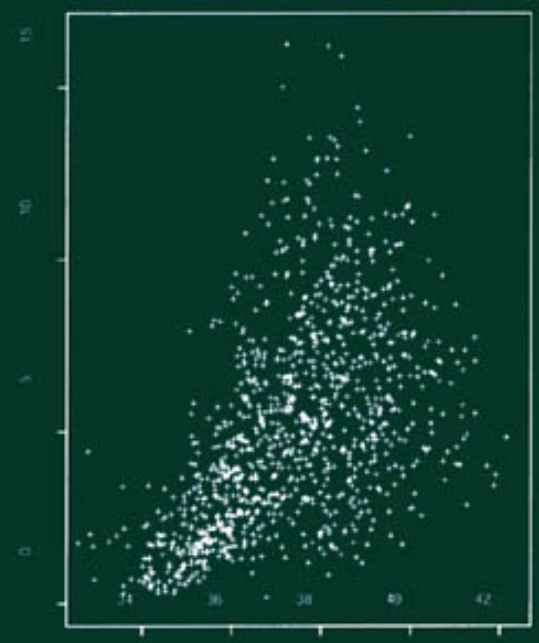
PROCEEDINGS OF THE EIGHTH WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS AND THE THIRD WORKSHOP ON ANALYTIC ALGORITHMICS AND COMBINATORICS



Edited by Rajeev Raman, Robert Sedgewick,
and Matthias F. Stallmann

Miami, FL

2006

PROCEEDINGS OF THE EIGHTH WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS AND THE THIRD WORKSHOP ON ANALYTIC ALGORITHMICS AND COMBINATORICS

SIAM PROCEEDINGS SERIES LIST

Fifth International Conference on Mathematical and Numerical Aspects of Wave Propagation (2000), Alfredo Bermúdez, Dolores Gómez, Christophe Hazard, Patrick Joly, and Jean E. Roberts, editors

Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (2001), S. Rao Kosaraju, editor

Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing (2001), Charles Koelbel and Juan Meza, editors

Computational Information Retrieval (2001), Michael Berry, editor

Collected Lectures on the Preservation of Stability under Discretization (2002), Donald Estep and Simon Tavener, editors

Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (2003), Martin Farach-Colton, editor

Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (2003), Richard E. Ladner, editor

Fast Algorithms for Structured Matrices: Theory and Applications (2003), Vadim Olshevsky, editor

Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (2004), Ian Munro, editor

Applied Mathematics Entering the 21st Century: Invited Talks from the ICIAM 2003 Congress (2004), James M. Hill and Ross Moore, editors

Proceedings of the Fourth SIAM International Conference on Data Mining (2004), Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David Skillicorn, editors

Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (2005), Adam Buchsbaum, editor

Mathematics for Industry: Challenges and Frontiers. A Process View: Practice and Theory (2005), David R. Ferguson and Thomas J. Peters, editors

Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (2006), Cliff Stein, editor

Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments and the Third Workshop on Analytic Algorithmics and Combinatorics (2006), Rajeev Raman, Robert Sedgewick, and Matthias F. Stallmann, editors

Proceedings of the Sixth SIAM International Conference on Data Mining (2006), Joydeep Ghosh, Diane Lambert, David Skillicorn, and Jaideep Srivastava, editors

PROCEEDINGS OF THE EIGHTH WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS AND THE THIRD WORKSHOP ON ANALYTIC ALGORITHMICS AND COMBINATORICS

Edited by Rajeev Raman, Robert Sedgewick, and Matthias F. Stallmann

siam

Society for Industrial and Applied Mathematics
Philadelphia

PROCEEDINGS OF THE EIGHTH WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS AND THE THIRD WORKSHOP ON ANALYTIC ALGORITHMICS AND COMBINATORICS

Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments, Miami, FL, January 21, 2006

Proceedings of the Third Workshop on Analytic Algorithmics and Combinatorics, Miami, FL, January 21, 2006

The workshop was supported by the ACM Special Interest Group on Algorithms and Computation Theory and the Society for Industrial and Applied Mathematics.

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2006922417
ISBN 0-89871-610-1

Workshop on Analytic Algorithmics and Combinatorics

ALENEX WORKSHOP PREFACE

The annual Workshop on Algorithm Engineering and Experiments (ALENEX) provides a forum for the presentation of original research in all aspects of algorithm engineering, including the implementation and experimental evaluation of algorithms and data structures. ALENEX 2006, the eighth workshop in this series, was held in Miami, Florida, on January 21, 2006. The workshop was sponsored by SIAM, the Society for Industrial and Applied Mathematics, and SIGACT, the ACM Special Interest Group on Algorithms and Computation Theory.

These proceedings contain 15 contributed papers presented at the workshop, together with the abstract of an invited lecture by David Mount, entitled "Keep Your Friends Close and Your Enemies Closer: The Art of Proximity Searching." The contributed papers were selected from a total of 46 submissions based on originality, technical contribution, and relevance. Considerable effort was devoted to the evaluation of the submissions with four reviews or more per paper. It is nonetheless expected that most of the papers in these proceedings will eventually appear in finished form in scientific journals.

The workshop took place on the same day as the Third Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2006), and papers from that workshop also appear in these proceedings. As both workshops are concerned with looking beyond the big-oh asymptotic analysis of algorithms, we hope that the ALENEX community will find the ANALCO papers to be of interest.

We would like to express our gratitude to all the people who contributed to the success of the workshop. In particular, we would like thank the authors of submitted papers, the ALENEX Program Committee members, and the external reviewers. Special thanks go to Adam Buchsbaum for answering our many questions along the way, to Andrei Voronkov for timely technical assistance with the use of the EasyChair system, and to Sara Murphy and Sarah M. Granlund for coordinating the production of these proceedings. Finally, we are indebted to Kirsten Wilden, for all of her valuable help in the many aspects of organizing this workshop.

Rajeev Raman and Matt Stallmann

ALENEX 2006 Program Committee

Ricardo Baeza-Yates, UPF, Barcelona, Spain and University of Chile, Santiago
Luciana Buriol, University of Rome "La Sapienza," Italy
Thomas Erlebach, University of Leicester, United Kingdom
Irene Finocchi, University of Rome "La Sapienza," Italy
Roberto Grossi, University of Pisa, Italy
Lutz Kettner, Max Planck Institute for Informatics, Saarbrücken, Germany
Eduardo Sany Laber, PUC, Rio de Janeiro, Brazil
Alex Lopez-Ortiz, University of Waterloo, Canada
Stefan Näher, University of Trier, Germany
Rajeev Raman (co-chair), University of Leicester, United Kingdom
Peter Sanders, University of Karlsruhe, Germany
Matt Stallmann (co-chair), North Carolina State University
Ileana Streinu, Smith College
Thomas Willhalm, Intel, Germany

ALENEX 2006 Steering Committee

Lars Arge, University of Aarhus
Roberto Battiti, University of Trento
Adam Buchsbaum, AT&T Labs—Research
Camil Demetrescu, University of Rome "La Sapienza"
Andrew V. Goldberg, Microsoft Research
Michael T. Goodrich, University of California, Irvine
Giuseppe F. Italiano, University of Rome, "Tor Vergata"
David S. Johnson, AT&T Labs—Research
Richard E. Ladner, University of Washington
Catherine C. McGeoch, Amherst College
Bernard M.E. Moret, University of New Mexico
David Mount, University of Maryland, College Park
Jack Snoeyink, University of North Carolina, Chapel Hill
Clifford Stein, Columbia University
Roberto Tamassia, Brown University

ALENEX 2006 External Reviewers

Ernst Althaus
Spyros Angelopolous
Lars Arge
Jeremy Barbay
Michael Baur
Luca Becchetti
Iwona Bialynicka-Birula
Brona Brejova
Saverio Caminiti
Timothy Chan
Valentina Ciriani
Carlos Cotta
Roman Dementiev
Camil Demetrescu
Reza Dorrigiv
Mitre Dourado
Arash Farzan
Gereon Frahling
G. Franceschini
Stefan Funke
Marco Gaertler
Emilio Di Giacomo
Robert Görke
Peter Hachenberger
Michael Hoffmann
Martin Holzer
Daniel Huson
Juha Kärkkäinen
Martin Kutz
Gad M. Landau
Marcelo Mas
Steffen Mecke
Andreas Meyer
Ulrich Meyer
Gabriel Moruz
David Mount
Carlos Oliveira
Anna Östlin Pagh
Maurizio Patrignani
Seth Pettie
Derek Phillips
Sylvain Pion
Maurizio Pizzonia
Marcus Poggi
Fabio Protti
Claude-Guy Quimper
Romeo Rizzi
Salvator Roura
Marie-France Sagot
Guido Schaefer
Dominik Schultes
Frank Schulz
Ingolf Sommer
Siang Wun Song
Renzo Sprugnoli
Eduardo Uchoa
Ugo Vaccaro

The papers in this proceedings, along with an invited talk by Michael Mitzenmacher on "New Results and Open Problems for Deletion Channels," were presented at the Third Workshop on Analytic Algorithmics and Combinatorics (ANALCO06), which was held in Miami on January 21, 2006. The aim of ANALCO is to provide a forum for the presentation of original research in the analysis of algorithms and associated combinatorial structures. The papers study properties of fundamental combinatorial structures that arise in practical computational applications (such as permutations, trees, strings, tries, and graphs) and address the precise analysis of algorithms for processing such structures, including average-case analysis; analysis of moments, extrema, and distributions; and probabilistic analysis of randomized algorithms. Some of the papers present significant new information about classic algorithms; others present analyses of new algorithms that present unique analytic challenges, or address tools and techniques for the analysis of algorithms and combinatorial structures, both mathematical and computational.

The workshop took place on the same day as the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX06); the papers from that workshop are also published in this volume. Since researchers in both fields are approaching the problem of learning detailed information about the performance of particular algorithms, we expect that interesting synergies will develop. People in the ANALCO community are encouraged to look over the ALENEX papers for problems where the analysis of algorithms might play a role; people in the ALENEX community are encouraged to look over these ANALCO papers for problems where experimentation might play a role.

ANALCO 2006 Program Committee

Jim Fill, Johns Hopkins University
Mordecai Golin, Hong Kong University of Science and Technology
Philippe Jacquet, INRIA, France
Claire Kenyon, Brown University
Colin McDiarmid, University of Oxford
Daniel Panario, Carleton University
Robert Sedgewick (chair), Princeton University
Alfredo Viola, University of Uruguay
Mark Ward, Purdue University

ANALCO 2006 Steering Committee

Philippe Flajolet, INRIA, France
Robert Sedgewick, Princeton University
Wojciech Szpankowski, Purdue University

This page intentionally left blank

Workshop on Algorithm Engineering and Experiments

This page intentionally left blank

Exact and Efficient Construction of Minkowski Sums of Convex Polyhedra with Applications*

Efi Fogel† Dan Halperin†

Abstract

We present an exact implementation of an efficient algorithm that computes Minkowski sums of convex polyhedra in $\mathbb{R}^3$. Our implementation is complete in the sense that it does not assume general position. Namely, it can handle degenerate input, and it produces exact results. We also present applications of the Minkowski-sum computation to answer collision and proximity queries about the relative placement of two convex polyhedra in $\mathbb{R}^3$. The algorithms use a dual representation of convex polyhedra, and their implementation is mainly based on the Arrangement package of CGAL, the Computational Geometry Algorithm Library. We compare our Minkowski-sum construction with the only three other methods that produce exact results we are aware of. One is a simple approach that computes the convex hull of the pairwise sums of vertices of two convex polyhedra. The second is based on Nef polyhedra embedded on the sphere, and the third is an output sensitive approach based on linear programming. Our method is significantly faster. The results of experimentation with a broad family of convex polyhedra are reported. The relevant programs, source code, data sets, and documentation are available at `http://www.cs.tau.ac.il/~efif/CD`, and a short movie [16] that describes some of the concepts portrayed in this paper can be downloaded from `http://www.cs.tau.ac.il/~efif/CD/Mink3d.avi`.

1 Introduction

Let P and Q be two closed convex polyhedra in $\mathbb{R}^d$. The Minkowski sum of P and Q is the convex polyhedron $M = P \oplus Q = \{p + q \,|\, p \in P, q \in Q\}$. A polyhedron P translated by a vector t is denoted by P^t. *Collision Detection* is a procedure that determines whether P and Q overlap. The *Separation Distance* $\pi(P,Q)$ and *the Penetration Depth* $\delta(P,Q)$ defined as

$$\pi(P,Q) = \min\{\|t\| \,|\, P^t \cap Q \neq \emptyset, t \in \mathbb{R}^d\} ,$$
$$\delta(P,Q) = \inf\{\|t\| \,|\, P^t \cap Q = \emptyset, t \in \mathbb{R}^d\}$$

are the minimum distances by which P has to be translated so that P and Q intersect or become interior disjoint respectively. The problems above can also be posed given a normalized direction d, in which case the minimum distance sought is in direction d. The *Directional Penetration Depth*, for example, is defined as

$$\pi_d(P,Q) = \inf\{a \,|\, P^{\vec{da}} \cap Q = \emptyset\} .$$

We present an exact, complete, and robust implementation of efficient algorithms to compute the Minkowski sum of two convex polyhedra, detect collision, and compute the Euclidean separation distance between, and the directional penetration-depth of, two convex polyhedra in $\mathbb{R}^3$. The algorithms use a dual representation of convex polyhedra, polytopes for short, named *Cubical Gaussian Map*. They are implemented on top of the CGAL library [1], and are mainly based on the Arrangement package of the library [17], although other parts, such as the Polyhedral-Surface package produced by L. Kettner [28], are used as well. The results obtained by this implementation are exact as long as the underlying number type supports the arithmetic operations $+$, $-$, $*$, and $/$ in unlimited precision over the rationals,[1] such as the rational number type `Gmpq` provided by GMP — Gnu's Multi Precision library [2]. The implementation is complete and robust, as it handles all degenerate cases, and guarantees exact results. We also report on the performance of our methods compared to other.

Minkowski sums are closely related to proximity queries. For example, the minimum separation

*This work has been supported in part by the IST Programmers of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE — Motion Planning in Virtual Environments), by the IST Programmers of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS — Algorithms for Complex Shapes), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

†School of Computer Science, Tel-Aviv University, 69978, Israel. {efif,danha}@post.tau.ac.il

[1] Commonly referred to as a *field* number type.

distance between two polytopes P and Q is the same as the minimum distance between the origin and the boundary of the Minkowski sum of P and the reflection of Q through the origin [12]. Computing Minkowski sums, collision detection and proximity computation comprise fundamental tasks in computational geometry [26, 32, 35]. These operations are ubiquitous in robotics, solid modeling, design automation, manufacturing, assembly planning, virtual prototyping, and many more domains; see, e.g., [10, 27, 29]. The wide spectrum of ideas expressed in the massive amount of literature published about the subject during the last three decades has inspired the development of quite a few useful solutions. For a full list of packages and overview about the subject see [32]. However, only recent advances in the implementation of computational-geometry algorithms and data structures made our exact, complete, and efficient implementation possible.

Various methods to compute the Minkowski sum of two polyhedra in $\mathbb{R}^3$ have been proposed. The goal is typically to compute the boundary of the sum and provide some representation of it. The combinatorial complexity of the Minkowski sum of two polyhedra of m and n features respectively can be as high as $\Theta(m^3n^3)$. One common approach to compute it, is to decompose each polyhedron into convex pieces, compute pairwise Minkowski sums of pieces of the two, and finally the union of the pairwise sums. Computing the exact Minkowski sum of non-convex polyhedra is naturally expensive. Therefore, researchers have focused on computing an approximation that satisfies some criteria, such as the algorithm presented by Varadhan and Manocha [36]. They guarantee a two-sides Hausdorff distance bound on the approximation, and ensure that it has the same number of connected components as the exact Minkowski sum. Computing the Minkowski sum of two convex polyhedra remains a key operation, and this is what we focus on. The combinatorial complexity of the sum can be as high as $\Theta(mn)$ when both polyhedra are convex.

Convex decomposition is not always possible, as in the presence of non-convex curved objects. In these cases other techniques must be applied, such as approximations using polynomial/rational curves in 2D [30]. Seong at al. [34] proposed an algorithm to compute Minkowski sums of a subclass of objects; that is, surfaces generated by slope-monotone closed curves. Flato and Halperin [7] presented algorithms for robust construction of planar Minkowski sums based on CGAL. While the citations in this paragraph refer to computations of Minkowski sums of non-convex polyhedra, and we concentrate on the convex cases, the latter is of particular interest, as our method makes heavy use of the same software components, in particular the CGAL Arrangement package [17], which went through a few phases of improvements since its usage in [7] and recently was redesigned and re-implemented [38].

A particular accomplishment of the *kinetic framework* in two dimensions introduced by Guibas et al. [24] was the definition of the *convolution* operation in two dimensions, a superset of the Minkowski sum operation, and its exploitation in a variety of algorithmic problems. Basch et al. extended its predecessor concepts and presented an algorithm to compute the convolution in three dimensions [8]. An output-sensitive algorithm for computing Minkowski sums of polytopes was introduced in [25]. Gritzmann and Sturmfels [22] obtained a polynomial time algorithm in the input and output sizes for computing Minkowski sums of k polytopes in $\mathbb{R}^d$ for a fixed dimension d, and Fukuda [18] provided an output sensitive polynomial algorithm for variables d and k. Ghosh [19] presented a unified algorithm for computing 2D and 3D Minkowski sums of both convex and non-convex polyhedra based on a *slope diagram* representation. Computing the Minkowski sum amounts to computing the slope diagrams of the two objects, merging them, and extracting the boundary of the Minkowski sum from the merged diagram. Bekker and Roerdink [9] provided a few variations on the same idea. The slope diagram of a 3D convex polyhedron can be represented as a 2D object, essentially reducing the problem to a lower dimension. We follow the same approach.

A simple method to compute the Minkowski sum of two polytopes is to compute the convex hull of the pairwise sum of the vertices of the two polytopes. While there are many implementations of various algorithms to compute Minkowski sums and answer proximity queries, we are unaware of the existence of complete implementations of methods to compute exact Minkowski sums other than (i) the naïve method above, (ii) a method based on Nef polyhedra embedded on the sphere [21], and (iii) an implementation of Fukuda's algorithm by Weibel [37]. Our method exhibits much better performance than the other methods in all cases, as demonstrated by the experiments listed in Table 4. Our method well handles degenerate cases that require special treatment when alternative representations are used. For example, the case of two parallel facets facing the same direction, one from each polytope, does not bear any burden on our method,

and neither does the extreme case of two polytopes with identical sets of normals.

In some cases it is sufficient to build only portions of the boundary of the Minkowski sum of two given polytopes to answer collision and proximity queries efficiently. This requires locating the corresponding features that contribute to the sought portion of the boundary. The *Cubical Gaussian Map*, a dual representation of polytopes in 3D used in our implementations, consists of six planar maps that correspond to the six faces of the unit cube — the parallel-axis cube circumscribing the unit sphere. We use the CGAL Arrangement package to maintain these data structures, and harness the ability to answer point-location queries efficiently that comes along, to locate corresponding features of two given polytopes.

The rest of this paper is organized as follows. The *Cubical Gaussian Map* dual representation of polytopes in $\mathbb{R}^3$ is described in Section 2 along with some of its properties. In Section 3 we show how 3D Minkowski sums can be computed efficiently, when the input polytopes are represented by cubical Gaussian maps. Section 4 presents an exact implementation of an efficient collision-detection algorithm under translation based on the dual representation, and provides suggestions for future directions. In Section 5 we examine the complexity of Minkowski sums, as a preparation for the following section, dedicated to experimental results. In this last section we highlight the performance of our method on various benchmarks. The software access-information along with some further design details are provided in the Appendix.

2 The Cubical Gaussian Map

The *Gaussian Map* G of a compact convex polyhedron P in Euclidean three-dimensional space $\mathbb{R}^3$ is a set-valued function from P to the unit sphere $\mathbb{S}^2$, which assigns to each point p the set of outward unit normals to support planes to P at p. Thus, the whole of a facet f of P is mapped under G to a single point — the outward unit normal to f. An edge e of P is mapped to a (geodesic) segment $G(e)$ on $\mathbb{S}^2$, whose length is easily seen to be the exterior dihedral angle at e. A vertex v of P is mapped by G to a spherical polygon $G(v)$, whose sides are the images under G of edges incident to v, and whose angles are the angles supplementary to the planar angles of the facets incident to v; that is, $G(e_1)$ and $G(e_2)$ meet at angle $\pi - \alpha$ whenever e_1 and e_2 meet at angle α. In other words, $G(v)$ is exactly the "spherical polar" of the link of v in P. (The link of a vertex is the intersection of an infinitesimal sphere centered at v with P, rescaled, so that the radius is 1.) The above implies that $G(P)$ is combinatorially dual to P, and metrically it is the unit sphere $\mathbb{S}^2$.

An alternative and practical definition follows. A direction in $\mathbb{R}^3$ can be represented by a point $u \in \mathbb{S}^2$. Let P be a polytope in $\mathbb{R}^3$, and let V denote the set of its boundary vertices. For a direction u, we define the *extremal point* in direction u to be $\lambda_V(u) = \arg\max_{p \in V} \langle u, p \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product. The decomposition of $\mathbb{S}^2$ into maximal connected regions, so that the extremal point is the same for all directions within any region forms the Gaussian map of P. For some $u \in \mathbb{S}^2$ the intersection point of the ray $\vec{ou}$ emanating from the origin with one of the hyperplanes listed below is a *central projection* of u denoted as $\hat{u_d}$. The relevant hyperplanes are $x_d = 1$, $d = 1, 2, 3$, if u lies in the positive respective hemisphere, and $x_d = -1$, $d = 1, 2, 3$ otherwise.

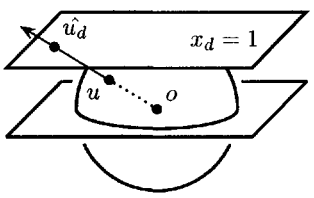


Figure 1: Central projection

Similarly, the *Cubical Gaussian Map* (CGM) C of a polytope P in $\mathbb{R}^3$ is a set-valued function from P to the six faces of the unit cube whose edges are parallel to the major axes and are of length two. A facet f of P is mapped under C to a central projection of the outward unit normal to f onto one of the cube faces. Observe that, a single edge e of P is mapped to a chain of at most three connected segments that lie in three adjacent cube-faces respectively, and a vertex v of P is mapped to at most five abutting convex dual faces that lie in five adjacent cube-faces respectively. The decomposition of the unit-cube faces into maximal connected regions, so that the extremal point is the same for all directions within any region forms the CGM of P. Likewise, the inverse CGM, denoted by C^{-1}, maps the six faces of the unit cube to the polytope boundary. Each planar face f is extended with the coordinates of its dual vertex $v = C^{-1}(f)$ among the other attributes (detailed below), resulting with a unique representation. Figure 2 shows the CGM of a tetrahedron.

While using the CGM increases the overhead of some operations sixfold, and introduces degeneracies that are not present in the case of alternative representations, it simplifies the construction and manipulation of the representation, as the partition of each cube face is a planar map of segments, a well known concept that has been intensively experimented with in recent years. We use the CGAL

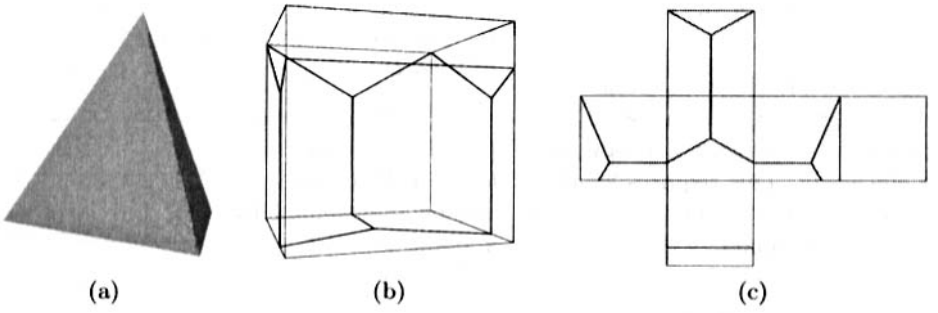


Figure 2: (a) A tetrahedron, (b) the CGM of the tetrahedron, and (c) the CGM unfolded. Thick lines indicate real edges.

`Arrangement_2`[2] data structure to maintain the planar maps. The construction of the six planar maps from the polytope features and their incident relations amounts to the insertion of segments that are pairwise disjoint in their interiors into the planar maps, an operation that can be carried out efficiently, especially when one or both endpoints are known, and we take advantage of it. The construction of the Minkowski sum, described in the next section, amounts to the computation of the overlay of six pairs of planar maps, an operation well supported by the data structure as well.

A related dual representation had been considered and discarded before the CGM representation was chosen. It uses only two planar maps that partition two parallel planes respectively instead of six, but each planar map partitions the entire plane.[3] In this representation facets that are near orthogonal to the parallel planes are mapped to points that are far away from the origin. The exact representation of such points requires coordinates with large bit-lengths, which increases significantly the time it takes to perform exact arithmetic operations on them. Moreover, facets exactly orthogonal to the parallel planes are mapped to points at infinity, and require special handling all together.

Features that are not in general position, such as two parallel facets facing the same direction, one from each polytope, or worse yet, two identical polytopes, typically require special treatment. Still, the handling of many of these problematic cases falls under the "generic" case, and becomes transparent to the CGM layer. Consider for example the case of two neighboring facets in one polytope that have parallel neighboring facets in the other. This translates to overlapping segments, one from each CGM of the two polytopes,[4] that appear during the Minkowski sum computation. The algorithm that computes it is oblivious to this condition, as the underlying `Arrangement_2` data structure is perfectly capable of handling overlapping segments. However, as mentioned above, other degeneracies do emerge, and are handled successfully. One example is a facet f mapped to a point that lies on an edge of the unit cube, or even worse, coincides with one of the eight corners of the cube. Figure 8(a,b,c) depicts an extreme degenerate case of an octahedron oriented in such a way that its eight facet-normals are mapped to the eight vertices of the unit cube respectively.

The dual representation is extended further, in order to handle all these degeneracies and perform all the necessary operations as efficiently as possible. Each planar map is initialized with four edges and four vertices that define the unit square — the parallel-axis square circumscribing the unit circle. During construction, some of these edges or portions of them along with some of these vertices may turn into real elements of the CGM. The introduction of these artificial elements not only expedites the traversals below, but is also necessary for handling degenerate cases, such as an empty cube face that appears in the representation of the tetrahedron and depicted in Figure 2(c). The global data consists of the six planar maps and 24 references to the vertices that coincide with the unit-cube corners.

The exact mapping from a facet normal in the 3D coordinate-system to a pair that consists of a planar map and a planar point in the 2D coordinate-

[2] CGAL prescribes the suffix _2 (resp. _3) for all data structures of planar objects (resp. 3D objects) as a convention.

[3] Each planar map that corresponds to one of the six unit-cube faces in the CGM representation also partitions the entire plane, but only the $[-1,-1] \times [1,1]$ square is relevant. The unbounded face, which comprises all the rest, is irrelevant.

[4] Other conditions translate to overlapping segments as well.

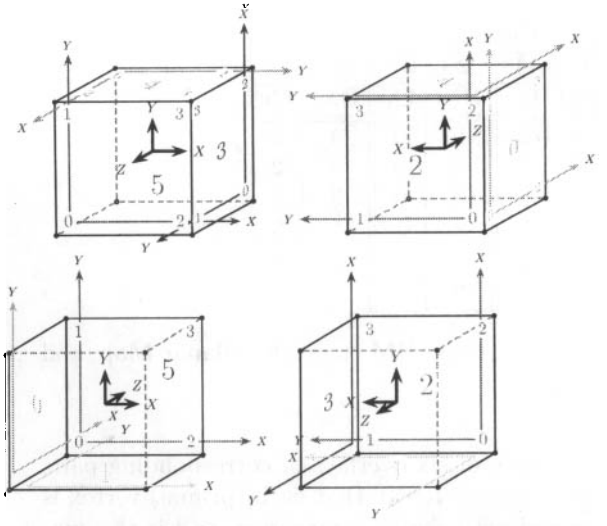
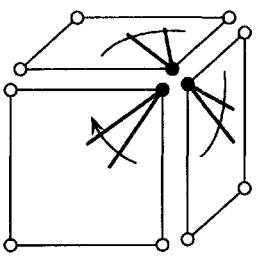


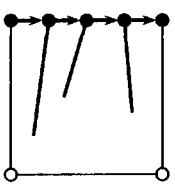
Figure 3: The data structure. Large numbers indicate plane ids. Small numbers indicate corner ids. X and Y axes in different 2D coordinate systems are rendered in different colors.

system is defined precisely through the indexing and ordering system, illustrated in Figure 3. Now before your eyes cross permanently, we advise you to keep reading the next few lines, as they reveal the meaning of some of the enigmatic numbers that appear in the figure. The six planar maps are given unique ids from 0 through 5. Ids 0, 1, and 2 are associated with planes contained in negative half spaces, and ids 3, 4, and 5 are associated with planes contained in positive half spaces. The major axes in the 2D Cartesian coordinate-system of each planar map are determined by the 3D coordinate-system. The four corner vertices of each planar map are also given unique ids from 0 through 3 in lexicographic order in their respective 2D coordinate-system, see Table 1 columns titled **Underlying Plane** and **2D Axes**.

