.

[LW95] L. Lovasz and P. Winkler. Mixing of random walks and other diffusions on a graph. Technical Report, Yale University, 1995.

[MR98] S. Muthukrishnan and R. Rajaraman. An adversarial model for distributed dynamic load balancing. *Proc. 10th ACM SPAA*, 1998.

[MGS98] S. Muthukrishnan, B. Ghosh, and M. Schultz. First- and second-order diffusive methods for rapid, coarse, distributed load balancing. To appear in *Theory of Computing Systems*, 1998. Special issue on ACM SPAA 96.

[SM86] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318-334, 1990.

[S98] A. Sinclair. Personal communication, 1998.

[V89] P. Vaidya. Speeding up linear programming using fast matrix multiplication. *Proc. 30th IEEE FOCS*, 332-337, 1989.

[Var62] R. Varga. *Matrix Iterative Analysis*. Prentice–Hall, Englewood Cliffs, NJ, 1962.

7 Appendix: Experimental Setup

Implementation Details. All algorithms were implemented in C. A graphical frontend based on Tcl/Tk was used to run experiments and display the results. All input graphs were supplied in the DIMACS graph format, with some extensions to specify multiple commodities and changes in the demands over time.

Most of the execution time is spent in Steps (2) and (3) of the algorithm, which were implemented together in one single loop over the edges. Thus, the partitioning of the commodities between the queues was done during the edge balancing process, by applying an appropriate scaling factor to the flow stored in a node. This resulted in a very efficient implementation for the maximum flow case.

For the multicommodity flow case, the running time of Step (3) is dominated by the algorithm for resolving contention between different commodities in Section 2.4.1 of [AL93], which requires sorting the commodities in each edge by the values of $\Delta_i(e)/d_i^2$. While these values vary between iterations, the changes become increasingly smaller as the method converges. We exploited this property by using insertion sort and inserting the commodities in the sorted order of the previous iteration.

Input Graphs.

In our experiments described in this paper, we used three different classes of input graphs: mesh graphs, random leveled graphs, and RMF graphs. The first two types of graphs were generated using the GENGRAPH program of Anderson et al. from the University of Washington. The RMF graphs were generated with the GENRMF program of Tamas Badics. Both programs are available from the DIMACS website. Examples of these graphs are shown in Figures 7, 7, and 7.

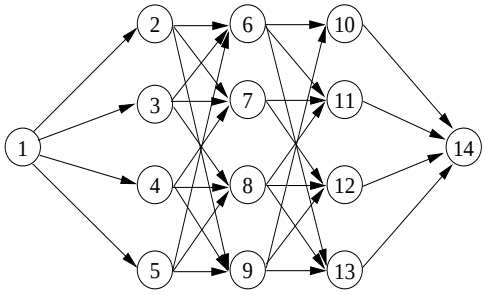


Fig. 11. Mesh graph with 3 levels and 14 nodes. All edges have randomly chosen capacity, except for edges connecting to the source or sink, which have capacity large enough such that they never constitute a bottleneck.

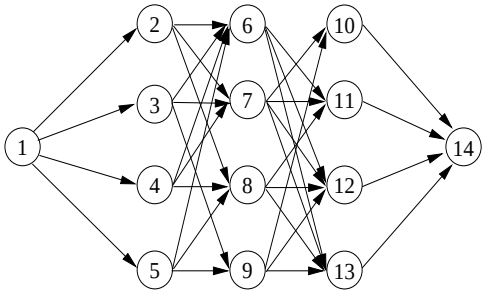


Fig. 12. Random leveled graph with 3 levels and 14 nodes. All edges have randomly chosen capacity, except for edges connecting to the source or sink, which have capacity large enough such that they never constitute a bottleneck.

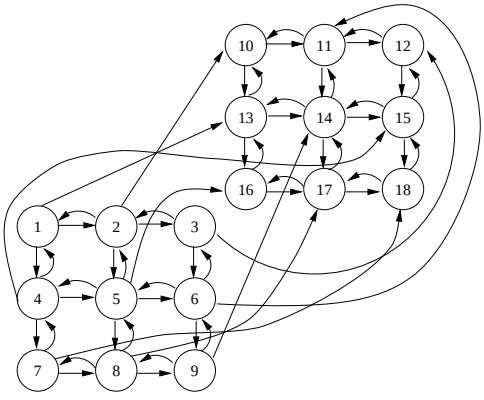


Fig. 13. A $3 \times 3 \times 2$ RMF graph. All edges between different layers have randomly chosen capacity, while edges inside a layer have capacity large enough such that they never constitute a bottleneck.

This article was processed using the LaTeX macro package with LLNCS style

Author Index