A *doubly-connected edge list* (DCEL) data structure is used by the `Arrangement_2` data structure to maintain the incidence relations on its features. Each topological edge of the subdivision is represented by two halfedges with opposite orientation, and each halfedge is associated with the face to its left. Each feature type of the `Arrangement_2` data structure is extended to hold additional attributes. Some of the attributes are introduced only in order to expedite the computation of certain operations, but most of them are necessary to handle degenerate cases such as a planar vertex lying on the unit-square boundary. Each planar-map vertex v is extended with (i) the coefficients of the plane containing the polygonal facet $C^{-1}(v)$, (ii) the location of the vertex — an enumeration indicating whether the vertex coincides with a cube corner, or lies on a cube edge, or contained in a cube face, (iii) a boolean flag indicating whether it is non-artificial (there exists a facet that maps to it), and (iv) a pointer to a vertex of a planar map associated with an adjacent cube-face that represents the same central projection for vertices that coincide with a cube corner or lie on a cube edge. Each planar-map halfedge e is extended with a boolean flag indicating whether it is non-artificial (there exists a polytope edge that maps to it). Each planar-map face f is extended with the polytope vertex that maps to it $v = C^{-1}(f)$.



Each vertex that coincides with a unit-cube corner or lies on a unit-cube edge contains a pointer to a vertex of a planar map associated with an adjacent cube face that represents the same central projection. Vertices that lie on a unit-cube edge (but do not coincide with unit-cube corners) come in pairs. Two vertices that form such a pair lie on the unit-square boundary of planar maps associated with adjacent cube faces, and they point to each other. Vertices that coincide with unit-cube corners come in triplets and form cyclic chains ordered clockwise around the respective vertices. The specific connections are listed in Table 1. As a convention, edges incident to a vertex are ordered clockwise around the vertex, and edges that form the boundary of a face are ordered counterclockwise. The `Polyhedron_3` and `Arrangement_2` data structures for example, both use a DCEL data structure that follows the convention above. We provide a fast clockwise traversal of the faces incident to any given vertex v. Clockwise traversals around internal vertices are immediately available by the DCEL. Clockwise traversals around boundary vertices are enabled by the cyclic chains above. This traversal is used to calculate the normal to the (primary) polytope-facet $f = C^{-1}(v)$ and to render the facet. Fortunately, rendering systems are capable of handling a sequence of vertices that define a polygon in clockwise order as well, an order opposite to the conventional ordering above.



The data structure also supports a fast traversal over the planar-map halfedges that form each one of the four unit-square edges. This traversal is used during construction to quickly locate a vertex that coincides with a cube corner or lies on a cube edge. It is also used to update the cyclic chains of pointers mentioned above; see Section 3.

Underlying Plane		2D Axes		Corner							
				0 (0,0)		**1** (0,1)		**2** (1,0)		**3** (1,1)	
Id	Eq	X	Y	PM	Cr	PM	Cr	PM	Cr	PM	Cr
0	$x=-1$	Z	Y	1	0	2	2	5	0	4	2
1	$y=-1$	X	Z	2	0	0	2	3	0	5	2
2	$z=-1$	Y	X	0	0	1	2	4	0	3	2
3	$x=1$	Y	Z	2	1	1	3	4	1	5	3
4	$y=1$	Z	X	0	1	2	3	5	1	3	3
5	$z=1$	X	Y	1	1	0	3	3	1	4	3

Table 1: The coordinate systems, and the cyclic chains of corner vertices. **PM** stands for **Planar Map**, and **Cr** stands for **Corner**.

We maintain a flag that indicates whether a planar vertex coincides with a cube corner, a cube edge, or a cube face. At first glance this looks redundant. After all, this information could be derived by comparing the x and y coordinates to -1 and $+1$. However, it has a good reason as explained next. Using exact number-types often leads to representations of the geometric objects with large bit-lengths. Even though we use various techniques to prevent the length from growing exponentially [17], we cannot avoid the length from growing at all. Even the computation of a single intersection requires a few multiplications and additions. Cached information computed once and stored at the features of the planar map avoids unnecessary processing of potentially-long representations.

3 Exact Minkowski Sums

The overlay of two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$ is a planar subdivision $\mathcal{S}$ such that there is a face f in $\mathcal{S}$ if and only if there are faces f_1 and f_2 in $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively such that f is a maximal connected subset of $f_1 \cap f_2$. The overlay of the Gaussian maps of two polytopes P and Q identifies all the pairs of features of P and Q respectively that have common supporting planes, as they occupy the same space on the unit sphere, thus, identifying all the pairwise features that contribute to the boundary of the Minkowski sum of P and Q. A facet of the Minkowski sum is either a facet f of Q translated by a vertex of P supported by a plane parallel to f, or vice versa, or it is a facet parallel to two parallel planes supporting an edge of P and an edge of Q respectively. A vertex of the Minkowski sum is the sum of two vertices of P and Q respectively supported by parallel planes. A similar argument holds for the cubical Gaussian map with the unit cube replacing the unit sphere. More precisely, a single map that subdivides the unit sphere is replaced by six planar maps, and the computation of a single overlay is replaced by the computation of six overlays of corresponding pairs of planar maps. Recall that each (primal) vertex is associated with a planar-map face, and is the sum of two vertices associated with the two overlapping faces of the two CGM's of the two input polytopes respectively.

Each planar map in a CGM is a convex subdivision. Finke and Hinrichs [15] describe how to compute the overlay of such special subdivisions optimally in linear time. However, a preliminary investigation shows that a large constant governs the linear complexity, which renders this choice less attractive. Instead, we resort to a sweep-line based algorithm that exhibits good practical performance. In particular we use the overlay operation supported by the `Arrangement_2` package. It requires the provision of a complementary component that is responsible for updating the attributes of the DCEL features of the resulting six planar maps.

The overlay operates on two instances of `Arrangement_2`. In the description below v_1, e_1, and f_1 denote a vertex, a halfedge, and a face of the first operand respectively, and v_2, e_2, and f_2 denote the same feature types of the second operand respectively. When the overlay operation progresses, new vertices, halfedges, and faces of the resulting planar map are created based on features of the two operands. There are ten cases described below that must be handled. When a new feature is created its attributes are updated. The updates performed in all cases except for case (1) are simple and require constant time. We omit their details due to lack of space.

1. A new vertex v is induced by coinciding vertices v_1 and v_2.
   The location of the vertex v is set to be the same as the location of the vertex v_1 (the locations of v_2 and v_1 must be identical). The induced vertex is not artificial if (i) at least one of the vertices v_1 or v_2 is not artificial, or

(ii) the vertex lies on a cube edge or coincides with a cube corner, and both vertices v_1 and v_2 have non-artificial incident halfedges that do not overlap.

2. A new vertex is induced by a vertex v_1 that lies on an edge e_2.

3. A new vertex is induced by a vertex v_2 that lies on an edge e_1.

4. A new vertex is induced by a vertex v_1 that is contained in a face f_2.

5. A new vertex is induced by a vertex v_2 that is contained in a face f_1.

6. A new vertex is induced by the intersection of two edges e_1 and e_2.

7. A new edge is induced by the overlap of two edges e_1 and e_2.

8. A new edge is induced by the an edge e_1 that is contained in a face f_2.

9. A new edge is induced by the an edge e_2 that is contained in a face f_1.

10. A new face is induced by the overlap of two faces f_1 and f_2.

After the six map overlays are computed, some maintenance operations must be performed to obtain a valid CGM representation. As mentioned above, the global data consists of the six planar maps and 24 references to vertices that coincide with the unit-cube corners. For each planar map we traverse its vertices, obtain the four vertices that coincide with the unit-cube corners, and initialize the global data. We also update the cyclic chains of pointers to vertices that represent identical central projections. To this end, we exploit the fast traversal over the halfedges that coincide with the unit-cube edges mentioned in Section 2.

The complexity of a single overlay operation is $O(k \log n)$, where n is the total number of vertices in the input planar maps, and k is the number of vertices in the resulting planar map. The total number of vertices in all the six planar maps in a CGM that represents a polytope P is of the same order as the number of facets in the primary polytope P. Thus, the complexity of the entire overlay operation is $O(F \log(F_1 + F_2))$, where F_1 and F_2 are the number of facets in the input polytopes respectively, and F is the number of facets in the Minkowski sum.

4 Exact Collision Detection

Computing the separation distance between two polytopes with m and n features respectively can be done in $O(\log m \log n)$ time, after an investment of at most linear time in preprocessing [13]. Many practical algorithms that exploit spatial and temporal coherence between successive queries have been developed, some of which became classic, such as the GJK algorithm [20] and its improvement [11], and the LC algorithm [31] and its optimized variations [14, 23, 33]. Several general-purpose software libraries that offer practical solutions are available today, such as the SOLID library [4] based on the improved GJK algorithm, the SWIFT library [5] based on an advanced version of the LC algorithm, the QuickCD library [3], and more. For an extensive review of methods and libraries see the recent survey [32].

Given two polytopes P and Q, detecting collision between them and computing their relative placement can be conveniently done in the configuration space, where their Minkowski sum $M = P \oplus (-Q)$ resides. These problems can be solved in many ways, and not all require the explicit representation of the Minkowski sum M. However, having it available is attractive, especially when the polytopes are restricted to translations only, as the combinatorial structure of the Minkowski sum M is invariant to translations of P or Q. The algorithms described below are based on the following well known observations:

$$P^u \cap Q^w \neq \emptyset \Leftrightarrow w - u \in M = P \oplus (-Q) \ ,$$
$$\delta(P^u, Q^w) = \min\{\|t\| \mid (w - u + t) \in M, t \in \mathbb{R}^3\} \ ,$$
$$\pi_d(P^u, Q^w) = \inf\{\alpha \mid (w - u + \vec{d}\alpha) \notin M\} \ .$$

Given two polytopes P and Q in the CGM representation, we reflect Q through the origin to obtain $-Q$, compute the Minkowski sum M, and retain it in the CGM representation. Then, each time P or Q or both translate by two vectors u and w in $\mathbb{R}^3$ respectively, we apply a procedure that determines whether the query point $s = w - u$ is inside, on the boundary of, or outside M. In addition to an enumeration of one of the three conditions above, the procedure returns a witness of the respective relative placement in form of a pair that consists of a vertex $v = C(f)$ — a mapping of a facet f of M embedded in a unit cube face, and the planar map $\mathcal{P}$ containing v. This information is used as a hint in consecutive invocations. The facet f is the one stabbed by the ray r emanating from an internal point $c \in M$, and going through s. The internal point could be the average of all vertices

of M computed once and retained along M, or just the midpoint of two vertices that have supporting planes with opposite normals easily extracted from the CGM. Once f is obtained, determining whether P^u and Q^w collide is trivial, according to the first formula (of the three) above.

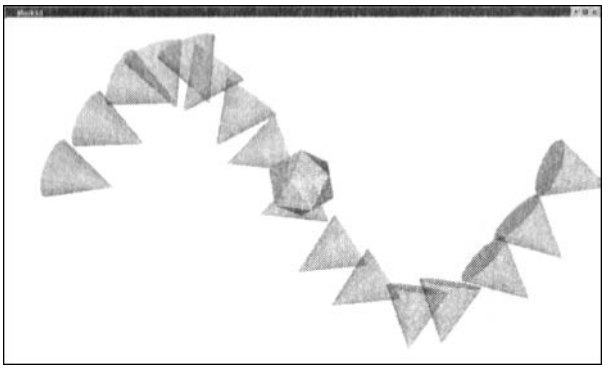


Figure 4: Simulation of motion.

The procedure applies a local walk on the cube faces. It starts with some vertex v_s, and then performs a loop moving from the current vertex to a neighboring vertex, until it reaches the final vertex, perhaps jumping from a planar map associated with one cube-face to a different one associated with an adjacent cube-face. The first time the procedure is invoked, v_s is chosen to be a vertex that lies on the central projection of the normal directed in the same direction as the ray r. In consecutive calls, v_s is chosen to be the final vertex of the previous call exploiting spatial and temporal coherence. Figure 4 is a snapshot of a simulation program that detects collision between a static obstacle and a moving robot, and draws the obstacle and the trail of the robot. The Minkowski sum is recomputed only when the robot is rotated, which occurs every other frame. The program is able to identify the case where the robot grazes the obstacle, but does not penetrate it. The computation takes just a fraction of a second on a Pentium PC clocked at 1.7 GHz. Similar procedures that compute the directional penetration-depth and minimum distance are available as well.

We intend to develop a complete integrated framework that answers proximity queries about the relative placement of polytopes that undergo rigid motions including *rotation* using the cubical Gaussian-map in the follow-up project. Some of the methods we foresee compute only those portions of the Minkowski sum that are absolutely necessary, making our approach even more competitive. Briefly, instead of computing the Minkowski sum of P and $-Q$, we walk simultaneously on the two respective CGM's, producing one feature of the Minkowski sum at each step of the walk. Such a strategy could be adapted to the case of rotation by rotating the trajectory of the walk, keeping the CGM of $-Q$ intact, instead of rotating the CGM itself.

5 Minkowski Sum Complexity

The number of facets of the Minkowski sum of two polytopes in $\mathbb{R}^3$ with m and n facets respectively is bounded from above by $\Theta(mn)$. Before reporting on our experiments, we give an example of a Minkowski sum with complexity $\Omega(mn)$. The example depicted in Figure 6 gives rise to a number as high as $\frac{(m+1)(n+1)}{2}$ when mn is odd, and $\frac{(m+1)(n+1)+1}{2}$ when mn is even. The example consists of two identical squashed dioctagonal pyramids, each containing n faces ($n = 17$ in Figure 6), but one is rotated about the Z axis approximately[5] 90° compared to the other. This is perhaps best seen when the spherical Gaussian map is examined, see Figure 5. The pyramid must be squashed to ensure that the spherical edges that are the mappings of the pyramid-base edges are sufficiently long. (A similar configuration, where the polytopes are non-squashed is depicted in Figure 8(d,e,f,g,h,i). A careful counting reveals that the number of vertices in the dual representation excluding the artificial vertices reaches $\frac{(m+1)(n+1)}{2} = 162$, which is the number of facets of the Minkowski sum. We are still investigating the problem of bounding the *exact* maximum complexity of the Minkowski sum of two polytopes. Our preliminary results imply that the coefficient of the mn component is higher than in the example illustrated here.

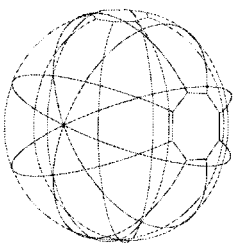
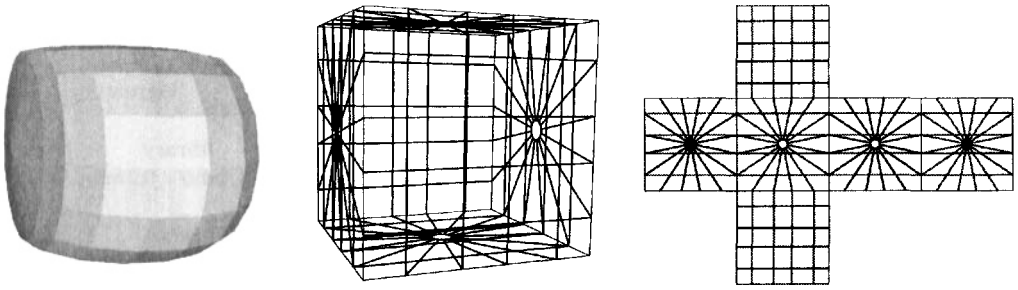


Figure 5: $m = n = 9$

Not every pair of polytopes yields a Minkowski sum proportional to mn. As a matter of fact, it can be as low as n in the extremely-degenerate case of two identical polytopes variant under scaling. Even if no degeneracies exist, the complexity can be proportional to only $m + n$, as in the case of two geodesic spheres[6] level $l = 2$ slightly rotated

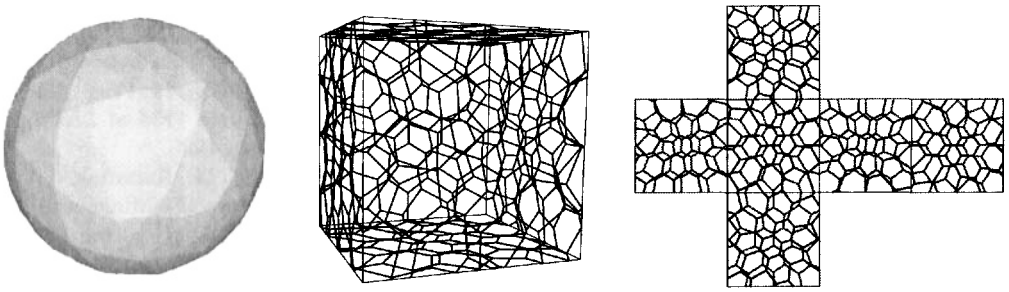
[5] The results of all rotations are approximate, as we have not yet dealt with exact rotation. One of our immediate future goals is the handling of exact rotations.

[6] An icosahedron, every triangle of which is divided into $(l+1)^2$ triangles, whose vertices are elevated to the circumscribing sphere.



(a) (b) (c)

Figure 6: (a) The Minkowski sum of two approximately orthogonal squashed dioctagonal pyramids, (b) the CGM, and (c) the CGM unfolded, where red lines are graphs of edges that originate from one polytope and blue lines are graphs of edges that originate from the other.



(a) (b) (c)

Figure 7: (a) The Minkowski sum of two geodesic spheres level 2 slightly rotated with respect to each other, (b) the CGM of the Minkowski sum, and (c) the CGM unfolded.

with respect to each other, depicted in Figure 7. Naturally, an algorithm that accounts for all pairs of vertices, one from each polytope, is rendered inferior compared to an output sensitive algorithm such as ours in such cases, as we demonstrate in the next section.

6 Experimental Results

We have created a large database of convex polyhedra in polygonal representation stored in an extended VRML format [6]. In particular, each model is provided in a representation that consists of the array of boundary vertices and the set of boundary polygons, where each polygon is described by an array of indices into the vertex array. (Identical to the *IndexedFaceSet* representation.) Constructing the CGM of a model given in this representation is done indirectly. First, the CGAL `Polyhedron_3` data structure that represents the model is constructed [28]. This data structure consists of vertices, edges, and facets and incidence relations on them. Then, the CGM is constructed using the accessible incidence relations provided by `Polyhedron_3`. Once the construction of the CGM is complete, the intermediate representation is discarded.

Table 2 shows the number of vertices, halfedges, and faces of the six planar maps that comprise the CGM of our squashed dioctagonal pyramid. The number of faces of each planar map include the unbounded face. Table 3 shows the number of features in the primal and dual representations of a small subset of our polytopes collection. The number of planar features is the total number of features of the six planar maps.

Planar map	**V**	**HE**	**F**
0, $(x = -1)$	12	32	6
1, $(y = -1)$	36	104	18
2, $(z = -1)$	12	32	6
3, $(x = 1)$	12	32	6
4, $(y = 1)$	21	72	17
5, $(z = 1)$	12	32	6
Total	105	304	59

Table 2: The number of features of the six planar maps of the CGM of the dioctagonal pyramid object.

As mentioned above, the Minkowski sum of two polytopes is the convex hull of the pairwise sum of the vertices of the two polytopes. We have implemented this straightforward method using the CGAL *convex_hull_3* function, which uses the `Polyhedron_3` data structure to represent the resulting polytope, and used it to verify the correctness of our method. We compared the time it took to compute exact Minkowski sums using these two

methods, a third method implemented by Hachenberger based on Nef polyhedra embedded on the sphere [21], and a fourth method implemented by Weibel [37], based on an output sensitive algorithm designed by Fukuda [18].

The Nef-based method is not specialized for Minkowski sums. It can compute the overlay of two arbitrary Nef polyhedra embedded on the sphere, which can have open and closed boundaries, facets with holes, and lower dimensional features. The overlay is computed by two separate hemisphere-sweeps.

Fukuda's algorithm relies on linear programming. Its complexity is $O(\delta LP(3,\delta)V)$, where $\delta = \delta_1 + \delta_2$ is the sum of the maximal degrees of vertices, δ_1 and δ_2, in the two input polytopes respectively, V is the number of vertices of the resulting Minkowski sum, and $LP(d,m)$ is the time required to solve a linear programming in d variables and m inequalities. Note, that Fukuda's algorithm is more general, as it can be used to compute the Minkowski sum of polytopes in an arbitrary dimension d, and as far as we know, it has not been optimized specifically for $d = 3$.

The results listed in Table 4, produced by experiments conducted on a Pentium PC clocked at 1.7 GHz, show that our method is much more efficient in all cases, and more than three hundred times faster than the convex-hull method in one case. The last column of the table indicates the ratio $\frac{F_1 F_2}{F}$, where F_1 and F_2 are the number of facets of the input polytopes respectively, and F is the number of facets of the Minkowski sum. As this ratio increases, the relative performance of the output-sensitive algorithms compared to the convex-hull method, increases as expected.

Object Type	Primal			Dual		
	V	E	F	V	HE	F
Tetrahedron	4	6	4	38	94	21
Octahedron	6	12	6	24	48	12
Icosahedron	12	30	20	72	192	36
DP	17	32	17	105	304	59
PH	92	150	60	196	684	158
TI	120	180	62	230	840	202
GS4	252	750	500	708	2124	366

Table 3: Complexity of the primal and dual representations. DP — Dioctagonal Pyramid, PH — Pentagonal Hexecontahedron, TI — Truncated Icosidodecahedron, GS4 — Geodesic Sphere level 4.

References

[1] The CGAL project homepage. `http://www.cgal.org/`.

[2] The GNU MP bignum library. `http://www.swox.com/gmp/`.

[3] The QUICKCD library homepage. `http://www.ams.sunysb.edu/~jklosow/quickcd/QuickCD.html`.

[4] The SOLID library homepage. `http://www.win.tue.nl/cs/tt/gino/solid/`.

[5] The SWIFT++ library homepage. `http://gamma.cs.unc.edu/SWIFT++/`.

[6] The web3D homepage. `http://www.web3d.org/`.

[7] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geom. Theory Appl.*, 21:39–61, 2002.

[8] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proc. 4th Annu. Euro. Sympos. Alg.*, volume 1136 of *LNCS*, pages 302–319. Springer-Verlag, 1996.

[9] H. Bekker and J. B. T. M. Roerdink. An efficient algorithm to calculate the Minkowski sum of convex 3d polyhedra. In *Proc. of the Int. Conf. on Comput. Sci.-Part I*, pages 619–628. Springer-Verlag, 2001.

[10] J.-D. Boissonnat, E. de Lange, and M. Teillaud. Minkowski operations for satellite antenna layout. In *Proc. 13th Annu. ACM Sympos. on Comput. Geom.*, pages 67–76, 1997.

[11] S. A. Cameron. Enhancing GJK: computing minimum and penetration distances between convex polyhedra. In *Proc. of IEEE Int. Conf. Robot. Auto.*, pages 3112–3117, Apr. 1997.

[12] S. A. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. In *Proc. of IEEE Int. Conf. Robot. Auto.*, pages 591–596, 1986.

[13] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In *Proc. 17th Int. Colloq. Auto. Lang. Prog.*, volume 443 of *LNCS*, pages 400–413. Springer-Verlag, 1990.

[14] S. A. Ehmann and M. C. Lin. Accelerated proximity queries between convex polyhedra by multi-level Voronoi marching. In *Proc. IEEE/RST Int.. Conf. Intell. Robot. Sys.*, pages 2101–2106, 2000.

[15] U. Finke and K. H. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. 11th Annu. ACM Sympos. on Comput. Geom.*, pages 119–126. ACM Press, 1995.

[16] E. Fogel and D. Halperin. Video: Exact minkowski sums of convex polyhedra. In *Proc. ACM Sympos. on Comput. Geom.*, pages 382–383, 2005.

[17] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving CGAL's arrangements. In *Proc. 12th Annu. Euro. Sympos. Alg.*, volume 3221 of *LNCS*, pages 664–676. Springer-Verlag, 2004.

[18] K. Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. *Journal*

Object 1	Object 2	Minkowski Sum						CGM	NGM	Fuk	CH	$\frac{F_1F_2}{F}$
		Primal			Dual							
		V	E	F	V	HE	F					
Icos	Icos	12	30	20	72	192	36	**0.01**	0.36	0.04	0.1	20
DP	ODP	131	261	132	242	832	186	**0.02**	1.08	0.35	0.31	2.2
PH	TI	248	586	340	514	1670	333	**0.05**	2.94	1.55	3.85	10.9
GS4	RGS4	1048	2568	1531	1906	6288	1250	**0.31**	14.33	5.80	107.35	163.3

Table 4: Time consumption (in seconds) of the Minkowski-sum computation. Icos — Icosahedron, DP — Dioctagonal Pyramid, ODP — Orthogonal Dioctagonal Pyramid, PH — Pentagonal Hexecontahedron, TI — Truncated Icosidodecahedron, GS4 — Geodesic Sphere level 4, RGS4 — Rotated Geodesic Sphere level 4, **CH** — the Convex Hull method, **CGM** — the Cubical Gaussian Map based method, **NGM** — the Nef based method, **Fuk**— Fukuda's Linear Programming based algorithm, $\frac{F_1F_2}{F}$ — the ratio between the product of the number of input facets and the number of output facets.

of Symbolic Computation, 38(4):1261–1272, 2004.

[19] P. K. Ghosh. A unified computational framework for Minkowski operations. *Comp. Graph.*, 17(4):357–378, 1993.

[20] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects. *Proc. of IEEE Int. J. Robot. Auto.*, 4(2):193–203, 1988.

[21] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3d selective nef complexes: Data structure, algorithms, and implementation. In *Proc. 11th Annu. Euro. Sympos. Alg.*, volume 2832 of *LNCS*, pages 174–186. Springer-Verlag, 2003.

[22] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: Computational complexity and applications to Gröbner bases. *SIAM J. Disc. Math*, 6(2):246–269, 1993.

[23] L. Guibas, D. Hsu, and L. Zhang. H-walk: Hierarchical distance computation for moving convex bodies. In *ACM Sympos. on Comput. Geom.*, pages 265–273, 1999.

[24] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[25] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Disc. Comp. Geom.*, 2:175–193, 1987.

[26] D. Halperin, L. Kavraki, and J.-C. Latombe. Robotics. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 48, pages 1065–1093. CRC, 2004.

[27] A. Kaul and J. Rossignac. Solid-interpolation deformations: Construction and animation of PIPs. In *Eurographics'91*, pages 493–505, 1991.

[28] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13:65–90, 1999.

[29] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

[30] I. K. Lee, M. S. Kim, and G. Elber. Polynomial/rational approximation of Minkowski sum boundary curves. *Graphical Models and Image Processing*, 60(2):136–165, 1998.

[31] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proc. of IEEE Int. Conf. Robot. Auto.*, pages 1008–1014, 1991.

[32] M. C. Lin and D. Manocha. Collision and proximity queries. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 35, pages 787–807. CRC, 2004.

[33] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208, 1998.

[34] J.-K. Seong, M.-S. Kim, and K. Sugihara. The Minkowski sum of two simple surfaces generated by slope-monotone closed curves. In *Geom. Model. Proc.: Theory and Appl.*, pages 33–42. IEEE Comput. Sci., 2002.

[35] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 47, pages 1037–1064. CRC, 2004.

[36] G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. In *Proc. Comput. Graph, and Appl., 12th Pacific Conf. on (PG'04)*, pages 392–401. IEEE Comput. Sci., 2004.

[37] C. Weibel. Minkowski sums. `http://roso.epfl.ch/cw/poly/public.php`.

[38] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to cgal's arrangement package. In *Library-Centric Software Design Workshop (LCSD'05)*, 2005. Available online at `http://lcsd05.cs.tamu.edu/#program`.

A Software Components, Libraries and Packages

We have developed the `Cubical_gaussian_map_3` data structure, which can be used to construct and maintain cubical Gaussian-maps, and compute Minkowski sums of pairs of polytopes represented by the `Cubical_ gaussian_map_3` data structure.[7] We have developed two interactive 3D applications; a player of 3D objects stored in an extended VRML format, and an interactive application that detects collisions and answers proximity queries for polytopes that undergo translation and rotation. The format was extended with two geometry nodes: the `ExactPolyhedron` node represents models using the CGAL `Polyhedron_3` data structure, and the `CubicalGaussianMap` node represents models using the `Cubical_gaussian_map_3` data structure. Inability to provide exact coordinates impairs the entire process. To this end, the format was further extended with a node called `ExactCoordinate` that represents exact coordinates. It has a field member called `ratPoint` that specifies triple rational-coordinates, where each coordinates is specified by two integers, the numerator and the denominator of a coordinate in $\mathbb{R}^3$. Both applications are linked with (i) CGAL, (ii) a library that provides the exact rational number-type, and (iii) internal libraries that construct and maintain 3D scene-graphs, written in C++, and built on top of `OpenGL`. We experimented with two different exact number types: one provided by LEDA 4.4.1, namely `leda_rat`, and one by GMP 4.1.2, namely `Gmpq`. The former does not normalize the rational numbers automatically. Therefore, we had to initiate normalization operations to contain their bit-length growth. We chose to do it right after the central projections of the facet-normals are calculated, and before the chains of segments, which are the mapping of facet-edges, are inserted into the planar maps. Our experience shows that indiscriminate normalization considerably slows down the planar-map construction, and the choice of number type may have a drastic impact on the performance of the code overall. The internal code was divided into three libraries; (i) `SGAL` — The main 3D scene-graph library, (ii) `SCGAL` — Extensions that depend on CGAL, and (iii) `SGLUT` — Miscellaneous windowing and main-event loop utilities that depend on the `glut` library.

The 3D programs, source code, data sets, and documentation can be downloaded from `http://www.cs.tau.ac.il/~efif/CD/3d`.

[7] We intend to introduce a package by the same name, `Cubical_gaussian_map_3`, to a prospective future-release of CGAL.

Unfortunately, compiling and executing the programs require an unpublished fairly recent version of CGAL. Thus, until the upcoming public release of CGAL (version 3.2) becomes available, the programs are useful only for those who have access to the internal release. Precompiled executables, compiled with `g++` 3.3.2 on Linux Debian, are available as well.

B Additional Models

See next page.

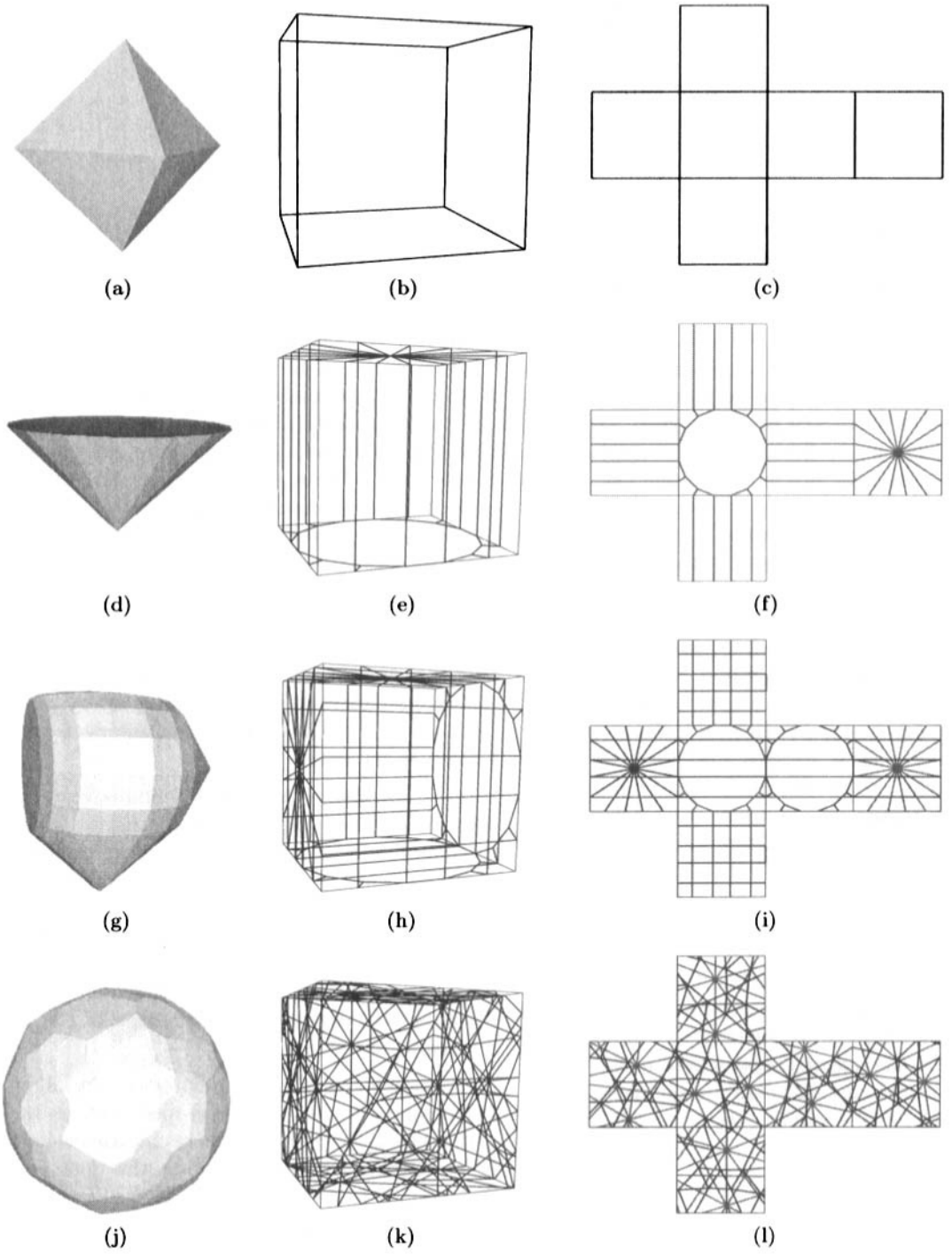


Figure 8: (a) An octahedron, (d) a dioctagonal pyramid, (g) the Minkowski sum of two approximately orthogonal dioctagonal pyramids, (j) the Minkowski sum of a Pentagonal Hexecontahedron and a Truncated Icosidodecahedron, (b,e,h,k) the CGM of the respective polytope, and (c,f,i,l) the CGM unfolded.

An Experimental Study of Point Location in General Planar Arrangements*

Idit Haran† Dan Halperin†

Abstract

We study the performance in practice of various point-location algorithms implemented in CGAL, including a newly devised *Landmarks* algorithm. Among the other algorithms studied are: a naïve approach, a "walk along a line" strategy and a trapezoidal-decomposition based search structure. The current implementation addresses general arrangements of arbitrary planar curves, including arrangements of non-linear segments (e.g., conic arcs) and allows for degenerate input (for example, more than two curves intersecting in a single point, or overlapping curves). All calculations use exact number types and thus result in the correct point location. In our Landmarks algorithm (a.k.a. Jump & Walk), special points, "landmarks", are chosen in a preprocessing stage, their place in the arrangement is found, and they are inserted into a data-structure that enables efficient nearest-neighbor search. Given a query point, the nearest landmark is located and then the algorithm "walks" from the landmark to the query point. We report on extensive experiments with arrangements composed of line segments or conic arcs. The results indicate that the Landmarks approach is the most efficient when the overall cost of a query is taken into account, combining both preprocessing and query time. The simplicity of the algorithm enables an almost straightforward implementation and rather easy maintenance. The generic programming implementation allows versatility both in the selected type of landmarks, and in the choice of the nearest-neighbor search structure. The end result is a highly effective point-location algorithm for most practical purposes.

*Work reported in this paper has been supported in part by the IST Programme of the EU as a Shared-corst RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes), by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

†School of Computer Science, Tel-Aviv University, 69978, Israel. {haranidi,danha}@post.tau.ac.il

1 Introduction

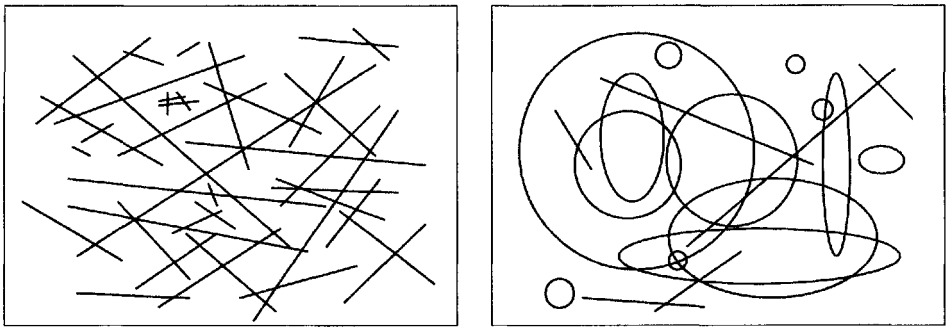
Given a set $\mathcal{C}$ of n planar curves, the *arrangement* $\mathcal{A}(\mathcal{C})$ is the subdivision of the plane induced by the curves in $\mathcal{C}$ into maximal connected cells. The cells can be 0-dimensional (*vertices*), 1-dimensional (*edges*) or 2-dimensional (*faces*). The *planar map* of $\mathcal{A}(\mathcal{C})$ is the embedding of the arrangement as a planar graph, such that each arrangement vertex corresponds to a planar point, and each edge corresponds to a planar subcurve of one of the curves in $\mathcal{C}$. Arrangements and planar maps are ubiquitous in computational geometry, and have numerous applications (see, e.g., [5, 18].) Figure 1 shows two arrangements of different types of curves, one induced by line segments and the other by conic arcs.[1] The planar point-location problem is one of the most fundamental problems applied to arrangements: Preprocess an arrangement into a data structure, so that given any query point q, the cell of the arrangement containing q can be efficiently retrieved.

In case the arrangement remains unmodified once it is constructed, it may be useful to invest considerable amount of time in preprocessing in order to achieve real-time performance of point-location queries. On the other hand, if the arrangement is dynamic, and new curves are inserted to it (or removed from it), an auxiliary point-location data-structure that can be efficiently updated must be employed, perhaps at the expense of the query answering speed.

A naïve approach to point location might be traversing over all the edges and vertices in the arrangement, and finding the geometric entity that is exactly on, or directly above, the query point. The time it takes to perform the query using this approach is proportional to the number of edges n, both in the average and worst-case scenarios.

A more economical approach [25] is to draw a vertical line through every vertex of the arrangement to obtain vertical *slabs* in which point location is almost one-dimensional. Then, two binary searches suffice to answer a query: one on x-coordinates for the slab containing q, and one on

[1] A *conic curve* is an algebraic planar curve of degree 2. A *conic arc* is a bounded segment of a conic curve.



(a) (b)

Figure 1: Random arrangements of line segments (a) and of conic arcs (b).

edges that cross the slab. Query time is $O(\log n)$, but the space may be quadratic. In order to reduce the space to linear storage space, Sarnack and Tarjan [26] used Persistent Search Trees. Edahiro et al. [15] used these ideas and developed a point-location algorithm that is based on a grid. The plane is divided into cells of equal size called buckets using horizontal and vertical partition lines. In each bucket the local point location is performed using the slabs algorithm described above.

Another approach aiming at worst-case query time $O(\log n)$ was proposed by Kirkpatrick [19], using a data structure of size $O(n)$. Mulmuley [23] and Seidel [27] proposed an alternative method that uses the vertical decomposition of the arrangement into pseudo-trapezoidal cells, and constructs a search Directed Acyclic Graph (DAG) over these simple cells. We refer to the latter algorithm, which is based on Randomized Incremental Construction, as the *RIC* algorithm.

Point location in Delaunay triangulations was extensively studied: Early works on point location in triangulations can be found in [21] and [22]. Devillers et al. [12] proposed a *Walk along a line* algorithm, which does not require the generation of additional data structures, and offers $O(\sqrt{n})$ query time on the average ($O(n)$ in the worst case). The walk may begin at an arbitrary vertex of the triangulation, and advance towards the query point. Due to the simplicity of the structures (triangles), the walk consists of low-cost operations. Devillers later proposed a walk strategy based on a Delaunay hierarchy [10], which uses a hierarchy of triangles, and performs a hierarchical search from the highest level in the hierarchy to the lowest. At each level of the hierarchical search, a walk is performed to find the triangle in the next lower level, until the triangle in the lowest level is found. Other algorithms that were developed only for Delaunay triangulations, often referred to as *Jump & Walk* algorithms, were proposed by Devroye et al. [13, 14].

Arya et al. [6] devised point location algorithms aiming at good average (rather than worst-case) query time. The efficiency of these algorithms is measured with respect to the entropy of the arrangement.

The algorithms presented in this paper are part of the arrangement package in CGAL, the Computational Geometry Algorithms Library [1]. CGAL is the product of a collaborative effort of several sites in Europe and Israel, aiming to provide a generic and robust, yet efficient, implementation of widely used geometric data structures and algorithms. It is a software library written in C++ according to the generic programming paradigm. Robustness of the algorithms is achieved by both handling all degenerate cases, and by using exact number types. CGAL's arrangement package was the first generic software implementation, designed for constructing arrangements of arbitrary planar curves and supporting operations and queries on such arrangements [16, 17]. The arrangement class-template is parameterized by a traits class that encapsulates the geometry of the family of curves it handles. Robustness is guaranteed, as long as the traits classes use exact number types for the computations they perform. Among the number-type libraries that are used are GMP- Gnu's multi-precision library [4], for rational numbers, and CORE [2] and LEDA [3] for algebraic numbers.

Point location constitutes a significant part of the arrangement package, as it is a basic query applied to arrangements during their construction. Various point-location algorithms (also referred to as point-location strategies) have been implemented as part of the CGAL's arrangement package: The *Naïve* strategy traverses all vertices and edges, and locates the nearest edge or vertex that is situated exactly on, or immediately above, the query point. The *Walk* algorithm traces (in reverse order) a vertical ray r emanating from

the query point to infinity; it traverses the *zone*[2] of r in the arrangement. This vertical walk is simpler than a walk along an arbitrary direction (that will be explained in details below, as part of the Landmarks algorithm), as it requires simpler predicates ("above/below" comparisons). Simple predicates are desirable in exact computing especially with non-linear curves. Both the Naïve and the Walk strategies maintain no data structures, beyond the basic representation of the arrangement, and do not require any preprocessing stage. Another point-location strategy implemented in CGAL for line-segments arrangement is a triangulation algorithm, which consists of a preprocessing stage where the arrangement is refined using a Constrained Delaunay Triangulation. In the triangulation, point location is implemented using a triangulation hierarchy [10]. The algorithm uses the triangulation package of CGAL [9]. The RIC point-location algorithm described above was also implemented in CGAL [16].

The motivation behind the development of the new, Landmarks, algorithm, was to address both issues of preprocessing complexity and query time, something that none of the existing strategies do well. The Naïve and the Walk algorithms have, in general, bad query time, which precludes their use in large arrangements. The RIC algorithm answers queries very fast, but it uses relatively large amount of memory and requires a complex preprocessing stage. In the case of dynamic arrangements, where curves are constantly being inserted to or removed from, this is a major drawback. Moreover, in real-life applications the curves are typically inserted to the arrangement in non-random order. This reduces the performance of the RIC algorithm, as it relies on random order of insertion, unless special procedures are followed [11].

In the *Landmarks* algorithm, special points, which we call "landmarks", are chosen in a preprocessing stage, their place in the arrangement is found, and they are inserted into a hierarchical data-structure enabling fast nearest-neighbor search. Given a query point, the nearest landmark is located, and a "walk" strategy is applied, starting at the landmark and advancing towards the query point. This walk part differs from other walk algorithms that were tailored for triangulations (especially Delaunay triangulations), as it is geared towards general arrangements that may contain faces of arbitrary topology, with unbounded complexity, and a variety of degeneracies. It also differs from the Walk algorithm implemented in CGAL as the walk direction is arbitrary, rather than vertical. Tests that were carried out using the Landmarks algorithm, reported in Section 3 indicate that the Landmarks algorithm has relatively short preprocessing stage, and it answers queries fast.

The rest of this paper is organized as follows: Section 2 describes the Landmarks algorithm in details. Section 3 presents a thorough point-location benchmark conducted on arrangements of varying size and density, composed of either line segments or conic arcs, with an emphasis on studying the behavior of the Landmarks algorithm. Concluding remarks are given in Section 4.

2 Point Location with Landmarks

The basic idea behind the *Landmarks* algorithm is to choose and locate points (landmarks) within the arrangement, and store them in a data structure that supports nearest-neighbor search. During query time, the landmark closest to the query point is found using the nearest-neighbor search and a short "walk along a line" is performed from the landmark towards the query point. The key motivation behind the Landmarks algorithm is to reduce the number of costly algebraic predicates involved in the Walk or the RIC algorithms at the expense of increased number of the relatively inexpensive coordinate comparisons (in nearest-neighbor search.)

The algorithm relies on three independent components, each of which can be optimized or replaced by a different component (of the same functionality):

1. Choosing the landmarks that faithfully represent the arrangement, and locating them in the arrangement.
2. Constructing a data structure that supports nearest-neighbor search (such as a kd-trees [8]), and using this structure to find the nearest landmark given a query point.
3. Applying a "walk along a line" procedure, moving from the landmark towards the query point.

The following sections elaborate on these components.

2.1 Choosing the Landmarks. When choosing the landmarks we aim to minimize the expected length of the "walk" inside the arrangement towards a query point. The search for a good set of landmarks has two aspects:

1. Choosing the number of landmarks.
2. Choosing the distribution of the landmarks throughout the arrangement.

[2] The *zone* of a curve is the collection of all the cells in the arrangement that the curve intersects.

It is clear that as the number of landmarks grows, the walk stage becomes faster. However, this results in longer preprocessing time, and larger memory usage. Indeed, in certain cases the nearest-neighbor search consumes a significant portion of the overall query time (when "overshooting" with the number of landmarks - see Section 3.3 below).

What constitutes a good set of landmarks depends on the specific structure of the arrangement at hand. In order to assess the quality of the landmarks, we defined a metric representing the complexity of the walk stage: The *arrangement distance* (AD) between two points is the number of faces crossed by the straight line segment that connects these points. If two points reside in the same face of the arrangement, the arrangement distance is defined to be zero. The arrangement distance may differ substantially from the Euclidean distance, as two points, which are spatially close, can be separated in an arrangement by many small faces.

The landmarks may be chosen with respect to the (0,1 or 2-dimensional) cells of the arrangement. One can use the vertices of the arrangement as landmarks, points along the edges (e.g., the edges midpoints), or interior points in the faces. In order to choose representative points inside the faces, it may be useful to preprocess the arrangement faces, which are possibly non-convex, for example using vertical decomposition or triangulation.[3] Such preprocessing will result in simple faces (pseudo trapezoids and triangles respectively) for which interior points can be easily determined. Landmarks may also be chosen independently of the arrangement geometry. One option is to spread the landmarks randomly inside a rectangle bounding the arrangement. Another is to use a uniform grid, or to use other structured point sets, such as Halton sequences or Hammersley points [20, 24]. Each choice has its advantages and disadvantages and improved performance may be achieved using combinations of different types of landmark choices.

In the current implementation the landmark type is given as a template parameter, called *generator*, to the Landmarks algorithm, and can be easily replaced. This generator is responsible for creating the sets of landmark points and updating them if necessary. The following types of landmark generators were implemented: *LM(vert)* – all the arrangement vertices are used as landmarks, *LM(mide)* – midpoints of all the arrangement edges are chosen, *LM(rand)* – random points are selected, *LM(grid)* – the landmarks are chosen on a uniform grid, and *LM(halton)* – Halton sequence points are used. In the LM(rand), LM(grid) and LM(halton) the number of landmarks is given as a parameter to the generator, and is set to be the number of vertices by default. The benefit of using vertices or edge's midpoints as landmarks, is that their location in the arrangement is known, and they represent the arrangement well (dense areas contain more vertices). The drawback is that walking from a vertex requires a preparatory step in which we examine all incident faces around the vertex to decide on the startup face. Walking from the midpoints of the edges also requires a small preparatory step to choose between the two faces incident to the edge.

For random landmarks, we use uniform samples inside the arrangement bounding-rectangle. After choosing the points, we have to locate them in the arrangement. To this end, we use the newly implemented batched point location in CGAL, which uses the sweep algorithm for constructing the arrangement, while adding the landmark points as special events in the sweep. When reaching such a special event during the sweep, we search the y-structure to find the edge that is just above the point. Similar preprocessing is conducted on the uniform grid, when the grid points are used as landmarks, and also on the Halton points. When random points, grid points or Halton points are used, it is in most cases clear in which face a landmark is located (as opposed to the case of vertices or edge midpoints). Thus, a preparatory step is scarcely required at the beginning of the walk stage.

2.2 Nearest Neighbor Search Structure.

Following the choice and location of the landmarks, we have to store them in a data structure that supports nearest-neighbor queries. The search structure should allow for fast preprocessing and query. A search structure that supports approximate nearest-neighbor search can also be suitable, since the landmarks are used as starting points for the walk, and the final accurate result of the point location is computed in the walk stage.

Exact results can be obtained by constructing a Voronoi diagram of the landmarks. However, locating the query point in the Voronoi diagram is again a point-location problem. Thus, using Voronoi diagrams as our search structure takes us back to the problem we are trying to solve. Instead, we look for a simple data structure that will answer nearest-neighbor queries quickly, even if only approximately.

The nearest-neighbor search structure is a template parameter to the Landmarks algorithm. This modularity enables us to test several nearest-neighbor structures. One implementation uses

[3] Triangulation is relevant only in case of arrangements of line segments.

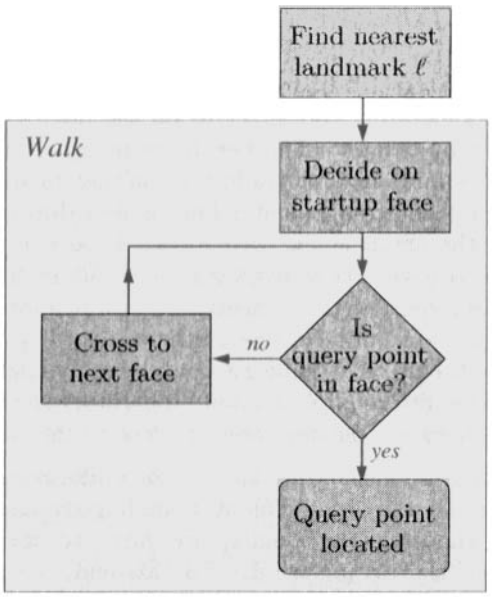


Figure 2: The query algorithm diagram.

the CGAL's spatial searching package, which is based on kd-trees. The input points provided to this structure (landmarks, query points) are approximations of the original points (rounded to double), which leads to extremely fast search. Again, we emphasize that the end result is always exact.

Another implementation uses the ANN package [7], which supports data structures and algorithms for both exact and approximate nearest neighbor searching. The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies. Few tests that were made using this package show similar results to those using CGAL's kd-tree.

In the special case of LM(grid), no search structure is needed, and the closest landmark can be found in $O(1)$ time.

2.3 Walking from the Landmark to the Query Point. The "walk" algorithm developed as part of this work is geared towards general arrangements, which may contain faces of arbitrary topology and of unbounded (not necessarily constant) complexity. This is different from previous Walk algorithms that were tailored for triangulations, especially the Delaunay triangulation.

The "walk" stage is summarized in the diagram in Figure 2. First, the startup face must be determined. As explained in the previous section, certain types of landmarks (vertices, edges) are not associated with a single startup face. A virtual line segment s is then drawn from the landmark (whose location in the arrangement is known) to the query point q. Based on the direction of s, the startup face f out of the faces incident to the landmark is associated with the landmark.

Then, a test whether the query point q lies inside f is applied. This operation requires a pass over all the edges on the face boundary. This pass is quick, since we only count the number of f's edges above q. We first check if the point is in the edge's x-range. If it is, we check the location of q with respect to the edge, and count the edge only if the point is below it. If the number of edges above q is odd, then q is found to be inside f, and the query is terminated.

Otherwise, we continue our walk along the virtual segment s toward q. In order to walk along s, we need to find the first edge e on f's boundary that intersects s. Since the arrangement's data-structure holds for each edge the information of both faces incident to this edge, all we need is to cross to the face on the other side of e.

Figure 3 shows two examples of walking from a vertex type landmark towards the query point.

As explained above, crossing to the next face requires finding the edge e on the boundary of f that intersects s. Actually, there is no need to find the exact intersection point between e and s, as this may be an expensive operation. Instead, it is sufficient to perform a simpler operation. The idea is to consider the x-range that contains both the curves s and e, and compare the vertical order of these curves on the left and right boundaries of this range. If the vertical order changes, it implies that the curves intersect; see, e.g., Figure 4(a). In case several edges on f's boundary intersects s, we cross using the first edge that was found, and mark this edge as used. This edge will not be crossed again during this walk, which assures that the walk process ends.

Care should be exercised when dealing with special cases, such as when s and e share a common endpoint, as shown in Figure 4(b). In this case we need to compare the curves slightly to the right of this endpoint (the endpoint of e is the landmark ℓ). Another case that is relevant to non-linear curves, shown in Figure 4(c), is when e and s intersect an even number of times (two in this case), and thus no crossing is needed.

3 Experimental Results

3.1 The Benchmark. In this section we describe the benchmark we used to study the behavior of various point-location algorithms and specifically the newly proposed Landmarks algorithm.

The benchmark was conducted using four

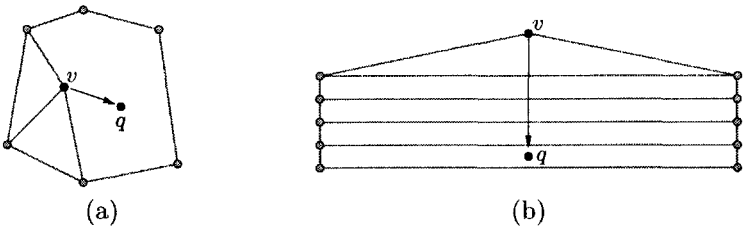


Figure 3: Walking from a landmark located on a vertex v to a query point q: no crossing is needed (a), multiple crossings are required during the walk (b).

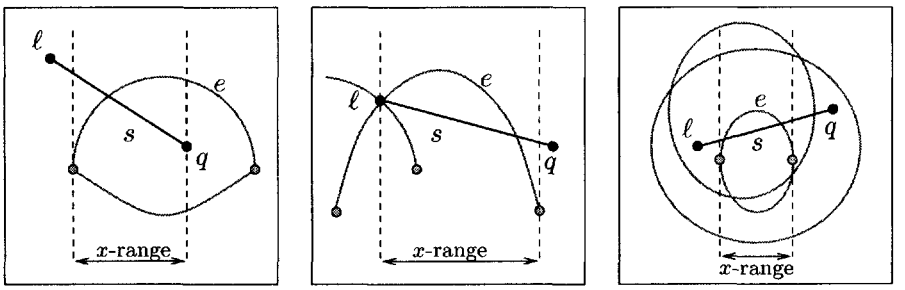


Figure 4: Walk algorithms, crossing to the next face. In all cases the vertical order of the curves is compared on the left and right boundaries of the marked x-range. (a) s and e swap their y-order, therefore we should use e to cross to the next face. (b) s and e share a common left endpoint, but e is above s immediately to the right of this point. (c) The y-order does not change, as s and e have an even number (two) of intersections.

types of arrangements: denotes as *random segments*, *random conics*, *robotics*, and *Norway*. Each arrangement in the first type was constructed by line segments that were generated by connecting pairs of points whose coordinates x, y are each chosen uniformly at random in the range $[0, 1000]$. We generated arrangements of various sizes, up to arrangements consisting of more than 1,350,000 edges.

The second type of arrangements, random conics, are composed of 20% random line segments, 40% circles and 40% canonical ellipses. The circles centers were chosen uniformly at random in the range $[0, 1000] \times [0, 1000]$ and their radii were chosen uniformly at random in the range $[0, 250]$. The ellipses were chosen in a similar manner, with their axes lengths chosen independently in the range $[0, 250]$.

The third type, *robotics*, is a line-segment arrangement that was constructed by computing the Minkowski sum[4] of a star-shaped robot and a set of obstacles. This arrangement consists of 25,533 edges. The last type, *Norway*, is also a line-segment arrangement, that was constructed by computing the Minkowski sum of the border of Norway and a polygon. The resulting arrangement consist of 42,786 edges.

For each arrangement we selected 1000 random query points to be located in the arrangement. For the comparison between the various algorithms, we measured the preprocessing time, the average query time, and the memory usage of the algorithms. All algorithms were run on the same set of arrangements and same sets of query points.

Several point-location algorithms were studied. We tested the different variants of the Landmarks algorithm: LM(vert), LM(rand), LM(grid), LM(halton) and LM(mide). The number of landmarks used in the LM(vert), LM(rand), LM(grid), LM(halton) is equal to the number of vertices of the arrangement. The number of landmarks used in the LM(mide) is equal to the number of edges of the arrangement. All Landmarks algorithms, besides LM(grid), use CGAL's kd-tree as their nearest neighbor search structure.

We also used the benchmark to study the Naïve algorithm, the Walk (from infinity) algorithm, the RIC algorithm, and the Triangulation algorithm (only for line segments). The LM(mide) was also not implemented on conic-arc arrangements, since finding the midpoint of a conic arc connecting two vertices of the arrangement, which

[4]The *Minkowski sum* of sets A and B is the set $\{a+b \mid a \in A, b \in B\}$

may have been constructed by intersection of two conic curves, is not a trivial operation, and the middle point may possibly be of high algebraic degree.

As stated above, all calculations use exact number types, and result in the exact point location. The benchmark was conducted on a single 2.4GHz PC with 1GB of RAM, running under LINUX.

3.2 Results. Table 1 shows the average query time associated with point location in arrangements of varying types and sizes using the different point-location algorithms. The number of edges mentioned in these tables is the number of undirected edges of the arrangement. In the CGAL implementation each edge is represented by two halfedges with opposite orientations.

Table 2 shows the preprocessing time for the same arrangements and same algorithms as in Table 1. The actual preprocessing consist of two parts: Construction of the arrangement (common to all algorithms), and construction of auxiliary data structures needed for the point location, which are algorithm specific. As mentioned above, the Naïve and the Walk strategies do not require any specific preprocessing stage besides constructing the arrangement, and therefore do not appear in the table.

Table 3 shows the memory usage of the point-location strategies of the random line-segment arrangements from Tables 1 and 2.

The information presented in these tables shows that, unsurprisingly, the Naïve and the Walk strategies, although they do not require any preprocessing stage and any memory besides the basic arrangement representation, result with the longest query time in most cases, especially in case of large arrangements.

The Triangulation algorithm has the worst preprocessing time, which is mainly due to the time for subdividing the faces of the arrangement using Constrained Delaunay Triangulation (CDT); this implies that resorting to CDT is probably not the way to go for point location in arrangements of segments. The query time of this algorithm is quite fast, since it uses the Dalaunay hierarchy, although it is not as fast as the RIC or the Landmarks algorithm.

The RIC algorithm results with fast query time, but it consumes the largest amount of memory, and its preprocessing stage is very slow.

All the Landmarks algorithms have rather fast preprocessing time and fast query time. The LM(vert) has by far the fastest preprocessing time, since the location of the landmarks is known, and there is no need to locate them in the preprocessing stage. The LM(grid) has the fastest query time for large-size arrangements induced by both line-segments and conic-arcs. The size of the memory used by LM(vert) algorithm is the smallest of all algorithms.

The other two variants of landmarks that were examined but are not reported in the tables are (i) the LM(halton), which has similar results to that of the LM(rand), and (ii) the LM(mide) which yields similar results to those of the LM(vert), although since it uses more landmarks, it has a little longer query and preprocess, which makes it less efficient for these types of arrangement.

Figure 5 presents the combined cost of a query (amortizing also the preprocessing time over all queries) on the last random-segments arrangement shown in the tables, which consists of more than 1,350,000 edges. The x-axis indicates the number of queries m. The y-axis indicates the average amortized cost-per-query, $cost(m)$, which is calculated in the following manner:

$$cost(m) = \frac{\text{preprocessing time}}{m} + \text{average query time} \quad (3.1)$$

We can see that when m is small, the cost is a function of the preprocessing time of the algorithm. Clearly, when $m \to \infty$, $cost(m)$ becomes the query time. For the Naïve and the Walk algorithms that do not require preprocessing, $cost(m) = \textit{query time} = \textit{constant}$. Looking at the lower envelope of these graphs we can see that for $m < 100$ the Walk algorithm is the most efficient. For $100 < m < 100,000$ the LM(vert) algorithm is the most efficient, and for $m > 100,000$ the LM(grid) algorithm gives the best performance. As we can see, for each number of queries, there exists a Landmarks algorithm, which is better than the RIC algorithm.

3.3 Analysis. As mentioned in Sections 2 and 3, there are various parameters that effect the performance of the Landmarks algorithm, such as the number of landmarks, their distribution over the arrangement, and the structure used for the nearest-neighbor search. We checked the effect of varying the number of landmarks on the performance of the algorithm, using several random arrangements.

Table 4 shows typical results, obtained for the last random-segments arrangement of our benchmark. The landmarks used for these tests were random points sampled uniformly in the bounding rectangle of the arrangement. As expected, increasing the number of random landmarks in-

Arrang. Type	#Edges	Naïve	Walk	RIC	Triang.	LM (vert)	LM (rand)	LM (grid)
	2112	2.2	0.8	0.06	0.86	0.16	0.13	0.13
random	37046	36.7	3.6	0.09	1.17	0.20	0.16	0.15
segments	235446	241.4	9.7	0.12	1.96	0.38	0.35	0.18
	955866	1636.1	15.0	0.23	1.83	1.27	1.45	0.18
	1366364	2443.6	18.0	0.27	2.10	1.80	2.06	0.19
random	1001	1.4	0.2	0.05	N/A	0.31	0.08	0.07
conics	3418	5.6	0.5	0.07	N/A	0.32	0.07	0.06
	13743	21.7	1.1	0.09	N/A	0.38	0.07	0.07
robotics	25533	37.6	1.3	0.08	0.39	0.12	0.11	0.07
Norway	42786	65.7	0.9	0.10	0.52	0.15	0.15	0.08

Table 1: Average time (in milliseconds) for one point-location query.

Arrang. Type	#Edges	Construct. Arrangement	RIC	Triang.	LM (vert)	LM (rand)	LM (grid)
	2112	0.07	0.5	11.2	0.01	0.12	0.13
random	37046	1.26	29.7	360.2	0.05	2.97	2.95
segments	235446	8.90	115.0	3360.1	0.33	24.23	22.25
	955866	60.51	616.5	21172.2	2.25	141.88	100.79
	1366364	97.67	1302.3	33949.1	3.37	212.79	148.61
random	1001	8.24	2.20	N/A	0.01	0.17	0.22
conics	3418	29.22	6.09	N/A	0.03	0.61	0.80
	13743	127.04	28.26	N/A	0.13	2.72	3.57
robotics	25533	2.63	8.29	34.67	0.06	1.69	0.35
Norway	42786	5.28	20.06	70.33	0.10	3.23	2.37

Table 2: Preprocessing time (in seconds).

creases the preprocessing time of the algorithm. However, the query time decreases only until a certain minimum around 100,000 landmarks, and it is much larger for 1,000,000 landmarks. The last column in the table shows the percentage of queries, where the chosen startup landmark was in the same face as the query point. As expected, this number increases with the number of landmarks.

An in-depth analysis of the duration of the Landmarks algorithm reveals that the major time-consuming operations vary with the size of the arrangement (and consequently, the number of landmarks used), and with the Landmarks type used. Figure 6 shows the duration percentages of the various steps of the query operation, in the LM(vert) and LM(grid) algorithms. As can be seen in the LM(vert) diagram, the nearest-neighbor search part increases when more landmarks are present, and becomes the most time-consuming part in large arrangements. In the LM(grid) algorithm, this step is negligible.

A significant step that is common to all Landmarks algorithms, checking whether the query point is in the current face, also consumes a significant part of the query time. This part is the major step of the LM(grid) algorithm.

Additional operation shown in the LM(vert) diagram is finding the startup face in a specified direction. This step is relevant only in the LM(vert) and the LM(mide) algorithms. The last operation, crossing to the next face, is relatively short in LM(vert), as in most cases (more than 90%) the query point is found to be inside the startup face. This step is a little longer in LM(grid) than in LM(vert), since only about 70% of the query points are found to be in the same face as the landmark point.

4 Conclusions

We propose a new Landmarks algorithm for exact point location in general planar arrangements, and have integrated an implementation of our algorithm into CGAL. We use generic programming, which allows for the adjustment and extension for any type of planar arrangements. We tested the performance of the algorithm on arrangements constructed of different types of curves, i.e., line segments and conic arcs, and compared it with other point-location algorithms.

The main observation of our experiments is that the Landmarks algorithm is the best strategy considering the cost per query, which takes

Arrang. Type	#Edges	Arrangement Size	RIC	Triang.	LM (vert)	LM (rand)	LM (grid)
random segments	2112	0.8	1.3	0.3	0.2	0.5	0.5
	37046	9.5	21.5	7.7	2.6	8.1	6.8
	235446	57.3	136.5	46.4	17.0	51.9	44.4
	955866	231.3	555.0	206.1	55.8	208.5	178.1
	1366364	333.8	793.2	268.9	86.8	307.0	258.9

Table 3: Memory usage (in MBytes) by the point location data structure.

Number of Landmarks	Preprocessing Time [sec]	Query Time [msec]	% Queries with AD=0
100	61.7	4.93	3.4
1000	59.0	1.60	7.6
10000	60.8	0.58	19.2
100000	74.3	0.48	42.3
1000000	207.2	3.02	71.9

Table 4: LM(rand) algorithm performance for a fixed arrangement and a varying number of random landmarks.

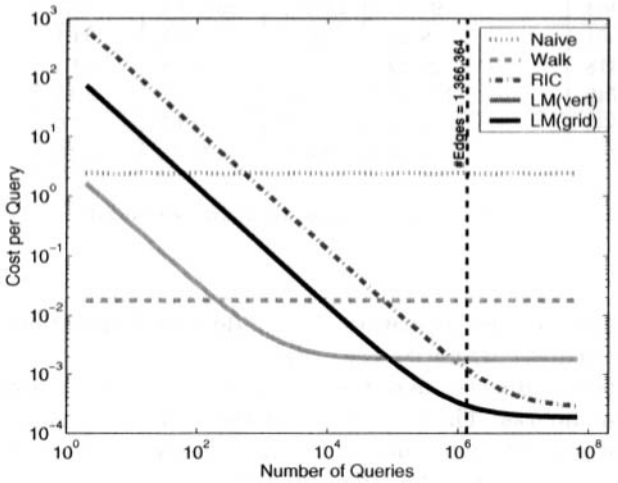


Figure 5: The average combined (amortized) cost per query in a large arrangement, with 1,366,384 edges.

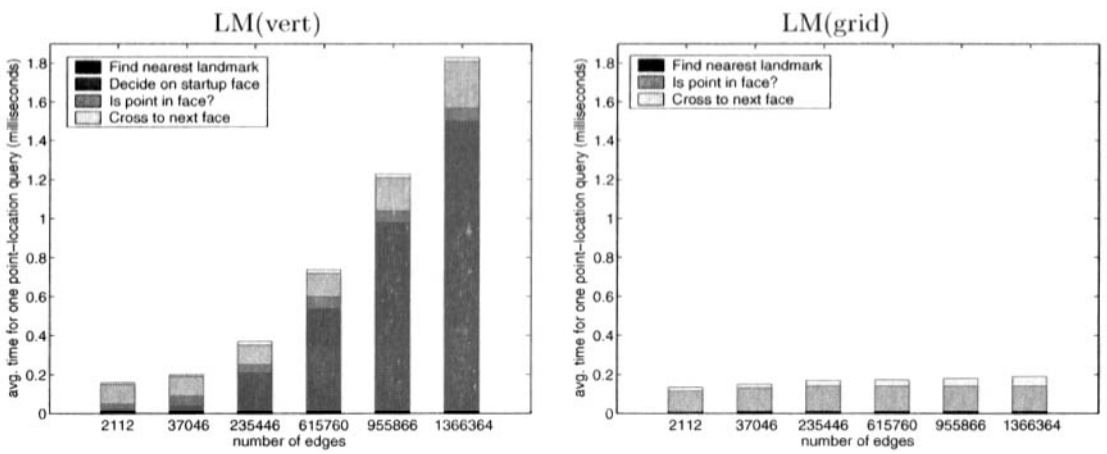


Figure 6: The average breakdown of the time required by the main steps of the Landmarks algorithms in a single point-location query, for arrangements of varying size.

into account both (amortized) preprocessing time and query time. Moreover, the memory space required by the algorithm is smaller compared to other algorithms that use auxiliary data structure for point location. The algorithm is easy to implement, maintain, and adjust for different needs using different kinds of landmarks and search structures.

It remains open to study the optimal number of landmarks required for arrangements of different sizes. This number should balance well between the time it takes to find the nearest landmark using the nearest-neighbor search structure, and the time it takes to walk from the landmark to the query point.

Acknowledgments

We wish to thank Ron Wein for his great help regarding conic-arc arrangements, and for his drawings. We also thank Efi Fogel for adjusting the benchmark for our needs, and Oren Nechushtan for testing the RIC algorithm implemented in CGAL.

References

[1] The CGAL project homepage. `http://www.cgal.org/`.

[2] The CORE library homepage. `http://www.cs.nyu.edu/exact/core_pages/`.

[3] The LEDA homepage. `http://www.algorithmic-solutions.com/enleda.htm`.

[4] The GNU MP bignum library. `http://www.swox.com/gmp/`.

[5] P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[6] S. Arya, T. Malamatos, and D. M. Mount. Entropy-preserving cutting and space-efficient planar point location. In *Proc. 12th ACM-SIAM Sympos. Disc. Alg.*, pages 256–261, 2001.

[7] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45:891–923, 1998.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[9] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *Comput. Geom. Theory Appl.*, 22(1–3):5–19.

[10] O. Devillers. The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.*, 13:163–180, 2002.

[11] O. Devillers and P. Guigue. The shuffling buffer. *Internat. J. Comput. Geom. Appl.*, 11:555–572, 2001.

[12] O. Devillers, S. Pion, and M. Teillaud. Walking in a triangulation. *Internat. J. Found. Comput. Sci.*, 13:181–199, 2002.

[13] L. Devroye, C. Lemaire, and J.-M. Moreau. Fast Delaunay point-location with search structures. In *Proc. 11th Canad. Conf. Comput. Geom.*, pages 136–141, 1999.

[14] L. Devroye, E. P. Mücke, and B. Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

[15] M. Edahiro, I. Kokubo, and T. Asano. A new point-location algorithm and its practical efficiency — comparison with existing algorithms. *ACM Trans. Graph.*, 3:86–109, 1984.

[16] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *J. Exp. Algorithmics*, 5:13, 2000.

[17] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving cgal's arrangements. In *Proc. 12th Annual European Symposium on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 664–676. Springer-Verlag, 2004.

[18] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, 2nd edition, 2004.

[19] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[20] J. Matoušek. *Geometric Discrepancy — An Illustrated Guide.* Springer, 1999.

[21] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing.* Cambridge University Press, Cambridge, UK, 2000.

[22] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1996.

[23] K. Mulmuley. A fast planar partition algorithm, I. *J. Symbolic Comput.*, 10(3-4):253–280, 1990.

[24] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *Regional Conference Series in Applied Mathematics.* CBMS-NSF, 1992.

[25] F. P. Preparata and M. I. Shamos. *Computational Geometry — An Introduction.* Springer, 1985.

[26] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, July 1986.

[27] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991.

Summarizing Spatial Data Streams Using ClusterHulls *

John Hershberger† Nisheeth Shrivastava‡ Subhash Suri‡

Abstract

We consider the following problem: given an on-line, possibly unbounded stream of two-dimensional points, how can we summarize its spatial distribution or *shape* using a small, bounded amount of memory? We propose a novel scheme, called *ClusterHull*, which represents the shape of the stream as a dynamic collection of convex hulls, with a total of at most m vertices, where m is the size of the memory. The algorithm dynamically adjusts both the number of hulls and the number of vertices in each hull to best represent the stream using its fixed memory budget. This algorithm addresses a problem whose importance is increasingly recognized, namely the problem of summarizing real-time data streams to enable on-line analytical processing. As a motivating example, consider habitat monitoring using wireless sensor networks. The sensors produce a steady stream of geographic data, namely, the locations of objects being tracked. In order to conserve their limited resources (power, bandwidth, storage), the sensors can compute, store, and exchange ClusterHull summaries of their data, without losing important geometric information. We are not aware of other schemes specifically designed for capturing shape information in geometric data streams, and so we compare ClusterHull with some of the best general-purpose clustering schemes such as CURE, k-median, and LSEARCH. We show through experiments that ClusterHull is able to represent the shape of two-dimensional data streams more faithfully and flexibly than the stream versions of these clustering algorithms.

1 Introduction

The extraction of meaning from data is perhaps the most important problem in all of science. Algorithms that can aid in this process by identifying useful structure are valuable in many areas of science, engineering, and information management. The problem takes many forms in different disciplines, but in many settings a *geometric abstraction* can be convenient: for instance, it helps formalize many informal but visually meaningful concepts such as similarity, groups, shape, etc. In many applications, geometric coordinates are a natural and integral part of data: e.g., locations of sensors in environmental monitoring, objects in location-aware computing, digital battlefield simulation, or meteorological data. Even when data have no intrinsic geometric association, many natural data analysis tasks such as clustering are best performed in an appropriate artificial coordinate space: e.g., data objects are mapped to points in some Euclidean space using certain attribute values, where similar objects (points) are grouped into spatial clusters for efficient indexing and retrieval. Thus we see that the problem of finding a simple characterization of a distribution known only through a collection of sample points is a fundamental one in many settings.

Recently there has been a growing interest in detecting patterns and analyzing trends in data that are generated continuously, often delivered in some fixed order and at a rapid rate. Some notable applications of such data processing include monitoring and surveillance using sensor networks, transactions in financial markets and stock exchanges, web logs and click streams, monitoring and traffic engineering of IP networks, telecommunication call records, retail and credit card transactions, and so on. Imagine, for instance, a surveillance application, where a remote environment instrumented by a wireless sensor network is being monitored through sensors that record the movement of objects (e.g., animals). The data gathered by each sensor can be thought of as a stream of two-dimensional points (geographic locations). Given the severe resource constraints of a wireless sensor network, it would be rather inefficient for each sensor to send its entire stream of raw data to a remote base sta-

*A partial summary of this work will be presented as a poster at ICDE '06, and represented in the proceedings by a three-page abstract.

†Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070, USA, and (by courtesy) Computer Science Department, University of California at Santa Barbara. john_hershberger@mentor.com.

‡Computer Science Department, University of California, Santa Barbara, CA 93106, USA. {nisheeth,suri}@cs.ucsb.edu. The research of Nisheeth Shrivastava and Subhash Suri was supported in part by National Science Foundation grants IIS-0121562 and CCF-0514738.

tion. Indeed, it would be far more efficient to compute and send a compact geometric summary of the trajectory. One can imagine many other remote monitoring applications like forest fire hazards, marine life, etc., where the shape of the observation point cloud is a natural and useful data summary. Thus, there are many sources of "transient" geometric data, where the key goal is to spot important trends and patterns, where only a small summary of the data can be stored, and where a "visual" summary such as shape or distribution of the data points is quite valuable to an analyst.

A common theme underlying these data processing applications is the continuous, real-time, large-volume, transient, single-pass nature of data. As a result, *data streams* have emerged as an important paradigm for designing algorithms and answering database queries for these applications. In the data stream model, one assumes that data arrive as a continuous stream, in some arbitrary order possibly determined by an adversary; the total size of the data stream is quite large; the algorithm may have memory to store only a tiny fraction of the stream; and any data not explicitly stored are essentially lost. Thus, data stream processing necessarily entails *data reduction*, where most of the data elements are discarded and only a small representative sample is kept. At the same time, the patterns or queries that the applications seek may require knowledge of the entire history of the stream, or a large portion of it, not just the most recent fraction of the data. The lack of access to full data significantly complicates the task of data analysis, because patterns are often hidden, and easily lost unless care is taken during the data reduction process. For simple database aggregates, sub-sampling can be appropriate, but for many advanced queries or patterns, sophisticated synopses or summaries must be constructed. Many such schemes have recently been developed for computing quantile summaries [21], most frequent or top-k items [23], distinct item counts [3, 24], etc.

When dealing with geometric data, an analyst's goal is often not as precisely stated as many of these *numerically-oriented* database queries. The analyst may wish to understand the general structure of the data stream, look for unusual patterns, or search for certain "qualitative" anomalies before diving into a more precisely focused and quantitative analysis. The "shape" of a point cloud, for instance, can convey important qualitative aspects of a data set more effectively than many numerical statistics. In a stream setting, where the data must be constantly discarded and compressed, special care must be taken to ensure that the sampling faithfully captures the overall shape of the point distribution.

Shape is an elusive concept, which is quite challenging even to define precisely. Many areas of computer science, including computer vision, computer graphics, and computational geometry deal with representation, matching and extraction of shape. However, techniques in those areas tend to be computationally expensive and unsuited for data streams. One of the more successful techniques in processing of data streams is clustering. The clustering algorithms are mainly concerned with identifying dense groups of points, and are not specifically designed to extract the boundary features of the cluster groups. Nevertheless, by maintaining some sample points in each cluster, one can extract some information about the geometric shape of the clusters. We will show, perhaps unsurprisingly, that ClusterHull, which explicitly aims to summarize the geometric shape of the input point stream using a limited memory budget, is more effective than general-purpose stream clustering schemes, such as CURE, k-median and LSEARCH.

1.1 ClusterHull

Given an on-line, possibly unbounded stream of two-dimensional points, we propose a scheme for summarizing its spatial distribution or *shape* using a small, bounded amount of memory m. Our scheme, called *ClusterHull*, represents the shape of the stream as a dynamic collection of convex hulls, with a total of at most m vertices. The algorithm dynamically adjusts both the number of hulls and the number of vertices in each hull to represent the stream using its fixed memory budget. Thus, the algorithm attempts to capture the shape by decomposing the stream of points into groups or clusters and maintaining an approximate convex hull of each group. Depending on the input, the algorithm adaptively spends more points on clusters with complex (potentially more interesting) boundaries and fewer on simple clusters. Because each cluster is represented by its convex hull, the ClusterHull summary is particularly useful for preserving such geometric characteristics of each cluster as its boundary shape, orientation, and volume. Because hulls are objects with spatial extent, we can also maintain additional information such as the number of input points contained within each hull, or their approximate *data density* (e.g., population divided by the hull volume). By shading the hulls in proportion to their density, we can then compactly convey a simple visual representation of the data distribution. By contrast, such information seems difficult to maintain in stream clustering schemes, because the cluster centers in those schemes

constantly move during the algorithm.

For illustration, in Figure 1 we compare the output of our ClusterHull algorithm with those produced by two popular stream-clustering schemes, k-median [19] and CURE [20]. The top row shows the input data (left), and output of ClusterHull (right) with memory budget set to $m = 45$ points. The middle row shows outputs of k-median, while the bottom row shows the outputs of CURE. One can see that both the boundary shapes and the densities of the point clusters are quite accurately summarized by the cluster hulls.

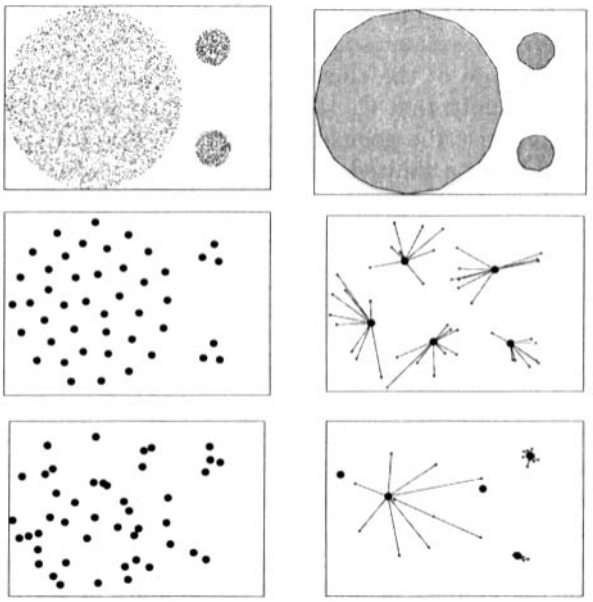


Figure 1: The top row shows the input data (left) and the output of ClusterHull (right) with memory budget of $m = 45$. The hulls are shaded in proportion to their estimated point density. The middle row shows two different outputs of the stream **k-medians** algorithm, with $m = 45$: in one case (left), the algorithm simply computes $k = 45$ cluster centers; in the other (right), the algorithm computes $k = 5$ centers, but maintains 9 (random) sample points from the cluster to get a rough approximation of the cluster geometry. (This is a simple enhancement implemented by us to give more expressive power to the k-median algorithm.) Finally, the bottom row shows the outputs of **CURE**: in the left figure, the algorithm computes $k = 45$ cluster centers; in the right figure, the algorithm computes $k = 5$ clusters, with $c = 9$ samples per cluster. CURE has a tunable shrinkage parameter, α, which we set to 0.4, in the middle of the range suggested by its authors [20].

We implemented ClusterHull and experimented with both synthetic and real data to evaluate its performance. In all cases, the representation by ClusterHull appears to be more information-rich than those by clustering schemes such as CURE, k-medians, or LSEARCH, even when the latter are enhanced with some simple mechanisms to capture cluster shape. Thus, our general conclusion is that ClusterHull can be a useful tool for summarizing geometric data streams.

ClusterHull is computationally efficient, and thus well-suited for streaming data. At the arrival of each new point, the algorithm must decide whether the point lies in one of the existing hulls (actually, within a certain ring around each hull), and possibly merge two existing hulls. With appropriate data structures, this processing can be done in amortized time $O(\log m)$ per point.

ClusterHull is a general paradigm, which can be extended in several orthogonal directions and adapted to different applications. For instance, if the input data are *noisy*, then covering all points by cluster hulls can lead to poor shape results. We propose an *incremental cleanup* mechanism, in which we periodically discard light-weight hulls, that deals with noise in the data very effectively. Similarly, the performance of a shape summary scheme can depend on the order in which input is presented. If points are presented in a bad order, the ClusterHull algorithm may create long, skinny, inter-penetrating hulls early in the stream processing. We show that a *period-doubling cleanup* is effective in correcting the effects of these early mistakes. When there is spatial coherence within the data stream, our scheme is able to exploit that coherence. For instance, imagine a point stream generated by a sensor field monitoring the movement of an *unknown* number of vehicles in a two-dimensional plane. The data naturally cluster into a set of spatially coherent trajectories, which our algorithm is able to isolate and represent more effectively than general-purpose clustering algorithms.

1.2 Related Work

Inferring shape from an unordered point cloud is a well-studied problem that has been considered in many fields, including computer vision, machine learning, pattern analysis, and computational geometry [4, 10, 11, 26]. However, the classical algorithms from these areas tend to be computationally expensive and require full access to data, making them unsuited for use in a data stream setting.

An area where significant progress has occurred on stream algorithms is *clustering*. Our focus is some-

what different from classical clustering—we are mainly interested in low-dimensional data and capturing the "surface" or boundary of the point cloud, while clustering tends to focus on the "volume" or density and moderate and large dimensions. While classical clustering schemes of the past have focused on cluster centers, which work well for spherical clusters, some recent work has addressed the problem of non-spherical clusters, and tried to pay more attention to the geometry of the clusters. Still this attention to geometry does not extend to the *shape* of the boundary.

Our aim is not to exhaustively survey the clustering literature, which is immense and growing, but only to comment briefly on those clustering schemes that could potentially be relevant to the problem of summarizing shape of two- or three-dimensional point streams. Many well-known clustering schemes (e.g., [5, 7, 16, 25]) require excessive computation and require multiple passes over the data, making them unsuited for our problem setting. There are machine-learning based clustering schemes [12, 13, 27], that use classification to group items into clusters. These methods are based on statistical functions, and not geared towards shape representation. Clustering algorithms based on spectral methods [8, 14, 18, 28] use the singular value decomposition on the similarity graph of the data, and are good at clustering statistical data, especially in high dimensions. We are unaware of any results showing that these methods are particularly effective at capturing boundary shapes, and, more importantly, streaming versions of these algorithms are not available. So, we now focus on clustering schemes that work on streams and are designed to capture some of the geometric information about clusters.

One of the popular clustering schemes for large data sets is BIRCH [30], which also works on data streams. An extension of BIRCH by Aggarwal et al. [2] also computes multi-resolution clusters in evolving streams. While BIRCH appears to work well for spherical-shaped clusters of uniform size, Guha et al. [20] experimentally show that it performs poorly when the data are clustered into groups of unequal sizes and different shapes. The CURE clustering scheme proposed by Guha et al. [20] addresses this problem, and is better at identifying non-spherical clusters. CURE also maintains a number of sample points for each cluster, which can be used to deduce the geometry of the cluster. It can also be extended easily for streaming data (as noted in[19]). Thus, CURE is one of the clustering schemes we compare against ClusterHull.

In [19], Guha et al. propose two stream variants of k-center clustering, with provable theoretical guarantees as well as experimental support for their performance. The stream k-median algorithm attempts to minimize the sum of the distances between the input points and their cluster centers. Guha et al. [19] also propose a variant where the number of clusters k can be relaxed during the intermediate steps of the algorithm. They call this algorithm LSEARCH (local search). Through experimentation, they argue that the stream versions of their k-median and LSEARCH algorithms produce better quality clusters than BIRCH, although the latter is computationally more efficient. Since we are chiefly concerned with the quality of the shape, we compare the output of ClusterHull against the results of k-median and LSEARCH (but not BIRCH).

1.3 Organization

The paper is organized in seven sections. Section 2 describes the basic algorithm for computing cluster hulls. In Section 3 we discuss the cost function used in refining and unrefining our cluster hulls. Section 4 provides extensions to the basic ClusterHull algorithm. In Sections 5 and 6 we present some experimental results. We conclude in Section 7.

2 Representing Shape as a Cluster of Hulls

We are interested in simple, highly efficient algorithms that can identify and maintain *bounded-memory approximations* of a stream of points. Some techniques from computational geometry appear especially well-suited for this. For instance, the *convex hull* is a useful shape representation of the *outer boundary* of the whole data stream. Although the convex hull accurately represents a convex shape with an arbitrary aspect ratio and orientation, it loses all the internal details. Therefore, when the points are distributed non-uniformly within the convex hull, the outer hull is a poor representation of the data.

Clustering schemes, such as k-medians, partition the points into groups that may represent the distribution better. However, because the goal of many clustering schemes is typically to minimize the maximum or the sum of distance functions, there is no explicit attention given to the shape of clusters—each cluster is conceptually treated as a ball, centered at the cluster center. Our goal is to mediate between the two extremes offered by the convex hull and k-medians. We would like to combine the best features of the convex hull—its ability to represent convex shapes with any

aspect ratio accurately—with those of ball-covering approximations such as k-medians—their ability to represent nonconvex and disconnected point sets. With this motivation, we propose the following measure for representing the shape of a point set under the bounded memory constraint.

> *Given a two-dimensional set of N points, and a memory budget of m, where $m \ll N$, compute a set of convex hulls such that (1) the collection of hulls uses at most m vertices, (2) the hulls together cover all the points of S, and (3) the total area covered by the hulls is minimized.*

Intuitively, this definition interpolates between a single convex hull, which potentially covers a large area, and k-medians clustering, which fails to represent the shape of individual clusters accurately. Later we will relax the condition of "covering all the points" to deal with *noisy data*—in the relaxed problem, *a constant fraction* of the points may be dropped from consideration. But the general goal will remain the same: to compute a set of convex hulls that attempts to cover the important geometric features of the data stream using least possible area, under the constraint that the algorithm is allowed to use at most m vertices.

2.1 Geometric approximation in data streams

Even the classical convex hull (outer boundary) computation involves some subtle and nontrivial issues in the data stream setting. What should one do when the number of extreme vertices in the convex hull exceeds the memory available? Clearly, some of the extreme vertices must be dropped. But which ones, and how shall we measure the *error* introduced in this approximation? This problem of summarizing the convex hull of a point stream using a fixed memory m has been studied recently in computational geometry and data streams [1, 6, 9, 17, 22]. An adaptive sampling scheme proposed in [22] achieves an optimal memory-error tradeoff in the following sense: given memory m, the algorithm maintains a hull that (1) lies within the true convex hull, (2) uses at most m vertices, and (3) approximates the true hull well—any input point not in the computed hull lies within distance $O(D/m^2)$ of the hull, where D is the diameter of the point stream. Moreover, the error bound of $O(D/m^2)$ is the best possible in the worst case.

In our problem setting, we will maintain not one but many convex hulls, depending on the geometry of the stream, with each hull roughly corresponding to a cluster. Moreover, the locations of these hulls are not determined *a priori*—rather, as in k-medians, they are dynamically determined by the algorithm. Unlike k-medians clusters, however, each hull can use a different fraction of the available memory to represent its cluster boundary. One of the key challenges in designing the ClusterHull algorithm is to formulate a good policy for this memory allocation. For this we will introduce a cost function that the various hulls use to decide how many hull vertices each gets. Let us first begin with an outline of our scheme.

2.2 The basic algorithm

The available memory m is divided into two pools: a fixed pool of k groups, each with a constant number of vertices; and a shared pool of $O(k)$ points, from which different cluster hulls draw additional vertices. The number k has the same rôle as the parameter fed to k-medians clustering—it is set to some number at least as large as the number of native clusters expected in the input. (Thus our representation will maintain a more refined view of the cluster structure than necessary, but simple post-processing can clean up the unnecessary sub-clustering.) The exact constants in this division are tunable, and we show their effect on the performance of the algorithm through experimentation. For the sake of concreteness, we can assume that each of the k groups is initially allocated 8 vertices, and the common pool has a total of $8k$ vertices. Thus, if the available memory is m, then we must have $m \geq 16k$.

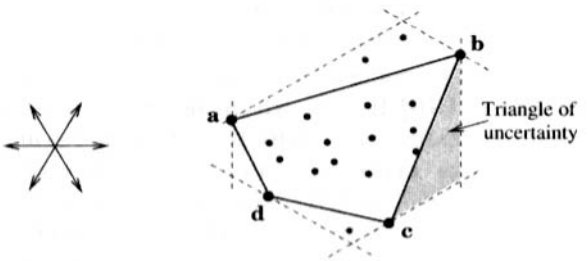


Figure 2: An approximate hull, with 6 sampling directions. The sample hull's vertices are a, b, c, d.

Our algorithm approximates the convex hull of each group by its extreme vertices in selected (sample) directions: among all the points assigned to this cluster group, for each sample direction, the algorithm retains the extreme vertex in that direction. See Figure 2 for an example. Each edge of this sampled hull supports what we call an *uncertainty triangle*—the triangle formed by the edge and the tangents at the two

endpoints of the edge in the sample directions for which those endpoints are extreme. A simple but important property of the construction is that the boundary of the true convex hull is sandwiched in the ring of uncertainty triangles defined by the edges of the computed hull. See Figure 3 for an illustration. The extremal directions are divided into two sets, one containing uniformly-spaced fixed directions, corresponding to the initial endowment of memory, and another containing adaptively chosen directions, corresponding to additional memory drawn from the common pool. The adaptive directions are added incrementally, bisecting previously chosen directional intervals, to minimize the error of the approximation.

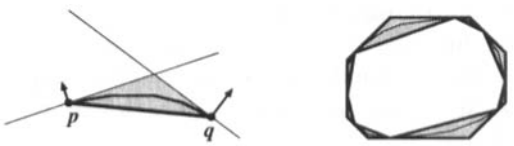


Figure 3: The true hull is sandwiched in a ring of uncertainty triangles.

Each hull has an individual cost associated with it, and the whole collection of k hulls has a total cost that is the sum of the individual costs. Our goal is to choose the cost function such that minimizing the total cost leads to a set of approximate convex hulls that represent the shape of the point set well. Furthermore, because our minimization is performed on-line, assigning each new point in the stream to a convex hull when the point arrives, we want our cost function to be *robust*: as much as possible, we want it to reduce the chance of assigning early-arriving points to hulls in a way that forces late-arriving points to incur high cost. We leave the technical details of our choice of the cost function to the following section.

Let us now describe the high-level organization of our algorithm. Suppose that the current point set S is partitioned among k convex hulls $H_1, \ldots, H_k$. The cost of hull H_i is $w(H_i)$, and the total cost of the partition $\mathcal{H} = \{H_1, \ldots, H_k\}$ is $w(\mathcal{H}) = \sum_{H \in \mathcal{H}} w(H)$. We process each incoming point p with the following algorithm:

Algorithm ClusterHull

if p is contained in any $H \in \mathcal{H}$, or in the ring of uncertainty triangles for any such H, **then**
  Assign p to H without modifying H.
else
  Create a new hull containing only p and add it to $\mathcal{H}$.
if $|\mathcal{H}| > k$ **then**
  Choose two hulls $H, H' \in \mathcal{H}$ such that merging H and H' into a single convex hull will result in the minimum increase to $w(\mathcal{H})$.
  Remove H and H' from $\mathcal{H}$, merge them to form a new hull H^*, and put that into $\mathcal{H}$.
  If H^* has an uncertainty triangle over either edge joining points of the former H and H' whose height exceeds the previous maximum uncertainty triangle height, refine (repeatedly bisect) the angular interval associated with that uncertainty triangle by choosing new adaptive directions until the triangle height is less than the previous maximum.
  while the total number of adaptive directions in use exceeds ck
    Unrefine (discard one of the adaptive directions for some $H \in \mathcal{H}$) so that the uncertainty triangle created by unrefinement has minimum height.

The last two steps (refinement and unrefinement) are technical steps for preserving the approximation quality of the convex hulls that were introduced in [22]. The key observation is that an uncertainty triangle with "large height" leads to a poor approximation of a convex hull. Ideally, we would like uncertainty triangles to be flat. The height of an uncertainty triangle is determined by two key variables: the length of the convex hull edge, and the angle-difference between the two sampling directions that form that triangle. More precisely, consider an edge $\overline{pq}$. We can assume that the extreme directions for p and q, namely, θ_p and θ_q, point toward the same side of $\overline{pq}$, and hence the intersection of the supporting lines projects perpendicularly onto $\overline{pq}$. Therefore the height of the uncertainty triangle is at most the edge length $\ell(\overline{pq})$ times the tangent of the smaller of the angles between $\overline{pq}$ and the supporting lines. Observe that the sum of these two angles equals the angle between the directions θ_p and θ_q. If we define $\theta(\overline{pq})$ to be $|\theta_p - \theta_q|$, then the height of the uncertainty triangle at $\overline{pq}$ is at most $\ell(\overline{pq}) \cdot \tan(\theta(\overline{pq})/2)$, which is closely approximated by

$$\frac{\ell(\overline{pq}) \cdot \theta(\overline{pq})}{2}. \tag{2.1}$$

This formula forms the basis for adaptively choosing new sampling directions: we devote more sampling directions to cluster hull edges whose uncertainty triangles have large height. Refinement is the process of introducing a new sampling direction that bisects two consecutive sampling directions; unrefinement is the converse of this process. The analysis in [22] showed

that if a single convex hull is maintained using $m/2$ uniformly spaced sampling directions, and $m/2$ adaptively chosen directions (using the policy of minimizing the maximum height of an uncertainty triangle), then the maximum distance error between true and approximate hulls is $O(D/m^2)$. Because in ClusterHull we share the refinement directions among k different hulls, we choose them to minimize the global maximum uncertainty triangle height explicitly. We point out that the allocation of adaptive directions is independent of the cost function $w(\mathcal{H})$. The cost function guides the partition into convex hulls; once that choice is made, we allocate adaptive directions to minimize the error for that partition. One could imagine making the assignment of adaptive directions dependent on the cost function, but for simplicity we have chosen not to do so.

3 Choosing a Cost Function

In this section we describe the cost function we apply to the convex hulls that ClusterHull maintains. We discuss the intuition behind the cost function, experimental support for that intuition, and variants on the cost function that we considered.

The *α-hull* is a well-known structure for representing the shape of a set of points [15]. It can be viewed as an extension of the convex hull in which half-planes are replaced by the complements of fixed-radius disks (i.e., the regions outside the disks). In particular, the convex hull is the intersection of all half-planes containing the point set, and the α-hull is the intersection of all disk-complements with radius ρ that contain the point set.[1] See Figure 4 for examples of the convex hull and α-hull on an L-shaped point set. The α-hull minimizes the area of the shape that covers the points, subject to the radius constraint on the disks.

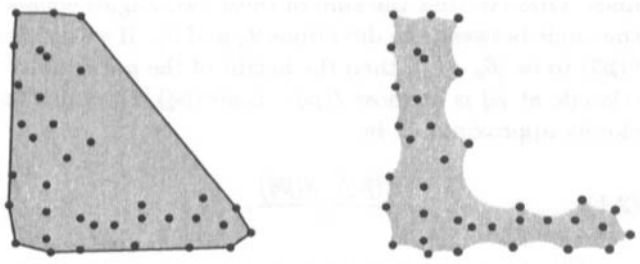


Figure 4: Shape representations for a set of points: (left) convex hull, (right) α-hull.

[1] In the definition of α-hulls, the disk radius $\rho = 1/|\alpha|$, and $\alpha \leq 0$, but we are not concerned with these technical details.

The α-hull is not well suited to represent the shape of a stream of points, because an unbounded number of input points may appear on the boundary of the shape. Our goal of covering the input points with bounded-complexity convex hulls of minimum total area is an attempt to mimic the modeling power of the α-hull in a data stream setting.

Although our goal is to minimize the total area of our convex hull representation, we use a slightly more complex function as the cost of a convex hull H:

$$w(H) = \texttt{area}(H) + \mu \cdot (\texttt{perimeter}(H))^2. \tag{3.2}$$

Here μ is a constant, chosen empirically as described below. Note that the perimeter is squared in this expression to match units: if the perimeter term entered linearly, then simply changing the units of measurement would change the relative importance of the area and perimeter terms, which would be undesirable.

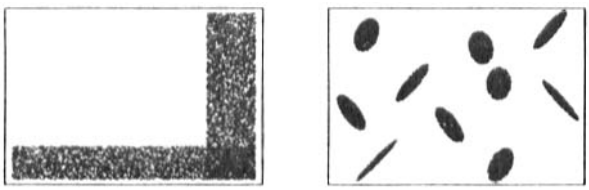


Figure 5: Input distributions: L-shaped and *ellipses*.

We want to minimize total area, and so defining $w(H) = \texttt{area}(H)$ seems natural; however, this proves to be infeasible in a stream setting. If a point set has only two points, the area of its convex hull is zero; thus all such hulls have the same cost. The first $2k$ points that arrive in a data stream are paired up into k two-point convex hulls, each with cost zero, and the pairing will be arbitrary. In particular, some convex hulls are likely to cross natural cluster boundaries. When these clusters grow as more points arrive, they will have higher cost than the optimal hulls that would have been chosen by an off-line algorithm. This effect is clearly visible in the clusters produced by our algorithm in Figure 6 (right) for the *ellipses* data set of Figure 5 (right). By contrast, the L-shaped distribution of Figure 5 (left) is recovered well using the area cost function, as shown in Figure 6 (left).

We can avoid the tendency of the area cost to create long thin needles in the early stages of the stream by minimizing the perimeter. If we choose $w(H) = \texttt{perimeter}(H)$, then the well-separated clusters of the *ellipses* data set are recovered perfectly, even when the points arrive on-line—see Figure 7 (right).

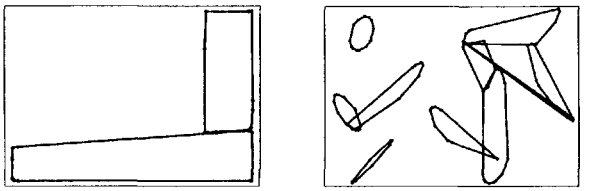


Figure 6: With the area cost function, ClusterHull faithfully recovers the L-shaped distribution of points. But it performs poorly on a set of $n = 10,000$ points distributed among ten elliptical clusters; it merges pairs of points from different groups and creates intersecting hulls.

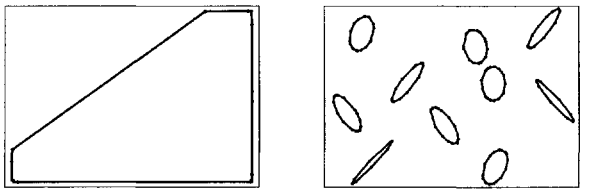


Figure 7: With the perimeter cost function, ClusterHull faithfully recovers the disjoint elliptical clusters, but performs poorly on the L-shaped distribution.

However, as the poor recovery of the L distribution shows (Figure 7 (left)), the perimeter cost has its own liabilities. The total perimeter of two hulls that are relatively near each other can often be reduced by merging the two into one. Furthermore, merging two large hulls reduces the perimeter more than merging two similar small ones, and so the perimeter cost applied to a stream often results in many small hulls and a few large ones that contain multiple "natural" clusters.

We need to incorporate both area and perimeter into our cost function to avoid the problems shown in Figures 6 and 7. Because our overall goal is to minimize area, we choose to keep the area term primary in our cost function (Equation 3.2). In that function $(\texttt{perimeter}(H))^2$ is multiplied by a constant μ, which is chosen to adjust the relative importance of area and perimeter in the cost. Experimentation shows that choosing $\mu = 0.05$ gives good shape reconstruction on a variety of inputs. With μ substantially smaller than 0.05, the perimeter effect is not strong enough, and with μ greater than 0.1, it is too strong. (Intuitively, we want to add just enough perimeter dependence to avoid creating needle convex hulls in the early stages of the stream.)

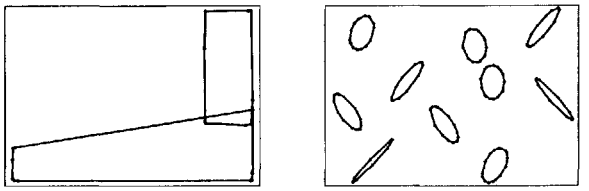


Figure 8: With the combined area and perimeter cost function, the algorithm ClusterHull recovers both the ellipse and L distributions. The choice of $\mu = 0.05$ gives good shape reconstruction.

We can understand the combined area-perimeter cost by modeling it as the area of a fattened convex hull. If we let $\rho = \mu \cdot \texttt{perimeter}(H)$, we see that the area-perimeter cost (3.2) is very close to the area obtained by fattening H by ρ. The true area is $\texttt{area}(H) + \rho \cdot \texttt{perimeter}(H) + \pi\rho^2 = \texttt{area}(H) + \rho^2(\frac{1}{\mu} + \pi)$; if μ is small, then $1/\mu$ is relatively large compared to π, and the extra $\pi\rho^2$ term is not very significant.

Because the cost (3.2) may fatten long thin clusters more than is desirable, we also experimented with replacing the constant μ in (3.2) by a value inversely related to the aspect ratio of H. The aspect ratio of H is $\texttt{diam}(H)/\texttt{width}(H) = \Theta((\texttt{perimeter}(H))^2/\texttt{area}(H))$. Thus if we simply replaced μ by $1/\texttt{aspectRatio}(H)$ in (3.2), we would essentially obtain the area cost. We compromised by using the cost

$$w(H) = \texttt{area}(H) + \mu \cdot (\texttt{perimeter}(H))^2/(\texttt{aspectRatio}(H))^x$$

for various values of x ($x = 0.5$, $x = 0.1$). The aspect ratio is conveniently approximated as $(\texttt{perimeter}(H))^2/\texttt{area}(H)$, since the quantities in that expression are already maintained by our convex hull approximation. Except in extreme cases, the results with this cost function were not enough different from the basic area-perimeter cost to merit a separate figure.

The cost (3.2) fattens each hull by a radius proportional to its own perimeter. This is appropriate if the clusters have different natural scales and we want to fatten each according to its own dimensions. However, in our motivating structure the α-hull, a uniform radius is used to define all the clusters. To fatten hulls uniformly, we could use the weight function

$$w(H) = \texttt{area}(H) + \rho \cdot \texttt{perimeter}(H) + \pi\rho^2.$$

However, the choice of the fattening radius ρ is problematic. We might like to choose ρ such that α-hulls

defined using radius-ρ disks form exactly k clusters, but then the optimum value of ρ would decrease and increase as the stream points arrived. We can avoid these difficulties by sticking to the simpler cost of definition (3.2).

4 Extensions and Enhancements

In this section we discuss how to enhance the basic ClusterHull algorithm to improve the quality of shape representation.

4.1 Spatial incoherence and period-doubling cleanup

In many data streams the arriving points are ordered arbitrarily, possibly even adversarily. The ClusterHull scheme (and indeed any on-line clustering algorithm) is vulnerable to early errors, in which an early-arriving point is assigned to a hull that later proves to be the wrong one.

Figure 9 (left) shows a particularly bad input consisting of five thin parallel stripes. We used ClusterHull with $\mu = 0.05$ to maintain five hulls, with the input points ordered randomly. A low density sample from the stripe distribution (such as a prefix of the stream) looks to the algorithm very much like uniformly distributed points. Early hull merges combine hulls from different stripes, and the ClusterHull algorithm cannot recover from this mistake. See Figure 9 (right).

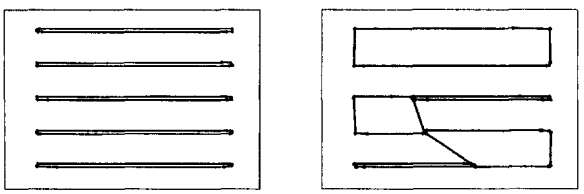


Figure 9: Processing the *stripes* input (left) in random order leads to errors for our algorithm (right).

If the input data arrive in random order, the idea of *period-doubling cleanup* may help identify and amplify the true clusters. The idea is to process the input stream in *rounds* in which the number of points processed doubles in each round. At the end of each round we identify low density hulls and discard them—these likely group points from several true clusters. The dense hulls are retained from round to round, and are allowed to grow.

Formally, the period-doubling cleanup operates as follows: For each $H \in \mathcal{H}$ we maintain the number of points it represents, denoted by $\texttt{count}(H)$. The density of any hull H is $\texttt{density}(H) = \texttt{count}(H)/\texttt{area}(H)$. The algorithm also maintains an approximate convex hull G of all the input points. After each round, it discards from $\mathcal{H}$ every hull H for which any of the following holds:

- $\texttt{count}(H) < \delta \cdot N/k$
- $\texttt{density}(H) < \texttt{density}(G)$
- $\texttt{density}(H) < \frac{1}{2} \cdot \texttt{median}\{\texttt{density}(\texttt{A}) : \texttt{A} \in \mathcal{H}\}$

Here N is the number of points seen so far. In our experiments, we set the tunable parameter δ to 0.1.

The first test takes care of hulls with a very small number of tightly clustered points (these may have high densities because of their smaller area, and will not be caught by density pruning). The second test discards hulls that have less than average density. The intuition is that each cluster should be at least as dense as the entire input space (otherwise it is not an interesting cluster). In case the points are distributed over a very large area, but the individual clusters are very compact, the average density may not be very helpful for discarding hulls. Instead, we should discard hulls that have low densities relative to other hulls in the data structure; the third test takes care of this case—it discards any hull with density significantly less than the median density.

Figure 10 (left) shows the result of period-doubling cleanup on the *stripes* distribution; the sparse hulls that were initially found have been discarded and five dense hulls have been correctly computed. We note that, with the same amount of memory, neither CURE nor the k-median clustering is able to represent the *stripes* distribution well (cf. Figure 10). Our experiments show that applying period-doubling cleanup helps improve clustering on almost all data sets.

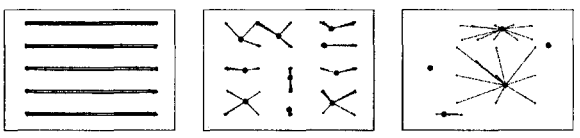


Figure 10: Period-doubling cleanup (left) on ClusterHull corrects the errors in the *stripes* distribution; the middle figure shows the output of k-medians, and the right figure shows the output of CURE.

4.2 Noisy data and incremental cleanup

Sampling error and outliers cause difficulty for nearly all clustering algorithms. Likewise, a few outliers can adversely affect the ClusterHull shape summary. An algorithm needs some way to distinguish between dense regions of the input distribution (the true clusters) and sparse ones (noise). In this section, we propose an *incremental cleanup* mechanism that can improve the performance of our algorithm in the presence of noise. Both the period-doubling and the incremental cleanup are inspired by sampling techniques used in frequency estimation in streams. In particular, period-doubling is inspired by sticky sampling and incremental cleanup is inspired by lossy counting [23]. The incremental cleanup also processes the input in rounds, but the rounds do not increase in length. This is because noise is not limited to the beginning of the input; if we increased the round length, all the hulls would be corrupted by noisy points. Instead, we fix the size of each round depending on the (estimated) noise in the input.

Specifically, the incremental cleanup assumes that the input stream consists of points drawn randomly from a fixed distribution, with roughly $(1 - \epsilon)N$ of them belonging to high density clusters and ϵN of them being low density noise. The expected noise frequency ϵ affects the quality of the output. We can estimate it conservatively if it is unknown. The idea is to set the value of δ to be roughly equal to ϵ, and process the input in rounds of $k/(2\epsilon)$ points. The logic is that in every round, only about $k/2$ hulls will be corrupted by noisy points, still leaving half of the hulls untouched and free to track the true distribution of the input. If we set k to be more than twice the expected number of natural clusters in the input, we obtain a good representation of the clusters.

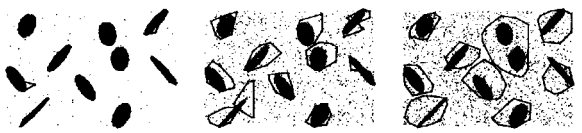


Figure 11: Incremental cleanup, with estimated noise frequency $\epsilon = 0.1$, applied to distributions with 1%, 10%, and 20% actual background noise.

This scheme propagates the good hulls (those with high density and high cardinality) from one round of the algorithm to the next, while discarding hulls that are sparse or belong to outliers. See Figure 11 for an example of how this scheme identifies true clusters and discards noisy regions. Of course, if noise is underestimated significantly (Figure 11 (right)), the quality of the cluster hulls suffers.

4.3 Spatial coherence and trajectory tracking

Section 4.1 considered spatially incoherent input streams. If the input *is* spatially coherent, as occurs in some applications, ClusterHull performs particularly well. If the input stream consists of locations reported by sensors detecting some moving entity (a light pen on a tablet, a tank in a battlefield, animals in a remote habitat), our algorithm effectively finds a covering of the trajectory by convex "lozenges." The algorithm also works well when there are multiple simultaneous trajectories to represent, as might occur when sensors track multiple independent entities. If the stripes of Figure 9 are fed to the algorithm in left-to-right order they are recovered perfectly; likewise in Figure 12 a synthetic trajectory is represented accurately.

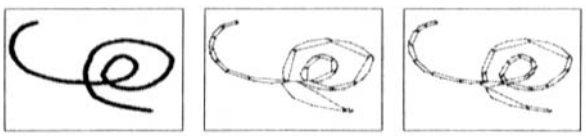


Figure 12: Input along a trajectory in (a); the shape is recovered well using $m = 100$ in (b), and using $m = 150$ in (c).

4.4 Density estimation and display

Stream versions of traditional clustering schemes (including k-median and CURE) do not include an estimate of the density of points associated with each cluster center, whereas each cluster hull H can easily maintain `count`(H). As in Section 4.1, this gives an estimate of the density of the points in each hull. If this information is displayed graphically (cf. Figure 13) it conveys more insight about the distribution of the input data than does the simple cluster-center output or even cluster-sample output.

5 Implementation and Experiments

We implemented the convex hull algorithm of [22] and the ClusterHull algorithm on top of it. The

convex hull algorithm takes logarithmic time per inserted point, on the average, but our ClusterHull implementation is more simple-minded, optimized more for ease of implementation than for runtime performance. The bottleneck in our implementation is neighborhood queries/point location, taking time proportional to the number of hulls. By storing the hull edges in a quad-tree, we could speed up these operations to $O(\log m)$ time.

When a new point arrives, we must check which hull it belongs to, if any. Using a quad-tree, this reduces to a logarithmic-time search, followed by logarithmic-time point-in-polygon tests with an expected constant number of hulls. Each new hull H must compute its optimum merge cost—the minimum increment to $w(\mathcal{H})$ caused by merging H with another hull. On average, this increment is greater for more distant hulls. Using the quad-tree we can compute the increment for $O(1)$ nearby hulls first. Simple lower bounds on the incremental cost of distant merges then let us avoid computing the costs for distant hulls. Computing the incremental cost for a single pair of hulls reduces to computing tangents between the hulls, which takes logarithmic time [22].

Merging hulls eliminates some vertices forever, and so we can charge the time spent performing the merge to the deleted vertices. Thus a careful implementation of the ClusterHull algorithm would process stream points in $O(\log m)$ amortized time per point, where m is the total number of hull vertices.

In the remainder of this section we evaluate the performance of our algorithm on different data sets. When comparing our scheme with k-median clustering [19], we used an enhanced version of the latter. The algorithm is allowed to keep a constant number of sample points per cluster, which can be used to deduce the approximate shape of that cluster. We ran the k-medians clustering using k clusters and total memory (number of samples) equal to m. CURE already has a parameter for maintaining samples in each cluster, so we used that feature. In this section, we analyze the output of these three algorithms (ClusterHull, k-median, CURE) on a variety of different data sets, and as a function of m, the memory.

Throughout this section, we use the *period-doubling cleanup* along with the area-perimeter cost (Equation 3.2) to compute the hulls. We use $\mu = .05$ and r, the number of initial sample directions per hull, equal to 8. The values of these parameters are critical for our algorithm; however, in this section *we use the same set of parameters for all data sets.* This shows that when tuned properly, our algorithm can generate good quality clusters for a variety of input distributions using a single set of parameters. In the next section, we will analyze in detail the effects of these parameters on the results of our scheme. To visualize the output, we also shade the hulls generated by our algorithm according to their densities (darker regions are more dense).

5.1 West Nile virus spread

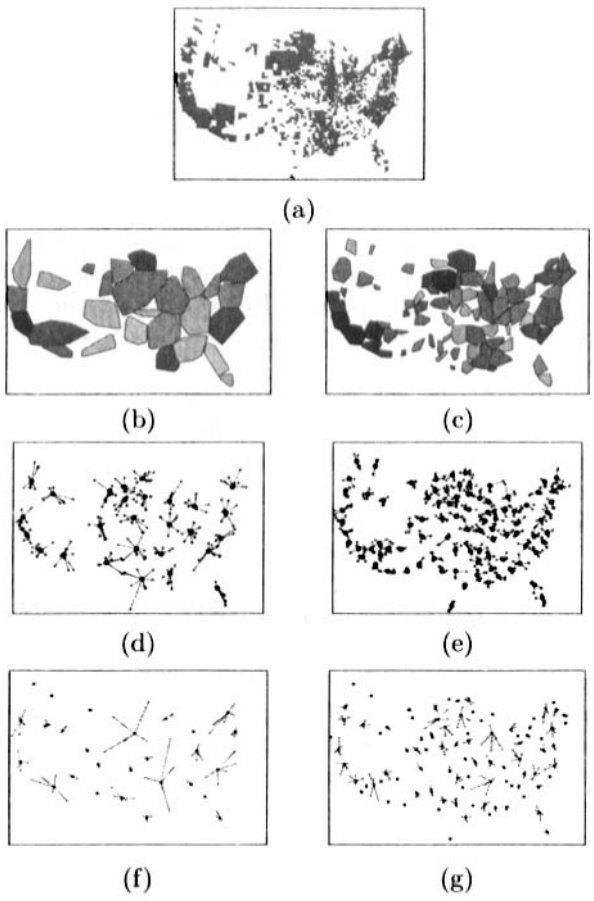


Figure 13: The *westnile* data set is shown in the top figure (a). Figures (b) and (c) show the outputs of ClusterHull for $m = 256$ and $m = 512$. Figures (d) and (e) show the corresponding outputs for k-medians. Figures (f) and (g) show the corresponding outputs for CURE.

Our first data set, *westnile* (Figure 13 (a)), contains about 68,000 points corresponding to the locations of the *West Nile* virus cases reported in the US, as collected by the CDC and the USGS [29]. We randomized the input order to eliminate any spatial coherence that might give an advantage to our algorithm. We ran ClusterHull to generate output of total size $m = 256$

and 512 (Figures (b) and (c)). The clustering algorithms k-medians and CURE were used to generate clusters with the same amount of memory. The results are shown in Figure 13.

All three algorithms are able to track high-density regions in coherent clusters, but there was little information about the shapes of the clusters in the output of k-medians or CURE. Visually the output of ClusterHull looks strikingly similar to the input set, offering the analyst a faithful yet compact representation of the geometric shapes of important regions.

5.2 The *circles* and the *ellipse* data sets

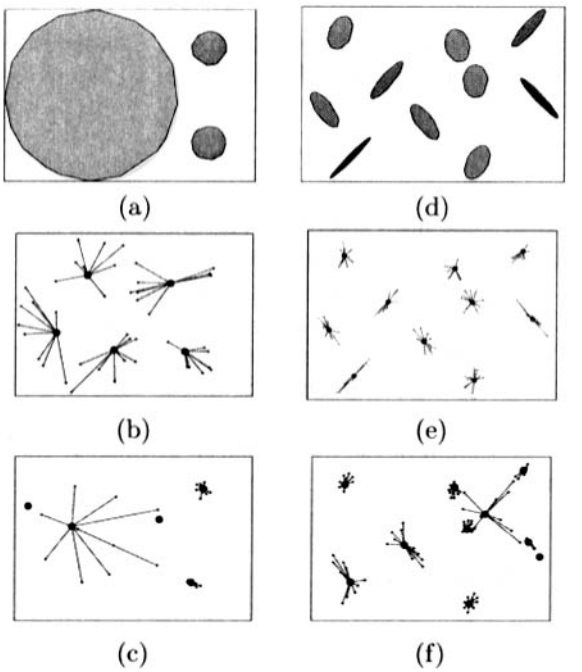


Figure 14: The left column shows output of ClusterHull (top), k-medians (middle) and CURE (bottom) for *circles* dataset with $m = 64$. The right column shows corresponding outputs for *ellipse* dataset with $m = 128$.

In this experiment, we compared ClusterHull with k-median and CURE on the *circles* and the *ellipse* data sets described earlier. The circles set contains $n = 10,000$ points generated inside 3 circles of different sizes. We ran the three algorithms with a total memory $m = 64$. The output of ClusterHull is shown in Figure 14 (a); the output of k-median is shown in (b); and the output of CURE is shown in (c).

Similarly, Figures 14 (d), (e), and (f), respectively, show the outputs of ClusterHull, k-median, and CURE on the ellipse data set with memory $m = 128$. The ellipse data set contains $n = 10,000$ points distributed among ten ellipse-shaped clusters.

In all cases, ClusterHull output is more accurate, visually informative, and able to compute the boundaries of clusters with remarkable precision. The outputs of other schemes are ambiguous, inaccurate, and lacking in details of the cluster shape boundary. For the circles data, the k-median does a poor job in determining the true cluster structure. For the ellipse data, CURE does a poor job in separating the clusters. (CURE needed a much larger memory—a "window size" of at least 500—to separate the clusters correctly.)

6 Tuning ClusterHull Parameters

In this section, we study the effects of various parameters on the quality of clusters.

6.1 Variation with r

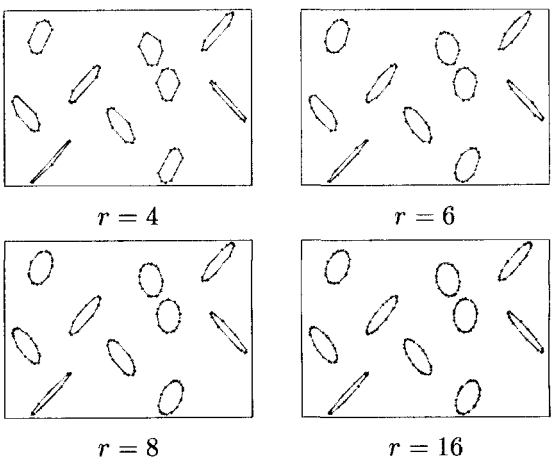


Figure 15: The result of varying r on the *ellipses* data set.

We first consider the effect of changing r, the number of initial directions assigned to each hull. To isolate the effects of r, we fixed the values of $\mu = .05$ and $k = 10$. We ran the experiments on two data sets, *ellipses* and *circles*. The results are shown in Figures 15 and 16, respectively.

The results show that the shape representation with 4 initial directions is very crude: ellipses are turned into pointy polygons. As we increase r, the representation of clusters becomes more refined. This contrast can be seen if we compare the boundary of

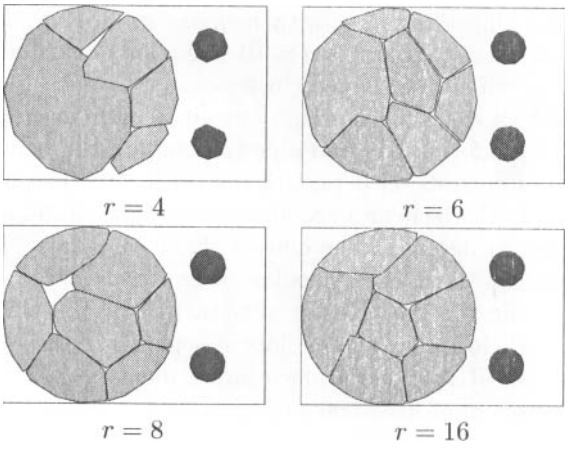


Figure 16: The result of varying r on the *circles* data set.

the big circle in Figure 16 for $r = 4$ and 8. However, increasing the number of directions means that we need more memory for the hulls (memory grows linearly with r). For $r = 8$, we get a good balance between memory usage and the quality of the shapes.

6.2 Variation with μ

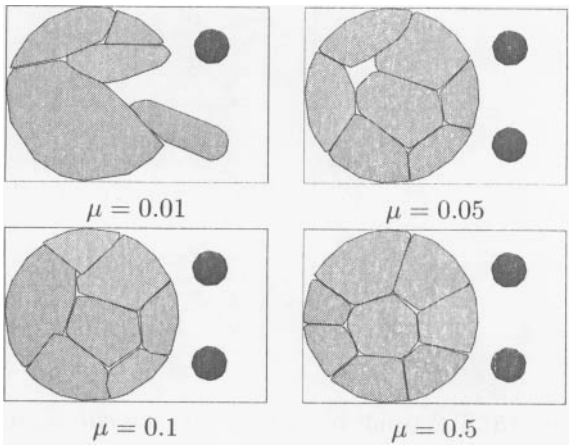


Figure 17: For the *circles* data set, ClusterHull recovers clusters correctly for $\mu \in [.05, .5]$, but fails for $\mu \leq .01$.

We considered values of μ in the range $[.01, .5]$. We fixed $r = 8$, and ran our algorithm for two data sets, *circles* and *stripes*. We fixed the number of hulls, $k = 10$ ($m = 128$) for *circles* and $k = 5$ ($m = 64$) for *stripes*.

If the value of μ is too small, the area dominates the cost function. This causes distant hulls to merge into long skinny hulls spanning multiple clusters. Although the period-doubling cleanup gets rid of most of them by discarding hulls with small densities, the output still contains some hulls spanning multiple natural clusters. Figure 17 shows this effect when $\mu = .01$.

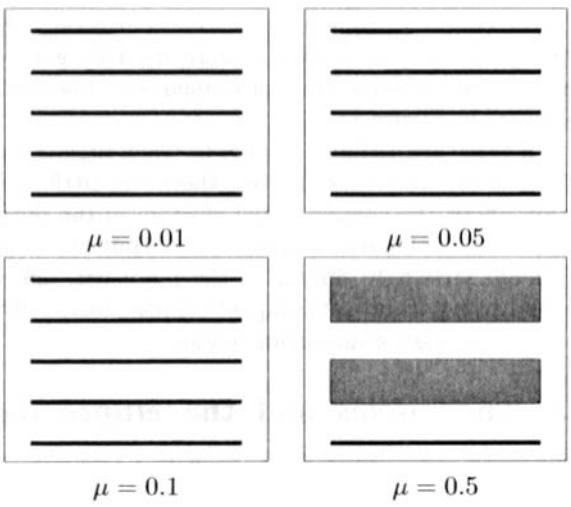


Figure 18: For the *stripes* data set, ClusterHull recovers clusters correctly for $\mu \in [.01, .1]$, but fails for $\mu \geq .5$.

On the other hand, if μ is increased too much, the cost function prefers decreasing the total perimeter, and it is hard to prevent large neighboring clusters from merging together. In Figure 18, neighboring stripes are merged into a single hull for $\mu = .5$. The results show that choosing μ in the range $[.05, .1]$ gives good clusters for most input sets.

7 Conclusion

We developed a novel framework for summarizing the geometric shape and distribution of a two-dimensional point stream. We also proposed an area-based quality measure for such a summary. Unlike existing stream-clustering methods, our scheme adaptively allocates its fixed memory budget to represent different clusters with different degrees of detail. Such an adaptive scheme can be particularly useful when the input has widely varying cluster structures, and the boundary shape, orientation, or volume of those clusters can be important clues in the analysis.

Our scheme uses a simple and natural cost function to control the cluster structure. Experiments show that this cost function performs well across widely different input distributions. The overall framework of ClusterHull is flexible and easily adapted to different applications. For instance, we show that the scheme can be enhanced with period-doubling and incremental

cleanup to deal effectively with noise and extreme data distributions. In those settings, especially when the input has spatial coherence, our scheme performs noticeably better than general-purpose clustering methods like CURE and k-medians.

Because our hulls tend to be more stable than, for instance, the centroids of k-medians, we can maintain other useful data statistics such as *population count* or *density* of individual hulls. (Our hulls grow by merging with other hulls, whereas the centroids in k-medians potentially shift after each new point arrival. The use of incremental cleanup may cause some of our hulls to be discarded, but that happens only for very low-weight, and hence less-interesting hulls.) Thus, the cluster hulls can capture some important frequency statistics, such as which five hulls have the most points, or which hulls have the highest densities, etc.

Although ClusterHull is inspired by the α-hull and built on top of an optimal convex hull structure, the theoretical guarantees of those structures do not extend to give approximation bounds for ClusterHull. Providing a theoretical justification for ClusterHull's practical performance is a challenge for future work.

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In Proc. of the 29th VLDB conference, 2003.

[3] N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* **58** (1999), 137–147.

[4] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60:125–135, 1998.

[5] P. S. Bradley, U. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases. Proc. 4th International Conf. on Knowledge Discovery and Data Mining (KDD-98), 1998.

[6] T. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 152–159, 2004.

[7] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proc. 29th Annu. ACM Symp. Theory Computing*, pages 626–635, 1997.

[8] D. Cheng, R. Kannan, S. Vempala, G. Wang A Divide-and-Merge Methodology for Clustering. In Proc. of the ACM Symposium on Principles of Database Systems, 2005.

[9] G. Cormode and S. Muthukrishnan. Radial histogram for spatial streams. Technical Report 2003-11, DIMACS, 2003.

[10] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH 96*, pages 303–312, 1996.

[11] T. K. Dey, K. Mehlhorn, and E. Ramos. Curve reconstruction: Connecting dots with good reason. *Comp. Geom.: Theory and Appl.*, 15:229–244, 2000.

[12] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In Proc. of the 18th International Conference on Machine Learning, ICML, 2001

[13] P. Domingos, G. Hulten. Learning from Infinite Data in Finite Time. In Proc. Advances in Neural Information Processing Systems (NIPS), 2002

[14] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In Proc. of 10th Symposium on Discrete Algorithms (SODA), 1999.

[15] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[16] M. Ester, H. Kriegel and X. Xu. A database interface for clustering in large spatial databases. In Int. Conference on Knowledge Discovery in Databases and Data Mining (KDD-95), Montreal, Cananda, August 1995.

[17] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models, 2002. Manuscript.

[18] A. Frieze, R. Kannan and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. In Proc. of 39th Symposium on Foundations of Computer Science (FOCS), 1998.

[19] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. *Clustering data streams: Theory and practice.* IEEE Trans. Knowl. Data Engineering, Vol. 15, pages 515–528, 2003.

[20] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In Proc. of International Conference on Management of Data (SIGMOD), 1998.

[21] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. Proc. of SIGMOD, pages 58–66, 2001.

[22] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. In *Proc. 23rd ACM Sympos. Principles Database Syst.*, pages 252–262, 2004.

[23] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[24] S. Muthukrishnan. Data streams: Algorithms and applications. Preprint, 2003.

[25] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In Proceedings of the 20th VLDB Conference, 1994.

[26] J. O'Rourke and G. T. Toussaint. Pattern recognition. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 43, pages 797–813. CRC Press LLC, Boca Raton, FL, 1997.

[27] D. Pelleg, and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In Proc. of the 17th International Conference on Machine Learning (ICML), 2000.

[28] S. Vempala R. Kannan and A. Vetta On clusterings - good, bad and spectral. In Proc. 41st Symposium on the Foundation of Computer Science, FOCS, 2000.

[29] USGS West Nile Virus Maps - 2002. `http://cindi.usgs.gov/hazard/event/west_nile/`.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In Proc. of the International Conference on Management of Data (SIGMOD), 1996.

Distance-Sensitive Bloom Filters

Adam Kirsch * Michael Mitzenmacher †

Abstract

A Bloom filter is a space-efficient data structure that answers set membership queries with some chance of a false positive. We introduce the problem of designing generalizations of Bloom filters designed to answer queries of the form, "Is x close to an element of S?" where closeness is measured under a suitable metric. Such a data structure would have several natural applications in networking and database applications.

We demonstrate how appropriate data structures can be designed using locality-sensitive hash functions as a building block, and we specifically analyze the performance of a natural scheme under the Hamming metric.

1 Introduction

A Bloom filter is a simple, space-efficient, randomized data structure that allows one to answer set membership queries with a small but constant probability of a false positive.[1] Bloom filters have found numerous uses, particularly in distributed databases and networking (see, e.g. [2, 9, 10]). Here we initiate a new direction in the study of Bloom filters by considering *distance-sensitive Bloom filters* that answer approximate set membership queries in the following sense: given a metric space (U, d), a finite set $S \subset U$, and parameters $0 \leq \varepsilon < \delta$, the filter aims to effectively distinguish between inputs $u \in U$ such that $d(u, x) \leq \varepsilon$ for some $x \in S$ and inputs $u \in U$ such that $d(u, x) \geq \delta$ for every $x \in S$. Our constructions allow false positives and false negatives. By comparison, the standard Bloom filter corresponds to the case where $\varepsilon = 0$ and δ is any positive constant, and it only gives false positives.

We establish a framework for constructing distance-sensitive Bloom filters when the metric d admits a locality-sensitive hash family (see, e.g., [4, 5, 7]). The potential benefits of this type of data structure are its speed and space; it can provide a quick answer without performing comparisons against the entire set, or even without performing a full nearest-neighbor query, and it should require less space than the original data. For example, in a distributed setting, client processes might use such a filter to determine whether sending a nearest neighbor query to a server would be worthwhile without actually querying the server; if the filter indicates that there is no sufficiently close neighbor to the query, then the request can be skipped. Of course, in all applications the consequences of and tradeoffs between false positives and false negatives need to be considered carefully.

As an example of a possible application of a distance-sensitive Bloom filter, consider a large database that identifies characteristics of people using a large number of fields. Given a vector of characteristic values, one may want to know if there is a person in the database that matches on a large fraction of the characteristics. One can imagine this being useful for employment databases, where one is seeking a candidate matching a list of attributes. A company may wish to provide a distance-sensitive Bloom filter as a way of advertising the utility of its database while providing very little information about its actual content. Similarly, the data structure may also be useful for criminal identification; it could provide a quick spot-test for whether a suspect in custody matches characteristics of suspects for other unsolved crimes.

As a networking example, the SPIE system for IP traceback represents packet headers seen over a small interval of time by a Bloom filter [16]. The authors of that work found that packet headers usually remain consistent as packets travel among routers. A distance-sensitive Bloom filter might allow small changes in the packet header bits while still allowing the Bloom filter to answer queries appropriately.

As a very general example, suppose that we have a collection of sets, with each set being represented by a Bloom filter. For example, a Bloom filter might represent packet traces as above, or a sketch of a document as in [3]. The relative Hamming distance between two Bloom filters (of the same size, and created with the same hash functions) can be used as a measure

*Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. Supported in part by an NSF Graduate Research Fellowship and NSF grants CCR-9983832 and CCR-0121154. Email: kirsch@eecs.harvard.edu

†Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. Supported in part by NSF grants CCR-9983832 and CCR-0121154. Email: michaelm@eecs.harvard.edu

[1] We assume some familiarity with Bloom filters throughout the paper; see [2] for background.

of the similarity of the underlying sets (see, e.g., [2]). Hence, one can construct a distance-sensitive Bloom filter on top of such a collection of Bloom filters to attempt to quickly and easily answer questions of the form, "Are there any sets in the collection very close to this query set?" Such a general construction may prove useful for other distributed applications where Bloom filters are used. For this reason, we pay special attention to the case where $U = \{0,1\}^\ell$ and d is the relative Hamming metric on U.

A more distant but potentially very exciting application of distance-sensitive Bloom filters is in the context of DNA sequences. One might hope that such a filter could effectively handle queries when d is chosen to be the edit distance metric, in order to answer questions of the form, "Is there a DNA sequence close to this one in your database?" Unfortunately, edit distance currently appears to be too difficult to adequately handle in our framework, as there is no known good locality-sensitive hash function for edit distance, although there is recent work attempting to connect edit distance and Hamming distance via various reductions [7, 11]. Similar problems arise in computer virus detection, and ad hoc variations of Bloom filters have recently been used in this setting [15].

Although there are many potential applications, this problem does not appear to have been the subject of much study. Manber and Wu [8] considered the problem of handling a single character error using Bloom filters in the context of password security. Work on nearest-neighbor problems (including [4, 5, 7]) is clearly related, but the problem setting is not equivalent. Our work also seems similar in spirit to work on property testing [14, 6], and specifically to the recently introduced notion of tolerant property testing [13], although here the task is to design a *structure* based on an input set that will allow quick subsequent testing of closeness to that set, instead of an algorithm to test closeness of an object to some property.

Our main contributions in this paper are therefore

1. introducing the formal problem of developing Bloom filter variants that effectively determine whether queries are *close* to an item in a particular set,

2. developing the connection between this problem and locality-sensitive hashing, and

3. examining in detail the case where $U = \Sigma^\ell$ and d is the relative Hamming metric.

Our initial results are not as strong as one might hope. For example, when $U = \{0,1\}^\ell$, d is the relative Hamming metric on U, and $|S| = n$, our distance-sensitive Bloom filter is only efficient and effective for constant δ when $\varepsilon = O(1/\log n)$. That is, we can only differentiate between query strings that differ from all strings in S on a (constant) δ-fraction of bits and query strings that share a $1-\varepsilon = 1 - O(1/\log n)$-fraction of bits with some string in S. We would prefer ε to be constant. Nevertheless, our experiments suggest that even with this limitation distance-sensitive Bloom filters work sufficiently well to be useful in practice.

Our work leaves many additional open problems. Indeed, one obvious question is whether Bloom filters provide the appropriate paradigm for this problem; alternatives to standard Bloom filters have recently received study [12]. Our major reasons for initiating our study with Bloom filters are because they provide a natural theoretical framework and because Bloom filters are already widely accepted, understood, and used by practitioners.

2 A General Approach

This section gives a general approach to designing distance-sensitive Bloom filters for metric spaces (U,d) that admit a locality-sensitive hash family [5].

DEFINITION 2.1. *A family* $\mathcal{H} = \{h : U \to V\}$ *is* (r_1, r_2, p_1, p_2)-sensitive *with respect to a metric space* (U,d) *if* $r_1 < r_2$, $p_1 > p_2$, *and for any* $x, y \in U$,

- *if* $d(x,y) \le r_1$ *then* $\Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \ge p_1$, *and*
- *if* $d(x,y) > r_2$ *then* $\Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \le p_2$.

We say that any such family is a (U,d)-locality-sensitive hash (LSH) family, *omitting* (U,d) *when the meaning is clear.*

It turns out that our approach is more effectively expressed when we generalize Definition 2.1.

DEFINITION 2.2. *Let* (U,d) *be a metric space, and let* $p_L : \mathbb{R}_{\ge 0} \to [0,1]$ *and* $p_H : \mathbb{R}_{\ge 0} \to [0,1]$ *be non-increasing. A hash family* $\mathcal{H} : U \to V$ *is called* (p_L, p_H)-distance sensitive *(with respect to* (U,d)*) if for all* $x, y \in U$

$$p_L(d(x,y)) \le \Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \le p_H(d(x,y)).$$

We note that Definition 2.2 really does generalize Definition 2.1, since for any (r_1, r_2, p_1, p_2)-locality-sensitive hash family $\mathcal{H}$, we may set

$$p_L(r) = \begin{cases} p_1 & \text{if } r \le r_1 \\ 0 & \text{otherwise} \end{cases}$$

and

$$p_H(r) = \begin{cases} p_2 & \text{if } r > r_2 \\ 1 & \text{otherwise} \end{cases}$$

and get that $\mathcal{H}$ is a (p_L, p_H)-distance-sensitive hash family.

We are now ready to present our first and most general distance-sensitive Bloom filter construction, which is essentially a standard partitioned Bloom filter where the random hash functions are replaced by distance-sensitive hash functions. Let $\mathcal{H} : U \to V$ be a (p_L, p_H)-distance sensitive hash function, fix some $S \subset U$ with n elements, and let A be an array consisting of k disjoint m'-bit arrays, $A[1,1], \ldots, A[k,m']$ (for a total of $m = km'$ bits), where k and m' are parameters. If $V \neq [m']$, then let $\mathcal{H}' : V \to [m']$ be a weakly pairwise-independent hash family.

To initialize the filter, we first choose $h_1, \ldots, h_k \leftarrow \mathcal{H}$ independently. If $V \neq [m']$, then we also choose $h'_1, \ldots, h'_k \leftarrow \mathcal{H}'$ independently, and define $g_i = h'_i \circ h_i$ for $i \in [k]$. If $V = [m']$, then we define $g_i = h_i$ for $i \in [k]$. Next, we set all of the bits in A to zero. Finally, for $x \in S$ and $i \in [k]$, we set $A[i, g_i(x)] = 1$.

It remains to specify how to use the filter to effectively determine whether a query $u \in U$ is close to some $x \in S$. In a standard Bloom filter, to answer queries of the form, "Is $u \in S$?" we simply check if all of u's hash locations in A (that is, the set of bits $\{A[i, h_i(u)] : i \in [k]\}$) are set to 1; we return "$u \in S$" if this is the case and we return "$u \notin S$" otherwise. In our setting, we must consider that u might not be in S but still be close to some element of S, in which case it might be that not all of u's hash locations are set to 1. We consider instead the set of u's hash locations that are set to 1; specifically, by symmetry, it suffices to consider the number $B(u)$ of u's hash locations that are set to 1.

Now, it is easy to see that $B(u) = \sum_{i \in [k]} A[i, g_i(u)]$. As the $A[i, g_i(u)]$'s are independent and identically distributed bits, $B(u) \sim \text{Bin}(k, q_u)$ for some $q_u \in [0,1]$ (where $\text{Bin}(t, r)$ denotes the binomial distribution with t trials and common success probability r). We derive upper and lower bounds on q_u.

PROPOSITION 2.1. *In an abuse of notation, we define $d(u,S) = \min_{x \in S} d(u,x)$ for $u \in U$. Let $\mathbf{1}(\cdot)$ denote the indicator function. For any $u \in U$,*

$$p_L(d(u,S)) \leq q_u \leq \sum_{x \in S} p_H(d(x,u)) + \frac{nk}{m} \cdot \mathbf{1}(V \neq [m'])$$

Proof. For the lower bound, we write

$$\begin{aligned} q_u &= \Pr(A[1, g_1(u) = 1]) \\ &= \Pr(\exists x \in S : g_1(x) = g_1(u)) \\ &\geq \max_{x \in S} \Pr(g_1(x) = g_1(u)) \\ &\geq \max_{x \in S} p_L(d(x,u)) \\ &= p_L(d(u,S)), \end{aligned}$$

where the last step follows from the fact that p_L is non-increasing.

For the upper bound, we write

$$\begin{aligned} q_u &= \Pr(A[1, g_1(u) = 1]) \\ &= \Pr(\exists x \in S : g_1(x) = g_1(u)) \\ &\leq \sum_{x \in S} \Pr(g_1(x) = g_1(u)). \end{aligned}$$

Now, if $V = [m']$, then for any $x \in S$,

$$\Pr(g_1(x) = g_1(u)) = \Pr(h_1(x) = h_1(u)) \leq p_H(d(x,u)),$$

and if $V \neq [m']$, then for any $x \in S$,

$$\begin{aligned} &\Pr(g_1(x) = g_1(u)) \\ &\leq \Pr(h_1(x) = h_1(u)) \\ &\quad + \Pr(h'_1(h_1(x)) = h'_1(h_1(u)) \mid h_1(x) \neq h_1(u)) \\ &\leq p_H(d(x,u)) \\ &\quad + \Pr(h'_1(h_1(x)) = h'_1(h_1(u)) \mid h_1(x) \neq h_1(u)) \\ &\leq p_H(d(x,u)) + \frac{1}{m'}, \end{aligned}$$

so

$$\begin{aligned} q_u &\leq \sum_{x \in S} \Pr(g_1(x) = g_1(u)) \\ &\leq \sum_{x \in S} \left(p_H(d(x,u)) + \frac{1}{m'} \cdot \mathbf{1}(V \neq m') \right) \\ &= \sum_{x \in S} p_H(d(x,u)) + \frac{nk}{m} \cdot \mathbf{1}(V \neq m'). \end{aligned}$$

□

For real-valued random variables X and Y, we write $X \leq_{\text{st}} Y$ or if Y stochastically dominates X. That is $X \leq_{\text{st}} Y$ if for all $x \in \mathbb{R}$, $\Pr(X \geq x) \leq \Pr(Y \geq x)$. The following corollary now follows readily from Proposition 2.1.

COROLLARY 2.1. *For any $u \in U$,*

$$\begin{aligned} &\text{Bin}(k, p_L(d(u,S))) \\ &\leq_{\text{st}} B(u) \\ &\leq_{\text{st}} \text{Bin}\left(k, \sum_{x \in S} p_H(d(x,u)) + \frac{nk}{m} \cdot \mathbf{1}(V \neq [m'])\right). \end{aligned}$$

Corollary 2.1 suggests that if the filter is configured properly then whenever $u \in U$ is particularly close to some $x \in S$ and $v \in U$ is particularly far from every $x \in S$, we should have that q_u is substantially larger than q_v. Since binomial distributions are reasonably well-concentrated around their expectations, this intuition suggests the existence of some threshold t (that does not depend on the particular strings in S) such that $B(u)$ is unlikely to be below t and $B(v)$ is unlikely to be above t. It follows that for any $w \in U$ that is either particularly close to some $x \in S$ or particularly far from every $x \in S$, we can effectively determine which of these two cases applies by comparing $B(w)$ and t.

As an example, we consider $t = k$, which corresponds to the technique used by a standard Bloom filter. More specifically, we suppose that when the filter is queried with parameters $0 \leq \varepsilon < \delta$ and $u \in U$, it responds that "$d(u, S) \leq \varepsilon$" if all of u's hash locations are 1 and "$d(u, S) \geq \delta$" otherwise. For this scheme, Corollary 2.1 immediately tells us that

- if $d(u, S) \leq \varepsilon$, then Pr(the filter is incorrect) $\leq 1 - p_L(\varepsilon)^k$, and
- if $d(u, S) \geq \delta$, then Pr(the filter is incorrect) $\leq \left(n\left[p_H(d(\delta)) + \frac{k}{m} \cdot \mathbf{1}\left(V \neq [m']\right)\right]\right)^k$.

The setting of $t = k$ is simply an example. For any real application, it is important to assess the relative severity of false positives and false negatives and experimentally determine the value of t that optimizes the tradeoff between the observed frequencies of the two types of errors.

Before continuing, we note that the bound for the false positive probability is unfortunately rather weak in this general setting, due to the fact that the probability that a particular hash location is set to 1 (from the occurrence of the event $\{\exists x \in S : h_i(x) = h_i(u)\}$ for some fixed $i \in [k]$ and $u \in U$ with $d(u, S) \geq \delta$) is bounded only by $np_H(\delta)$; this requires that $p_H(\delta)$ is certainly $O(1/n)$ and preferably $o(1/n)$. In practice, this weakness may be avoided, although this depends on the set S (and the underlying metric); in particular, $\max_u \sum_{x \in S} p_H(d(x, u))$ will often be smaller than $np_H(\delta)$, and one may be able to bound $\max_u \sum_{x \in S} p_H(d(x, u))$ easily given S. Alternatively, the bound $np_H(\delta)$ may be reasonably tight, such as when $p_H(\delta)$ is small and collisions between elements of S are unlikely. Similar issues arise in nearest-neighbor problems [1]; this issue clearly warrants further study.

3 The Relative Hamming Metric on $U = \Sigma^\ell$

To give more insight and make the problem more concrete, we now focus on the special case where $U = \Sigma^\ell$ for some ℓ and alphabet Σ, and d is the relative Hamming metric on U (that is, $d(u, v) = \sum_{i=1}^{\ell} \mathbf{1}(u_i \neq v_i)/\ell$, where $\mathbf{1}(\cdot)$ is the indicator function and for $u \in U$ and $i \in [\ell]$, we let u_i denote the ith character of u). Without loss of generality, we suppose that $\Sigma = \{0, \ldots, r-1\}$ for some $r \geq 2$. An obvious alternative approach in this situation would be to simply choose s character locations (independently and uniformly at random) and for each string in S use these s characters as a sketch for the string. To check if any string in S has distance at most ε from some input x the sketches can be checked to see if any match in (slightly less than) a $1 - \varepsilon$ fraction of locations. Such schemes require checking at least $\Omega(n)$ characters to find a potential match; while this may be suitable for some applications, we aim (in the spirit of standard Bloom filter constructions) to use only a constant number of character lookups into the data structure, making this approach unsuitable (although it may also be useful in practice).

We first give a general analysis of a distance-sensitive Bloom filter and show that this analysis does not yield performance tradeoffs nearly as good as a standard Bloom filter. Then we show that by limiting our goals appropriately, our analysis yields results that are suitable in practice for the important case where $r = 2$. Finally, we present the results of simple experiments that demonstrate the potential practicability of the scheme.

Recall that, given parameters $0 \leq \varepsilon < \delta \leq 1$, our goal is to effectively distinguish between the strings $u \in U$ where $d(u, S) \leq \varepsilon$, which we call *ε-close to S*, and those where $d(u, S) \geq \delta$, which we call *δ-far from S*.

We define

$$c = \begin{cases} 1 & \text{if } r = 2 \\ 1/2 & \text{if } r > 2 \end{cases}$$

$$n = |S|$$

$$\ell' = \left\lceil \log_{\frac{1-c\varepsilon}{1-c\delta}} 4n \right\rceil$$

$$m' = 2^{\ell'}$$

$$m = km'$$

$$t = k(1 - c\varepsilon)^{\ell'}/2.$$

Here n is the number of items, m is the total size of the filter (in bits), $k \geq 1$ is the number of hash functions, and ℓ' is the number of locations read per hash function. If $r = 2$, the bits yield a location in the hash table in the natural way. Specifically, we define the hash family $\mathcal{H} : U \to [m']$ in the case where $r = 2$ as

follows: we choose $h \leftarrow \mathcal{H}$ by choosing $i_1, \ldots, i_{\ell'} \leftarrow [\ell]$ uniformly and independently, and then defining $h(u) = u_{i_1} \cdots u_{i_{\ell'}} + 1$ (where we are considering $u_{i_1} \cdots u_{i_{\ell'}}$ as a number in binary). If $r > 2$, we do essentially the same thing, but with an added level of pairwise independent hashing from Σ to $\{0,1\}$. More precisely, if $r > 2$, we let $\mathcal{H}' : \Sigma \to \{0,1\}$ be a pairwise independent hash family, and we define the hash family $\mathcal{H} : U \to [m']$ as follows: we choose $h \leftarrow \mathcal{H}$ by choosing $i_1, \ldots, i_{\ell'} \leftarrow [\ell]$ uniformly and independently, $h'_1, \ldots, h'_{\ell'} \leftarrow \mathcal{H}'$ independently, and then defining $h(u) = h'_1(u_{i_1}) \cdots h'_{\ell'}(u_{i_{\ell'}}) + 1$. Using these definitions, we construct the filter described in Section 2.

It is easy to verify that $\mathcal{H}$ is a (p_L, p_H)-distance sensitive hash function for $p_L(z) = p_H(z) = (1 - cz)^{\ell'}$. (Indeed, if $r = 2$, then $\mathcal{H}$ is a canonical example of a locality-sensitive hash family [5].) Proposition 2.1 now immediately yields the following result.

COROLLARY 3.1. *Consider some $u \in U$.*

- *If $d(u, S) \leq \varepsilon$, then $q_u \geq (1 - c\varepsilon)^{\ell'}$.*
- *If $d(u, S) \geq \delta$, then $q_u \leq n(1 - c\delta)^{\ell'}$.*

In Section 2, we gave some intuition as to why results of the above form should allow for a properly configured filter to effectively distinguish between those $u \in U$ that are ε-close to S and those $u \in U$ that are δ-far from S. Theorem 3.1 formalizes that intuition.

THEOREM 3.1. *When $t = k(1 - c\varepsilon)^{\ell'}/2$, then for any fixed $u \in U$, over the random choices made constructing the filter:*

- *If $d(u, S) \leq \varepsilon$, then*

$$\begin{aligned}\Pr(B(u) < t) &< \exp\left[-2t^2/k\right] \\ &= \exp\left[-k(1 - c\varepsilon)^{2\ell'}/2\right].\end{aligned}$$

- *If $d(u, S) \geq \delta$, then*

$$\begin{aligned}\Pr(B(u) \geq t) &\leq \exp\left[-t^2/2k\right] \\ &= \exp\left[-k(1 - c\varepsilon)^{2\ell'}/8\right].\end{aligned}$$

REMARK 3.1. While the correct choice of the threshold t in practice may depend on the relative importance of false positive and false negatives, the choice of $t = k(1 - c\varepsilon)^{\ell'}/2$ presented here allows provable statements that give insight into the performance tradeoffs involved in designing the filter; specifically, the bounds are the same order of magnitude in the exponent, and allow for the asymptotic analyses in Sections 3.1 and 3.2.

Proof. For the first inequality, we have $d(u, S) \leq \varepsilon$. Then

$$t - \mathbf{E}[B(u)] = t - kq_u \leq t - k(1 - c\varepsilon)^{\ell'} = -t$$

by Corollary 3.1. Therefore,

$$\begin{aligned}\Pr(B(u) < t) &= \Pr(B(u) - \mathbf{E}[B(u)] < t - \mathbf{E}[B(u)]) \\ &\leq \Pr(B(u) - \mathbf{E}[B(u)] < -t) \\ &< \exp\left[-2t^2/k\right],\end{aligned}$$

by the Azuma-Hoeffding inequality.

For the second inequality, if $d(u, S) \geq \delta$, then

$$\begin{aligned}t - \mathbf{E}[B(u)] &= t - kq_u \\ &\geq t - kn(1 - c\delta)^{\ell'} \\ &= k(1 - c\varepsilon)^{\ell'}\left[\frac{1}{2} - n\left(\frac{1 - c\delta}{1 - c\varepsilon}\right)^{\ell'}\right] \\ &\geq k(1 - c\varepsilon)^{\ell'}\left[\frac{1}{2} - \frac{1}{4}\right] \\ &= t/2,\end{aligned}$$

where the second step follows from Corollary 3.1, the fifth step follows from the fact that $\ell' \geq \log_{\frac{1-c\varepsilon}{1-c\delta}} 4n$, and the other steps are obvious. Therefore,

$$\begin{aligned}\Pr(B(u) \geq t) &= \Pr(B(u) - \mathbf{E}[B(u)] \geq t - \mathbf{E}[B(u)]) \\ &\leq \Pr(B(u) - \mathbf{E}[B(u)] \geq t/2) \\ &\leq \exp\left[-2(t/2)^2/k\right] = \exp\left[-t^2/2k\right],\end{aligned}$$

by the Azuma-Hoeffding inequality. □

3.1 Asymptotic Analysis: The Bad News. We now consider what Theorem 3.1 suggests about the performance of our basic distance-sensitive Bloom filter construction. While the experiments in Section 4 show that our construction is likely to be useful in many practical situations, it does not scale well to very large numbers of items. By comparison, we recall that standard Bloom filters have the following three very desirable properties when properly configured:

1. They use $O(n)$ bits (and the hidden constant is small).
2. They guarantee a small constant error probability (asymptotically and in practice).
3. To answer a query, one must only read a small constant number of bits in the array.

Our construction does not appear to meet these goals for constant (with respect to n) values of the

parameters ε and δ. In fact, it does not even seem to meet the last two goals for constant ε and δ. For example, for $r = 2$, if we take $k = O(1)$ and $\varepsilon = \Omega(1)$, then the bounds in Theorem 3.1 yield constant error probabilities only if $\ell' = O(1)$, in which case

$$\log_{\frac{1-\varepsilon}{1-\delta}} 4n = O(1) \quad \Longrightarrow \quad \delta = 1 - \frac{1}{n^{\Omega(1)}}.$$

Similarly, if $k = O(1)$ and $\delta = 1 - \Omega(1)$, then the bounds in Theorem 3.1 give constant error probabilities only if $(1-\varepsilon)^{\ell'} = \Omega(1)$, implying that

$$\left\lceil \log_{\frac{1-\varepsilon}{1-\delta}} 4n \right\rceil = \ell' = O\left(\frac{1}{\log \frac{1}{1-\varepsilon}}\right),$$

which cannot hold for constant ε. Therefore, the only way that Theorem 3.1 allows us to achieve the desired goals is if we restrict our attention to cases where the gap between ε and δ is extremely large for sufficiently large n.

3.2 Asymptotic Analysis: The Good News. If we loosen the desired properties for a distance-sensitive Bloom filter, we can still obtain results that appear useful in practice. Specifically, in the case of the relative Hamming metric, we note that the total length of the strings in the original set is at least $n\ell \log_2 r$ bits. Therefore, we should seek that the total length of the filter m is much less than $n\ell$, and not be concerned if $m = \omega(n)$ so long as this condition holds. Furthermore, and more importantly, we consider cases where δ is constant but ε is only $O(1/\log n)$. That is, we only seek to differentiate between query strings that differ from all strings in S on a (constant) δ-fraction of bits and query strings that share a $1 - \varepsilon = 1 - O(1/\log n)$-fraction of bits with some string in S. This restriction clearly limits the scalability of the construction. However, it still allows very useful results for n in the thousands or tens of thousands. (Larger n can be handled, but only with quite small values of ε or quite large values of ℓ.) In fact, as our experiments in Section 4 show, the asymptotic restriction that $\varepsilon = O(1/\log n)$ still allows for excellent performance with reasonable values of n, ℓ, and δ. And if a particular application allows for an even smaller value of ε, say $\varepsilon = c'/n$ for some constant c', then the performance for reasonable values of n and ℓ only improves, although the gap between ε and a reasonable value of δ may be quite large.

For convenience we focus on the binary case where $r = 2$, so $U = \{0,1\}^{\ell}$. In this case, $k\ell'$, the total number of sampled characters, is still logarithmic in n. The space used in the filter is

$$m = k2^{\ell'} = O\left(n^{1/\log_2 \frac{1-\varepsilon}{1-\delta}}\right).$$

For $\delta < 1/2$ (which is the normal case, as even random bit strings will agree on roughly 1/2 of the entries) and $\varepsilon = O(1/\log n)$, we have that $m = \omega(n)$. However, in many cases we can still configure the filter with reasonable parameters so that $m \ll n\ell$. To gain some insight (that will guide our experiments) it is worth considering some sample cases. For $n = 1000$, $\varepsilon = 0.1$, and $\delta = 0.4$, we find $\ell' = 21$. Hence for $k \leq 32$ the number of bits required is less than 2^{26}, giving an improvement over the total number of bits $n\ell$ whenever $\ell \geq 2^{16}$ bits or 8 Kilobytes. Similarly, for $n = 10000$, $\varepsilon = 0.05$, and $\delta = 0.4$, we find $\ell' = 24$, and again for $k \leq 32$ there is a space savings whenever $\ell \geq 2^{16}$.

Formally, we have the following asymptotic relationship, which shows that ℓ need only grow polynomially with n to have $m = o(n\ell)$. (Little effort has been made to optimize the constants below.)

PROPOSITION 3.1. *For any constant δ and $r \geq 2$, if $\log_n \ell = (4 - 6c\delta)/c\delta + \Omega(1/\log n)$, then we may choose $\varepsilon = \Omega(1/\log n)$ and have $m = o(n\ell)$.*

Proof. Let $\gamma = (\log_n \ell)/2$. Since $\ell \geq \log_r n$, we have that

$$\ell = \omega(1) = 2^{\omega(1)} = 2^{(\log_2 n)\omega(1/\log n)} = n^{\omega(1/\log n)},$$

so $\gamma = \omega(1/\log n)$. Therefore, we have that $m = o(n\ell)$ if

$$\frac{1}{\log_2 \frac{1-c\varepsilon}{1-c\delta}} + \gamma \leq \log_n n\ell$$

for sufficiently large n, since in that case

$$m/n\ell = O\left(n^{-\gamma}\right) = O\left(2^{-\gamma \log_2 n}\right) = O\left(2^{-\omega(1)}\right) = o(1).$$

Solving the above inequality for $c\varepsilon$ gives

$$c\varepsilon \leq 1 - (1 - c\delta)2^{1/\log_n n^{1-\gamma}\ell}.$$

Therefore, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$1 - (1 - c\delta)2^{1/\log_n n^{1-\gamma}\ell} \geq \Theta(1/\log n).$$

Equivalently, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$2^{1/\log_n n^{1-\gamma}\ell} \leq \frac{1 - \Theta(1/\log n)}{1 - c\delta}$$

Since $\ell = n^{2\gamma}$, we have that $\log_n n^{1-\gamma}\ell = 1 + \gamma$, and

$$\begin{aligned} 2^{1/\log_n n^{1-\gamma}\ell} &= \exp\left[\frac{\ln 2}{\log_n n^{1-\gamma}\ell}\right] \\ &= \exp\left[\frac{\ln 2}{1+\gamma}\right] \\ &\leq \exp\left[\frac{1}{1+\gamma}\right] \\ &\leq 1 + \frac{2}{1+\gamma}, \end{aligned}$$

where we have used the facts that $\gamma > 0$ and $e^x \leq 1+2x$ for $x \in [0,1]$. Therefore, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$1 + \frac{2}{1+\gamma} \leq \frac{1 - \Theta(1/\log n)}{1 - c\delta},$$

or, equivalently,

$$\begin{aligned} 1 + \gamma &\geq 2(1 - c\delta)\left(\frac{1}{c\delta - \Theta(1/\log n)}\right) \\ &= \frac{2(1-c\delta)}{c\delta}\left(\frac{1}{1 - \Theta(1/\log n)}\right) \\ &= \frac{2(1-c\delta)}{c\delta}(1 + \Theta(1/\log n)) \\ &= \frac{2(1-c\delta)}{c\delta} + \Theta(1/\log n), \end{aligned}$$

where we have used the Taylor series for $1/(1+x)$ (for $|x| < 1$). Therefore, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$\gamma \geq \frac{2 - 3c\delta}{c\delta} + \Theta(1/\log n),$$

which holds by our assumption on $\log_n \ell = 2\gamma$. □

Proposition 3.1 tells us that distance-sensitive Bloom filters can be very space efficient. Unfortunately, for certain reasonable settings of n, ε, and δ, the length ℓ of the strings may need to be very large in order for the filter to require less space than the natural encoding of the set, particularly when $r > 2$ (so $c = 1/2$). (And in these cases, one might be willing to sacrifice very fast query times to reduce the storage requirement using, for instance, the sketching approach mentioned in Section 3.) For example, consider the case where the characters in the alphabet Σ are bytes, so $r = 256$. Then if, as before, $n = 1000$, $\varepsilon = 0.1$, and $\delta = 0.4$, we now have $m' = 2^{49}$. Therefore, even if $k = 1$ (which is almost certainly too small to achieve good performance), $m \leq n\ell \log_2 r$ only if $\ell > 7 \times 10^{10}$. We are unaware of any current applications where ℓ can be this large, although there may be future applications of distance-sensitive Bloom filters to extremely long strings, such as DNA analysis. Thus, while distance-sensitive Bloom filters work reasonably well for binary strings, there is much room for improvement in dealing with larger alphabets.

4 A Simple Experiment

We present experimental results demonstrating the behavior of the basic distance-sensitive Bloom filter of Section 3 in the special case where $r = 2$. In our experiments, we populate our data set S with uniform random binary strings of 65,536 characters, and test whether we can distinguish strings that are near an element of S from strings far from every element of S (under the Hamming metric). In part, this test is chosen because it represents a worthwhile test case; surely we would hope that any worthwhile scheme would perform reasonably on random inputs. In application terms, for the setting described in Section 1 of a distance-sensitive Bloom filter taken over a collection of other Bloom filters, this random experiment roughly corresponds to sets of the same size with no overlapping elements; similar results would hold even if small overlaps between sets were allowed.

We consider sets of sizes $n = 1000$ and $n = 10000$ with $\ell = 65536$. For $n = 1000$, we take $\varepsilon = 0.1$ and $\delta = 0.4$, and for $n = 10000$, we take $\varepsilon = 0.05$ and $\delta = 0.4$. For each value of n and $k = \{1, \ldots, 25\}$, we repeat the following experiment 10 times. Generate 50000 independent *close* queries by randomly selecting a string in S and flipping an ε-fraction of its bits, and testing whether the filter returns a false negative. Then generate 50000 independent *far* queries by randomly selecting a string in S and flipping a δ-fraction of its bits, and testing whether the filter returns a false positive (here, the implicit assumption is that the query string is δ-far from *all* strings in S, not just the one chosen; this holds with very high probability). We compute observed false positive and false negative rates by averaging our results over all queries.

The results are given in Figures 1 and Figure 2, with a summary in Table 1. In the figures, the x-axis is given in terms of the number m of bits used, but recall that $m = km'$, so it also reflects the number k of hash functions. The abrupt jumps in the false positive rate correspond to values of k where the threshold value $\lfloor t \rfloor$ increases. At such a point, it naturally becomes much harder for a false positive to occur, and easier for a false negative to occur.

As the figures and table indicate, we successfully use fewer than $n\ell$ bits and require only a constant number of bit lookups into the data structure (the array A). The observed false positive and negative rates are quite reasonable; in particular, the false negative rate falls rapidly, which is good as false negatives are often very damaging for applications.

We emphasize that although the experimental results appear to give modest size improvements, this is because we have chosen a small value for the item size of ℓ bits. For larger values of ℓ, the results would be entirely the same, except that the ratio $m/n\ell$ would shrink further.

Before concluding, we point out a few interesting characteristics of these experiments. First, recall that these tests roughly correspond to the application described in Section 1 where a distance-sensitive Bloom

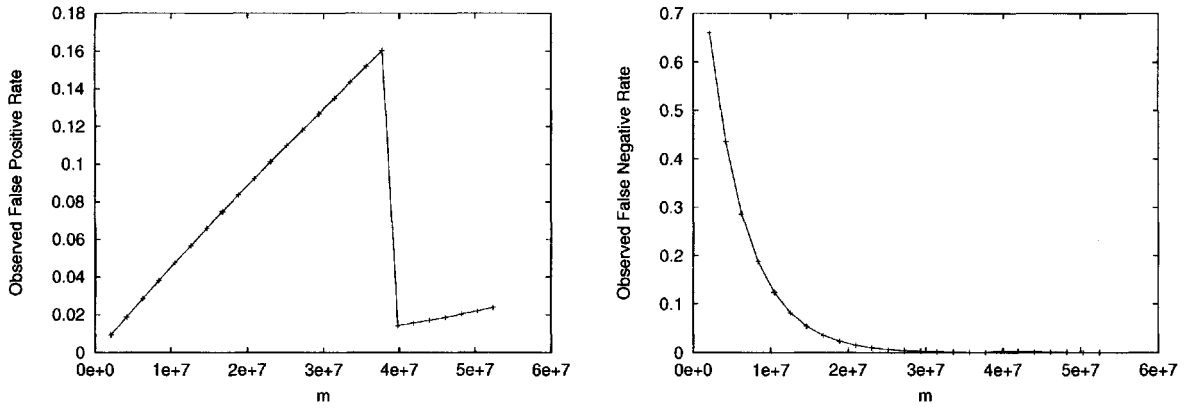


Figure 1: The observed false positive and false negative rates for $n = 1000$, $\ell = 65536$, $\varepsilon = 0.1$, and $\delta = 0.4$.

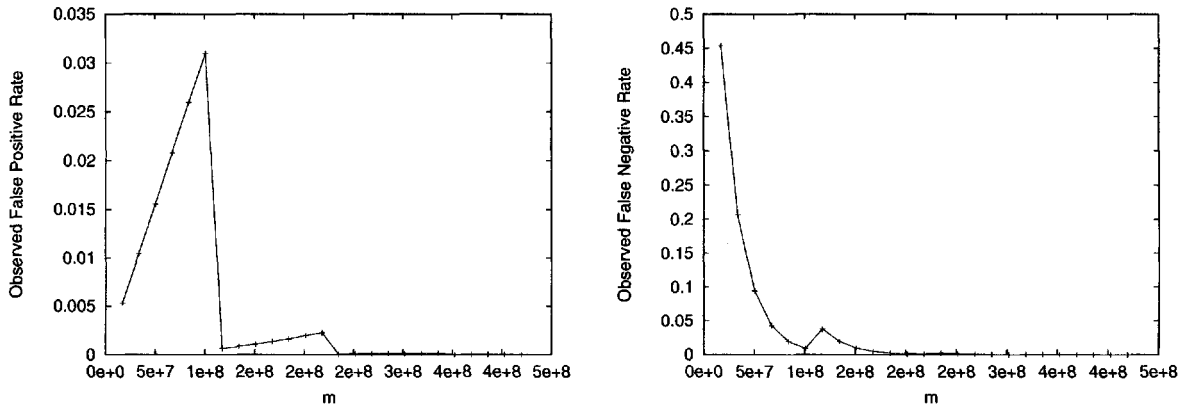


Figure 2: The observed false positive and false negative rates for $n = 10000$, $\ell = 65536$, $\varepsilon = 0.05$, and $\delta = 0.4$.

(a) $n = 1000$, $\ell = 65536$, $\varepsilon = 0.1$, and $\delta = 0.4$.

k	$k\ell'$	fp rate	fn rate	$m/n\ell$
5	105	0.04744	0.124236	0.16
10	210	0.09235	0.015366	0.32
15	315	0.134926	0.001934	0.48
20	420	0.01572	0.002816	0.64
25	525	0.023874	0.000372	0.8

(b) $n = 10000$, $\ell = 65536$, $\varepsilon = 0.05$, and $\delta = 0.4$.

k	$k\ell'$	fp rate	fn rate	$m/n\ell$
5	120	0.025958	0.019746	0.128
10	240	0.001338	0.00495	0.256
15	360	0.000068	0.00125	0.384
20	480	0.000158	0.000034	0.512
25	600	0.000006	0.000012	0.64

Table 1: The number $k\ell'$ of sampled bits, the observed false positive and false negative rates, and the corresponding storage requirements, for various values of k.

filter is formed for a collection of Bloom filters with no shared elements. In certain instances of this application, we might be justified in choosing a very small ε. For example, if the Bloom filters each represent sets of $n' = c_1 n$ elements and have size $c_2 n'$, and $\varepsilon = c_3/n$, where c_1, c_2, and c_3 are reasonable constants, then a query Bloom filter is ε-close to the set of Bloom filters if (roughly speaking) the set underlying the query filter shares all but a constant number of items with one of the other sets. While this threshold is certainly low, it may be suitable for some applications, and for such applications a properly configured distance-sensitive Bloom filter is likely to be extremely successful.

On a more general note, these experiments have the nice property that the set S is uniformly sampled from $\{0,1\}^{\ell}$. Thus, it is unlikely that any close query is ε-close to any element of S other than the one used to generate the query. Furthermore, for a far query, the events corresponding to hash collisions between the query and the elements of S are independent and, since ℓ' is reasonably sized, each occur with small probability. By looking back at the proof of Proposition 2.1, it follows that the bounds in Corollary 3.1 are fairly tight. Therefore, while those results may be very weak for certain data sets, it is impossible to substantially improve them without taking into account specific properties of S.

5 Conclusions and Further Work

We strongly believe that efficient distance-sensitive Bloom filters will find significant use in many applications. Our initial work suggests that distance-sensitive Bloom filters can be constructed with parameters suitable in practice, but further improvements appear possible.

We suggest a number of open questions for further work:

- Is there a general approach that would allow constant $\varepsilon, \delta > 0$, a linear number of bits, a constant number of hash functions, and constant false positive/false negative probabilities?
- Are there simple and natural conditions one can place on a set S for natural metrics that would yield stronger bounds?
- There are very natural information-theoretic bounds on the number of bits required for Bloom-filter-like structures. Are there equivalent bounds in this setting? (Perhaps one must fix a metric, such as the relative Hamming metric.)
- Can closeness in edit distance be handled using data structures of this type?

References

[1] A. Andoni, M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing using stable distributions. To appear in *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*, MIT Press.

[2] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485-509, 2004.

[3] N. Jain, M. Dahlin, and R. Tewari. Using Bloom filters to refine web search results. In *Proc. of the Eighth International Workshop on the Web and Databases (WebDB2005)*, 2005.

[4] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of the 20th ACM Symposium on Computational Geometry*, pp. 253-262, 2004.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proc. of the 25th International Conference on Very Large Data Bases*, pp. 518-529, 1999.

[6] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653-750, 1998.

[7] P. Indyk. Approximate nearest neighbor under edit distance via product metrics. In *Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 646-650, 2004.

[8] U. Manber and S. Wu. An algorithm for approximate membership checking with applications to password security. *Information Processing Letters*, 50(4):191-197, 1994.

[9] M. Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Transactions on Networking*, 10(5):613-620, 2002.

[10] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[11] R. Ostrovsky and Y. Rabani. Low distortion embeddings for edit distance. In *Proc. of the 37th Annual ACM Symposium on Theory of Computing*, 218-224, 2005.

[12] A. Pagh, R. Pagh, and S. Srinivas Rao. An Optimal Bloom Filter Replacement. In *Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 823-829, 2005.

[13] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. ECCC Report No. 10, 2004.

[14] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252-271, 1996.

[15] K. Shanmugasundaram, H. Brunnimann, and N. Memon. Payload attribution via hierarchical Bloom filters. In *Proc. of the 11th ACM Conference on Computer and Communications Security*, pp. 31-41, 2004.

[16] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer. Single-Packet IP Traceback. *IEEE/ACM Transactions on Networking*, 10(6):721-734, 2002.

An Experimental Study of Old and New Depth Measures*

John Hugg† Eynat Rafalin† Kathryn Seyboth† Diane Souvaine†‡

Abstract

Data depth is a statistical analysis method that assigns a numeric value to a point based on its centrality relative to a data set. Examples include the half-space depth (also known as Tukey depth), convex-hull peeling depth and L_1 depth. Data depth has significant potential as a data analysis tool. The lack of efficient computational tools for depth based analysis of large high-dimensional data sets, however, prevents it from being in widespread use.

We provide an experimental evaluation of several existing depth measures on different types of data sets, recognize problems with the existing measures and suggest modifications. Specifically, we show how the L_1 depth contours are not indicative of shape and suggest a PCA-based scaling that handles this problem; we demonstrate how most existing depth measures are unable to cope with multimodal data sets and how the newly suggested proximity graph depth addresses this issue; and we explore how depth measures perform when the underlying distribution is not elliptic.

Our experimental tool is of independent interest: it is an interactive software tool for the generation of data sets and visualization of the performance of multiple depth measures. The tool uses a hierarchical render-pipeline to allow for diverse data sets and fine control of the visual result. With this tool, new ideas in the field of data depth can be evaluated visually and quickly, allowing researchers to assess and adjust current depth functions.

1 Introduction

Over the last decade, statisticians have developed the concept of *data depth* as a method of multivariate data analysis that is an attractive alternative to classical statistics [47, 36, 35]. In this era, massive data sets are the norm, whether the discipline is financial markets, human biology, molecular biology or sociology. *Data depth* provides the ability to analyze, quantify and visualize these data sets without making prior assumptions about the probability distribution from which they come.

Proposed *data depth* metrics are inherently geometric, with a numeric value assigned to each data point that represents its *centrality* within the given data set. The *depth median*, the point of maximal depth, is the depth based estimator for the center of the data set. *Depth contours* can be used to visualize and quantify the data (see, e.g. [36]).

Data depth remains a relatively new field. A number of *data depth* measures have been proposed, analyzed, and, in some cases, coded, and new *data depth* measures continue to be proposed. Examples include *convex-hull peeling depth* [13, 4], *half-space depth* [21, 56], *simplicial depth* [34], *regression depth* [44, 48] and L_1 *depth* [57]. Despite all of the theoretical analysis and individual experiments, there is no conclusive evidence as to which depth measure should be applied to which data sets. One key problem is that several *data depth* measures which perform beautifully in two dimensions quickly become impractical as the dimensionality increases. Others that can be effectively computed in higher dimensions are not statistically significant and produce output that can be misleading.

The goal of this work has been to develop **Depth Explorer** as an experimental platform for analysis and visualization both of random data sets and of pre-existing data sets using multiple different *data depth* measures. In particular, **Depth Explorer** includes implementations of standard *data depth* measures such as *half-space depth*, *convex hull peeling depth*, and L_1 *depth*. It also includes implementations of the newer class of *proximity graph data depth* measures [42, 43] such as *Delaunay depth*, *Gabriel Graph depth*, and *β-skeleton depth*. **Depth Explorer** allows the user to analyze and visualize a data set using each of these measures. More importantly, however, the **Depth Explorer** experimental platform allows the comparison of *depth measures*. In particular, our testing of L_1 *depth* demonstrated the poor results that L_1 *depth* generates on certain types of data set. And yet L_1 *depth* has been one of the few *data depth* metrics known to be computable in time that is linear in dimension, rather than exponential. We have developed an enhanced version of L_1 *depth* that we call L_1 *scaling depth* that retains the computational efficiency of L_1 *depth* but produces much better output.

The paper reports not only on the development, availability, and features of the **Depth Explorer**

*This material is based upon work supported by the National Science Foundation under Grant No. CCF-0431027.

†Department of Computer Science, Tufts University, Medford, MA 02155. {jhugg,erafalin,kseyboth,dls}@cs.tufts.edu

‡2005-2006 MIT Visiting Scientist & Radcliffe Institute Fellow.

Sandbox but also on the results of the experiments we have performed using this platform. Section 2 provides background information on the *data depth concept*, including applications and examples of depth functions. Section 3 provides technical information about **Depth Explorer**. Sections 4, 5 and 6 present analysis of data sets and depth measures using **Depth Explorer** and Section 7 describes future work.

2 Data Depth

A *data depth* measures how deep (or central) a given point $x \in \mathbb{R}^d$ is relative to F, a probability distribution in $\mathbb{R}^d$, or relative to a given data cloud.

Sections 2.1 and 2.2) introduce general principles of data depth irrespective of the particular data depth functions chosen . Definitions of four types of data depth functions, with different features and computational complexities, are presented in Section 2.3).

2.1 General Concepts

The following concepts apply to the data depth methodology and distinguish it from other statistical methods.

- **Non-parametric methodology:** Scientific measurements can be viewed as sample points drawn from some unknown probability distribution, where the analysis of the measurements involves computation of quantitative characteristics of the probability distribution (*estimators*), based on the data set. If the underlying distribution is known (for example normal distribution, log-normal distribution, Cauchy, etc.), the characteristics of the data can be computed using methods from classical statistics. However, in most real life experiments the underlying distribution is not known. The concept of data depth requires *no assumption* about the underlying distribution and data is analyzed according to the relative position of the data points.

- **Center-outward ordering of points:** The data depth concept allows the creation of a multivariate analog to the univariate statistical analysis tool of *rank statistics*. *Rank statistics* is based on the ordering of one-dimensional observations, where the order *reflects extremeness, contiguity, variability or the effect of external contamination and provides a parameter estimation method* [4]. If $\mathcal{S} = \{X_1, \ldots X_n\}$ is a sample of observations in $\mathbb{R}^1$ then the order statistics is defined as $\{X_{[1]}, \ldots X_{[n]}\}$ where $X_{[1]} \leq X_{[2]} \ldots \leq X_{[n]}$.[1] In higher dimensions the order of multivariate data is not well defined, and several ordering methods were suggested (e.g. [4]). The data depth concept provides a method of extending order statistics to any dimension by ordering the points according to their depth values.

- **Application to multivariate (high-dimensional) data sets:** The concept of data depth is defined with respect to points in Euclidean space in *any* dimension, thus enabling the derivation of multivariate distributional characteristics of a data set. The methodology enables the exploration of high dimensional data sets using simple two-dimensional graphs that are easy to visualize and interpret, and using quantitative estimators.

- **Robustness:** In the statistical analysis of datasets, observations that deviate from the main part of the data (*outliers*) can have an undesirable influence on the analysis of the data. Many depth functions are "robust against the possibility of one or several unannounced outliers that may occur in the data and yield reasonable results even if several unannounced outliers occur in the data" [46]. For example, "by adding k bad data points to a data-set one can corrupt at most the k-outermost (half-space) depth contours while the ones inside must still reflect the shape of the good data" [12].

2.2 Depth Contours, Median and Mode

The **median** is a depth based estimator for the center of a data set. The median of a set S under some depth measure $D : S \rightarrow \mathbb{R}$ is the set of points M such that $\forall p \in M$, $D(p) \geq D(q)$ $\forall q \in S$. This definition supports the possibility that several points will tie for the deepest depth. The use of a single point or group of points as the median relies on the assumption of unimodality that is common in depth measures. **Depth Explorer** highlights the median points, visualizing the center of a data set.

The **mode** of a set is the most common value [59]. We use the term mode flexibly and refer to a *bimodal* (or *multimodal*) distribution or point set as one having two (or more) local maxima. The multiple maxima can be created, for example, from overlaying two different unimodal distributions. Often clustering algorithms are used to detect the points associated with each mode of the data set. This association, however, does not necessarily attribute a data point to the center of the distribution where it originated. For example, points located between two centers and far from each could be assigned to either of the two clusters.

Depth contours [55] are nested regions of increas-

[1] An alternative ranking order is from the outside inward, where the deepest point equals the median.

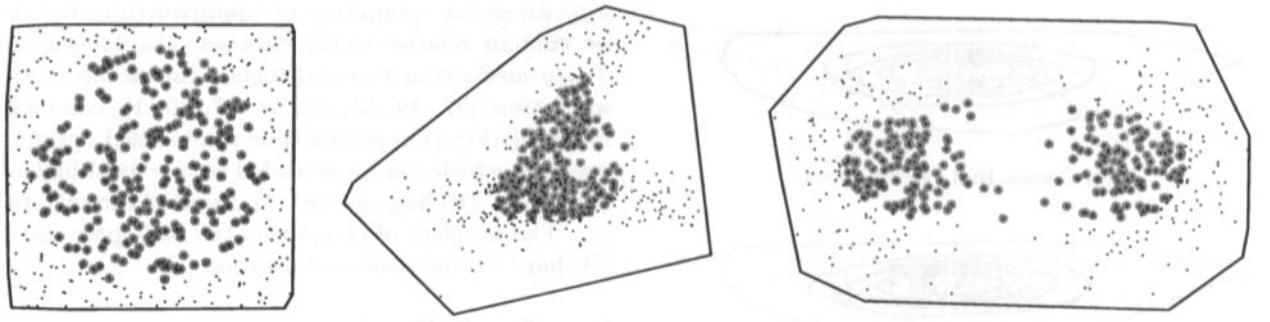


Figure 1: 50% deepest points highlighted for different distributions calculated using Half-space Depth. All distributions contain 500 points.

ing depth and serve as a topological map of the data. Let $D_F(x)$, $x \in \mathbb{R}^d$, be the value of a given depth function for point x with respect to a probability distribution F. The **region enclosed by the contour of depth** t is the set $R_F(t) = \{x \in \mathbb{R}^d : D_F(x) \geq t\}$. The α **central region,** C_α $(0 \leq \alpha \leq 1)$ is, for well behaved distributions, the region enclosed by the contour of depth t_α $C_\alpha = R_F(t_\alpha)$, where $P\{x \in \mathbb{R}^d : D_F(x) \leq t_\alpha\} = \alpha$ [36]. Well-behaved depth functions produce depth contours that are affinely equivariant, nested, connected and compact [61]. Contours have applications in visualization and quantification of data sets.

In **Depth Explorer** highlighting the $100\alpha\%$ deepest points visualizes the α central region. See, e.g., Figure 1.

If the underlying distribution is elliptic then the *convex hull* containing the $\alpha\%$ deepest points is a simplified sample estimate of the boundary of the contour [35]. However, in such a case the resulting contour may also contain shallower data points, that are not the $\alpha\%$ deepest points. In real life data sets usually the underlying probability distributions is not known, but if the data set is unimodal and convex then it is reasonable to assume that the underlying probability distribution is elliptic.

As a method of comparing the performance of a variety of depth measures, **Depth Explorer** can display the convex hulls enclosing the $20\%, \ldots 100\%$ deepest points under each of these measures, visualizing the type of contour produced by each measure. See Figure 2.

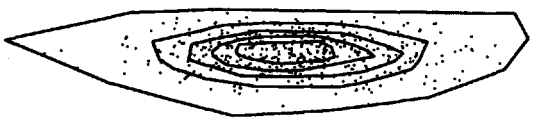
2.3 Depth Measures

We present four types of depth functions, with different features and computational complexities, that are all implemented in the **Depth Explorer Sandbox**. The *convex-hull peeling depth* is one of the early depth measures studied by the computational geometry community, but it lacks many statistical properties. *Half-space depth* is probably the best-known depth measures in the computational literature and has attractive statistical properties. The L_1 *depth* does not possess many desirable statistical properties, but the simplicity of its computation makes it useful for certain application of data depth. The newly suggested *proximity depth* was developed as a depth measure that is efficient to compute in high dimensions.

2.4 Convex-Hull Peeling Depth

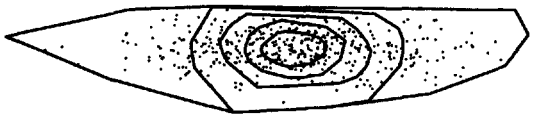
DEFINITION 2.1. *The* **convex-hull peeling depth** *[13, 4] of a point X_k with respect to a data set $\mathcal{S} = \{X_1, \cdots X_n\}$ in $\mathbb{R}^d$ is the level of the convex layer to which X_k belongs. The level of the convex layer is defined as follows: the points on the outer convex hull of $\mathcal{S}$ are designated level one and the points on the kth level are the points on the convex hull of the set $\mathcal{S}$ after the points on all previous levels were removed (see Figure 2(a)).*

Convex-hull peeling depth is appealing because of the relative simplicity of its computation. However it lacks distributional properties and is not robust in the presence of outliers. The convex-hull peeling layers can be computed in the plane in optimal $\Theta(n \log n)$ time using Chazelle's deletions-only dynamic convex-hull algorithm [8] and in higher dimensions using iterative applications of any convex-hull algorithm (the complexity of computing the convex hull for a set of n points in $\mathbb{R}^d$ *once* is $O(n \log n + n^{\lfloor \frac{d+1}{2} \rfloor})$ for even d [41] and $O(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$ for odd d [9] and it can be applied as many as $O(n)$ times to compute the entire set of peeling layers). The vertices of all convex layers can be computed in $O(n^{2-\gamma})$ time for any constant $\gamma < 2/(\lfloor d/2 \rfloor^2 + 1)$ [6].



(a) Convex Hull Peeling Depth

(b) Halfspace Depth



(c) L_1 Depth

(d) Delaunay-Based Proximity Depth

Figure 2: 20%, 40%, 60%, 80% and 100% contours for a data set consisting of 500 points, normally distributed, width scaled by 5. Note that the L_1 depth contours appear more round that is warranted by the distribution, while the Delaunay-Based Proximity Depth contours are the most elongated.

2.5 Half-space Depth

DEFINITION 2.2. *The* **half-space depth** *[21, 56] (in the literature sometimes called location depth or Tukey depth) of a point x relative to a set of points $\mathcal{S} = \{X_1, ..., X_n\}$ is the minimum number of points of $\mathcal{S}$ lying in any closed half-space passing through x (see Figure 2(b)).*

The half-space depth has many attractive statistical properties [61, 12]. However, its computation is exponential in dimension. The half-space depth of a *single point* in $\mathbb{R}^2$ can be computed in $O(n \log n)$ time [52], matching the lower bound [2]. The set of half-space *depth contours* can be computed in the plane in optimal $\Theta(n^2)$ time [38] (expanding upon ideas in [11]), or using other methods [52, 25, 29], including computation of the depth contours in $\mathbb{R}^d$ using parallel arrangement construction [17]. Theoretical and practical algorithms for computing or approximating the deepest contour relative to the data set (also known as the *Tukey median*) in two and higher dimensions exist for some time [52, 49, 51, 53, 58, 37, 30, 1], culminating in an $O(n \log n)$ expected time randomized optimal algorithm, which can be extended to any fixed higher dimension d, yielding an $O(n^{d-1})$ expected time algorithm [7]. The problem of computing the half-space depth is NP-hard for unbounded dimension [3].

2.6 The L_1 Depth

DEFINITION 2.3. *The L_1* **depth** *(L_1D) [57] of a point x with respect to a data set $\mathcal{S} = \{X_1, \cdots X_n\}$ in $\mathbb{R}^d$ is one minus the average of the unit vectors from x to all observations in $\mathcal{S}$:*
$L_1D(S, x) = 1 - \|\overline{e}(x)\|$, *where* $e_i(x) = \frac{x - X_i}{\|x - X_i\|}$, $\overline{e}(x) = \frac{\sum_{i=1}^{n} \eta_i e_i(x)}{\sum_j \eta_j}$. *$\eta_i$ is a weight assigned to observation X_i (and is 1 if all observations are unique), and $\|x - X_i\|$ is the Euclidean distance between x and X_i (see Figure 2(c)).*

Intuitively, the L_1 median of a cloud of points in $\mathbb{R}^n$ is the point that minimizes the sum of the Euclidean distances to all points in the cloud. The L_1 depth of a point can be summarized by the question, "How much does the cloud need to be skewed before this point becomes the L_1 median?". The L_1 depth ranges between 0 and 1. It is fast and easy to compute in any dimension, contributing to its appeal for study of large high-dimensional data sets. The L_1 depth is non-zero outside the convex hull of the data set and therefore can be used to measure within-cluster and between-cluster distance (see Section 7.1). However, it lacks many statistical properties.

2.7 Proximity Depth

For any proximity graph the proximity graph depth is defined using a point's minimum path length along graph edges to the convex hull of the data set $\mathcal{S}$.

DEFINITION 2.4. *The* ***[proximity graph] depth*** *of a point x relative to a set $\mathcal{S} = \{X_1 \ldots X_n\}$ is the minimum number of edges in the [proximity graph] of $\mathcal{S}$ that must be traversed in order to travel from x to any point on the convex hull of $\mathcal{S}$ (see Figure 3).*

Proximity graphs are graphs in which points close to each other by some definition of closeness are connected [24]. We concentrate our analysis on the *Delaunay triangulation* [16] and *β-skeletons* [28] which are a parameterized family of proximity graphs, which include as a special case the *Gabriel graph* and the *relative neighborhood graph*. We denote by $\delta(p, q)$ the Euclidean distance between points p and q.

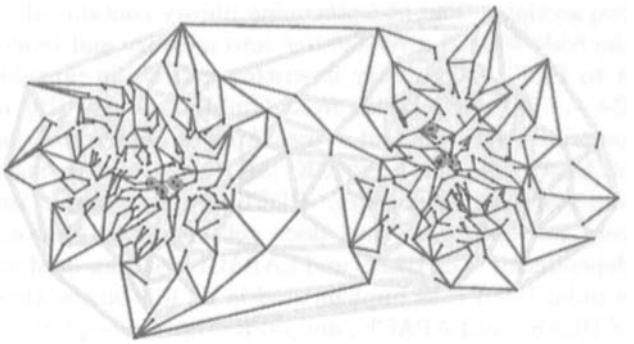


Figure 3: Exploring a proximity depth calculation. The edge-distance from each data point to a convex-hull point is illustrated using highlighted trees. Two normal clouds with 250 points separated horizontally by 6 standard deviations.

DEFINITION 2.5. *The **Delaunay triangulation** (DT) of a d-dimensional point set S is the simplicial decomposition of the convex hull of S such that the d-sphere defined by the points of every simplex in the decomposition contains no point $r \in S$ [16].*

This decomposition is the dual of the *Voronoi diagram* and is unique for every set of points [14].

DEFINITION 2.6. *The β **skeleton** of a point set S in $\mathbb{R}^d$ is the set of edges joining β-neighbors.*
*Points p and q are **lune-based β-neighbors** for $\beta \geq 1$, iff the lune defined by the* intersection *of the spheres centered at $(1-\frac{\beta}{2})p+\frac{\beta}{2}q$ and $(1-\frac{\beta}{2})q+\frac{\beta}{2}p$, each with radius $\frac{\beta}{2}\delta(p,q)$, contains no point $r \in S$.*
*Points p and q are **circle-based β-neighbors** for $\beta \geq 1$, iff the lune defined by the* union *of the two spheres of radius $\frac{\beta}{2}\delta(p,q)$ contains no point $r \in S$.*
*Points p and q are **β-neighbors** for $\beta < 1$, iff the lune defined by the intersection of the two sphere of radius $\frac{\beta}{2}\delta(pq)$ which contain p and q in their boundary contains no point $r \in S$ (for $\beta < 1$ the lune-based and circle-based neighbors are identical).*

For $\beta > 1$ the **lune-based β-skeletons** are planar and monotonic with respect to β: $G_{\beta_1}(S) \subset G_{\beta_2}(S)$, for $\beta_1 < \beta_2$. The *Gabriel graph* (GG) [18] is the lune-based 1-skeleton while the *relative neighborhood graph* (RNG) [54] is the lune-based 2-skeleton. The **circle-based β-skeletons** for $\beta > 1$, are not necessarily planar and have a reverse monotonic relation with respect to β: $G_{\beta_1}(S) \subset G_{\beta_2}(S)$, for $\beta_1 > \beta_2$.

For $\beta < 1$, as β becomes smaller, the β skeleton tends towards the complete graph.

Overall Complexity The depths of all points in a proximity graph can be determined in linear time in the number of edges in the graph by using a breadth-first search (BFS) of the graph, beginning at every point on the convex hull of $\mathcal{S}$ (Figure 3). Assignment of all depths of a point set is accomplished by (1) Computation of the proximity graph; (2) Location of all convex hull points; and (3) Breadth-first search of the proximity graph.

In two dimensions there are optimal $O(n \log n)$ time algorithms to compute the DT [16], the circle-based β-skeletons for $\beta \geq 1$, and the lune-based β-skeleton for $1 \leq \beta \leq 2$ [28, 33, 23]. The lune-based β-skeletons for $\beta > 2$ and the β-skeletons for $\beta < 1$ can be computed in optimal $O(n^2)$ time [28, 23]. Points on the convex hull can be determined in $O(n \log n)$ time [40], for an overall time requirement of $O(n \log n)$ for the DT and $O(n^2)$ or $O(n \log n)$ for the β-skeleton.

In dimensions higher than 2, the DT can be calculated in $O(n^{\lceil \frac{d}{2} \rceil})$ time [15]. The β-skeletons require checking n points for interiority on n^2 lunes, which requires a distance calculation for a total of $O(dn^3)$ time. More efficient algorithms for specific graphs like the GG or RNG or for 3-dimensional space are known [24]. The set of points on the convex hull of the set can be found in $O(mn)$ time, where m is the number of extreme points [39]. Breadth-first search then requires linear time in the size of the proximity graph. Clearly, the time complexity in higher dimensions is dominated by the computation of the proximity graph itself. Assignment of all depths, then, has a total complexity of $O(n^{\lceil \frac{d}{2} \rceil})$ time for Delaunay depth and $O(dn^3)$ time for the β-skeleton depths. The exponential dependence on dimension for calculating Delaunay depth makes it impractical for use in high dimensions.

3 The Depth Explorer Statistical Sandbox

Depth Explorer is an interactive software tool for the generation of data sets and visualization of the performance of multiple depth measures. The tool is aimed for use by statisticians, to visually and quickly evaluate new ideas in the field of depth based statistics, allowing researchers to assess and adjust current depth functions. It was designed to be simple to use. The tool uses a hierarchical render-pipeline to allow for diverse data sets and fine control of the visual result. The Depth Explorer source, executable for Mac OS X and documentation is publicly available and can be found in [22].

3.1 Scene Generation and the Render Tree

Depth Explorer enables automatic generation of data sets and transformation or composition of existing data sets. For each data set, **Depth Explorer** can quickly visualize the behavior of the depth measure on the data. Data sets are defined using a hierarchical representation

of the scene (the *render-tree*) in an XML file. Clicking a toolbar button switches to "live" view and renders the scene to the screen. The user can then switch back into XML editor mode to adjust the scene and then re-render. The changes are almost instantly re-rendered into the window, allowing for interactive experimentation. Generated PDF files can be saved to the filesystem.

Data generation, transformations and visualizations are specified hierarchically using XML tags in the *render tree*, see Figure 4. The tree is rendered starting from the leaf nodes and moving up. Each node is a C++ module that, given the output of its children nodes, modifies or appends the data, then passes it to its parent. Ultimately, the root node, called the *canvas* node describes the dimensions and scale of the PDF output file and renders its childrens' data onto the PDF document. The leaf nodes are data sources, either CSV files with a data set, or a random cloud generator with a set of points. **Depth Explorer** can currently create Gaussian clouds with any number of points with standard deviation 1, as well as uniformly distributed clouds on the range $(-1,-1)$ to $(1,1)$.

Non-leaf nodes modify data or add visualizations. **Depth Explorer** supports affine transformation nodes that can scale, rotate or translate nodes below them. With affine transformations and any number of randomly generated clouds, a broad spectrum of data sets can be generated to test statistical methods.

Visualizations include the display of the $\alpha\%$ contour by highlighting the $\alpha\%$ deepest points. If the set is dense enough this serves as a good visual approximation to the $\alpha/100$ central region, see, e.g., Figure 1. **Depth Explorer** displays the convex hulls enclosing the $20\%, 40\%, 60\%, 80\%$ and 100% deepest points, to visualize a subset of the depth contours, under the assumption that the underlying distribution is elliptic and the *convex hull* containing the $\alpha\%$ deepest points is a simplified sample estimate of the boundary of the contour, see Figure 2. Note that the resulting contour may contain shallower data points, that are not the $\alpha\%$ deepest points.

As **Depth Explorer** uses a modular, tree-based renderer, it is trivial to combine visualizations, even at different points in the hierarchy. By viewing two visualizations on the same data, it is possible to compare depth measures or convey two concepts in one image. Figure 3 illustrates how **Depth Explorer** can help visualize a powerful concept, such as computation of the proximity depth.

3.2 Depth Explorer Construction

The **Depth Explorer** program is currently divided into two sections. The *libdepthengine* library contains all of the code to load a *render-tree* into memory and render it to PDF. This library is written entirely in portable C++. It uses PDFlib from GmbH Software [19] to generate PDF files and it uses Apple's implementation of BLAS [5] and LAPACK [31] to do matrix math and compute eigenvalues. PDFlib is open source and very portable, and thus does not introduce platform dependency. The BLAS and LAPACK routines used are standard and thus implemented in all implementations of BLAS and LAPACK, and do not introduce platform dependence as well.

The **Depth Explorer** GUI links against *libdepthengine* and is responsible for editing XML data, viewing rendered scenes, saving rendered scenes as PDF files and printing. The GUI was developed with Objective-C using the Apple's Cocoa [10] application frameworks. Although the implementation is platform dependent, the amount of code is small and the complexity is very low compared to *libdepthengine*. For example, while the Cocoa frameworks provide support for XML parsing, parsing in **Depth Explorer** is done inside *libdepthengine* in portable C++.

3.3 Performance and Interactivity

Depth Explorer does not render instantaneously even a relatively easy depth measure like L_1. Nonetheless, almost all scenes are rendered within seconds, not minutes, allowing for interactive feedback. On a 1.5 Gigahertz Powerbook G4, rendering most scenes commonly requires 2-5 seconds. Rendering time is heavily dependent on the processing that is done in each node. If no depth-based visualizations are included in the render tree, than even the most complicated scene with thousands of data points will render almost instantly. When depth calculations are involved, computation time is slower.

To render an L_1 50% depth contour on a cloud of 1000 points takes about two seconds. To render the same contour on 10,000 points requires about 5 seconds. While the scale-up should be linear according to the algorithm, the software is tuned to handle large amounts of data, thus the slowdown is only apparent in very large datasets or with very slow computations.

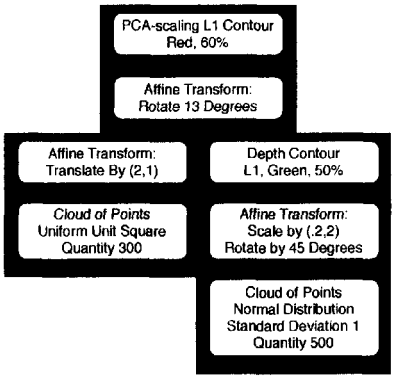
Thus **Depth Explorer** is not a "sit and wait" program: almost all scenes can be rendered in a matter of seconds, not minutes. Conversely, **Depth Explorer** is not a real-time program: it is not fast enough to give dynamic feedback as the user changes parameters to the render tree nodes. Thus a render button is required. As more (slower) depth measures[2] and complex visu-

[2]Depth Explorer's current implementation of Halfspace Depth

XML Representation of Render Tree

```
<canvas width='6' height='6' margins='.1' minrange='6'>
    <pcabag color='1,0,0' size='.6'>
        <transform angle='13'>
            <transform xtrans='2' ytrans='1'>
                <cloud type='uniform' points='300'/>
            </transform>
            <bag type='11' color='0,1,0' size='.5'>
                <transform xscale='.2' yscale='2' angle='45'>
                    <cloud type='normal' points='500'/>
                </transform>
            </bag>
        </transform>
    </pcabag>
</canvas>
```

Internal Representation of Render Tree



Final Rendered Output

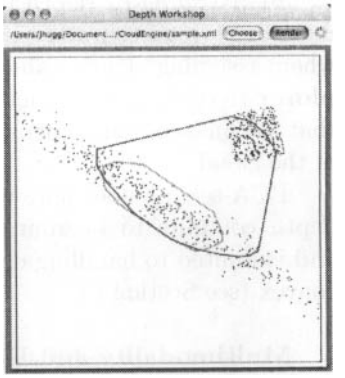


Figure 4: The Render tree form XML to Final Scene. This scene is defined by two source clouds, a uniform square with 300 points and a normal cloud with 500 points. The square is translated and rotated while the normal cloud is scaled and rotated. Both clouds are rotated 13 degrees together. Two depth contours are calculated, one L_1 50% contour on the normal cloud and one PCA-Based L_1 Scaling 60% contour on the entire scene.

alizations are supported, it seems unlikely that **Depth Explorer** will ever become fully realtime. See Section 7 for a discussion of speed improvements.

4 Analysis of the L_1 Depth Measure

The L_1 Depth measure has many statisticians very excited as the computation time is very low. With a fast depth measure available, suddenly the concept of data depth is applicable to many new and more complex problems. L_1 Depth is hoped to be "good enough", that is, its speed will make up for its sub-par results. Still, as L_1 Depth is a new and relatively untested depth measure, it is still unknown just how much quality of result is compromised for speed.

To answer this question, we analyzed the L_1 depth function on unimodal normally distributed data sets that were scaled or rotated. It is desirable that depth contours will be representative of the shape of the of data. However, as we demonstrated visually (see Figure 2) L_1 depth contours do not exhibit this desirable property, while all other tested depth measure do. Since the contour is supposed to be indicative of the shape of the cloud, the L_1 depth measure is not as useful as many other depth measures.

Using **Depth Explorer** to visualize how L_1 Depth performs on various data sets, we discovered that, under the L_1 depth, a point whose distance to the median point is small is more likely to have higher depth than the same point in other depth measures. Consequently, L_1 depth contours tend to be more circular than desired.

As L_1 depth is most successful on hyper-spherical data, by performing an affine transformations for non-spherical data we can create a shape that approaches a hypersphere and compute the L_1 depth on this scaled cloud. The depth values for the scaled cloud are then used for the corresponding points in the original cloud.

This works very well when data sets are scaled along an axis, but many times, a data set is elongated or compressed in directions not parallel to an axis. Thus, Principal Component Analysis (PCA) is used to determine the major directions of the data cloud. Scaling along these vectors produces an approximation of a hyperspherical cloud for many data sets (see Figure 5).

DEFINITION 4.1. *The k-PCA-based L_1 depth of a point x with respect to a data set $\mathcal{S} = \{X_1, ...X_n\}$ in $\mathbb{R}^d$ is the L_1 depth of $pca(x)$ with respect to a data set $pca(\mathcal{S}) = \{pca(X_1), \ldots pca(X_n)\}$. Let $v_1, v_2, \ldots v_n$ be the eigenvectors for $\mathcal{S}$, computed using PCA analysis. Then $pca : \mathbb{R}^d \to \mathbb{R}^d$ is defined as follows: (i) rotate $\mathcal{S}$ so that the first k primary vectors coincide with the axes and (ii) For each of the primary k axes, project the points of $\mathcal{S}$ on the axis and scale the projected set such that the .8 and .2 quantiles go to 1 and -1 respectively. (as a result of this transformation 60% of the data fits between -1 and 1 for each of the primary k axis).*

The *PCA-based L_1 depth* of a point x with respect to a data set $\mathcal{S}$ in $\mathbb{R}^d$ is the d-PCA-based L_1 depth. We assume that data sets are centrally symmetric relative to each of the axis and therefore always use the d-PCA-

and Beta-Based Proximity depth are naive and thus have $O(n^3)$ time complexity. This limits these particular depth calculations to about 500 points. This will be improved in a upcoming minor revision.

based L_1 depth.

PCA-scaling-L_1 depth removes much of the aforementioned bias towards points closer to the median when assigning depth values. We used **Depth Explorer** to confirm this visually, see, e.g. Figure 5. Note that the final contour generated approximates the shape of the cloud.

PCA-scaling does not, address the tendency for L_1 depth contours to be rounded in square data clouds and is limited to handling data clouds that are roughly convex (see Section 6).

5 Multimodality and Depth Measures

5.1 Multimodal Distribution

Depth Explorer was tested on multimodal data sets and the behavior of depth functions on these data sets was analyzed. The tested depth functions, convex-hull peeling depth, half-space depth and L_1 depth, were not well suited for analysis of multimodal data sets (see Figure 8). However, the newly suggested proximity depth was found to handle these data sets well, see Figure 8 and 9.

5.2 Median and Seeds

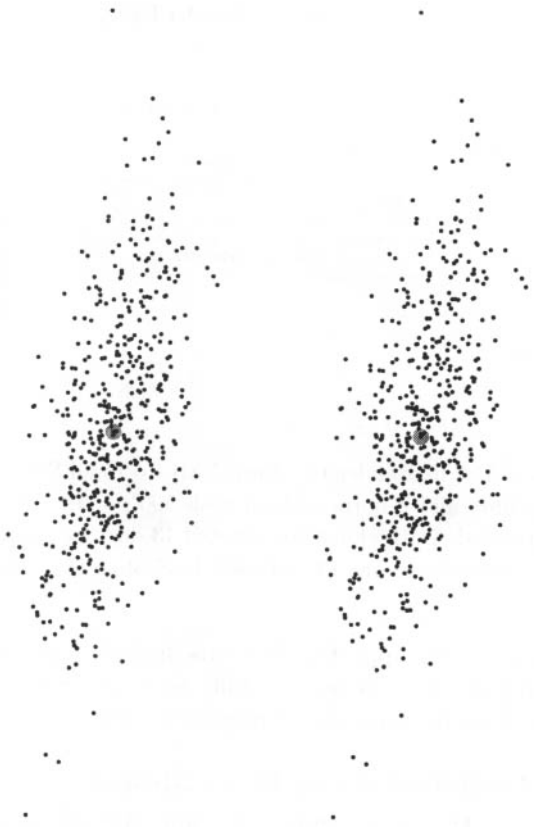
The median, as it was defined in Section 2, is not a good estimator in multimodal situations, because points that are not of maximum depth in the set may in fact be local maxima, local peaks in depth (see Figure 6). Estimating a set using a simple median ignored local maxima that represent several modes. The proximity graph depth allows the definition of an estimator that handles modes of a data set:

DEFINITION 5.1. *A **seed** of a point set S under some depth measure $D : S \to \mathbb{R}$ is a connected set of points $T \subset S$ st $\forall p, q \in T, D(p) = D(q)$ and $\forall r \in S, r \notin T$ adjacent to some $u \in T, D(r) < D(u)$.*

Unfortunately, points of different clusters are not necessarily distinguished by the proximity graph depth measures. Distributions that are too close behave as a single mode; those separated by large empty regions appear unimodal as the dearth of points between the clusters prevents paths from traveling inward quickly from the convex hull.

Quantitative analysis

Tests performed on Depth Explorer verify the abilitiy of the proximity graph seeds to discern unimodality and bimodality quantitatively. Analysis of both bimodal and unimodal planar point sets was performed using a separate program that computed the seeds. Bimodal sets in two dimensions (containing x and y coordinates) were created by combining two 200-point normal distribution sets, each with x and y standard deviations of 10. They began centered at the same x-coordinate, but were then separated in increments of 10. The unimodal sets began with an x-coordinate standard deviation of 10, which was gradually increased to stretch the unimodal set in the x-direction. We plotted the standard deviation of the x-values of the points of local maxima against the x-coordinate range of the set. When the bimodal sets are close, they behave very similarly to the unimodal sets, but as they pull apart their seeds separate, illustrating the bimodal behavior (Figures 7 and 9). This behavior is similar for each of the proximity graph depth measures.



(a) Convex Hull Peeling Depth (b) Halfspace Depth

Figure 6: Highlighted deepest point approximates the depth median. 500 points, Y-axis compressed 70%, rotated 10 degrees.

Algorithms Finding seeds requires a recursive search to locate all points associated with the seed and to locate all points in S that are connected to that seed. The basic process to compute the seeds is that of comparing the depth of point p to the depth of its neighbors in the proximity graph and checking

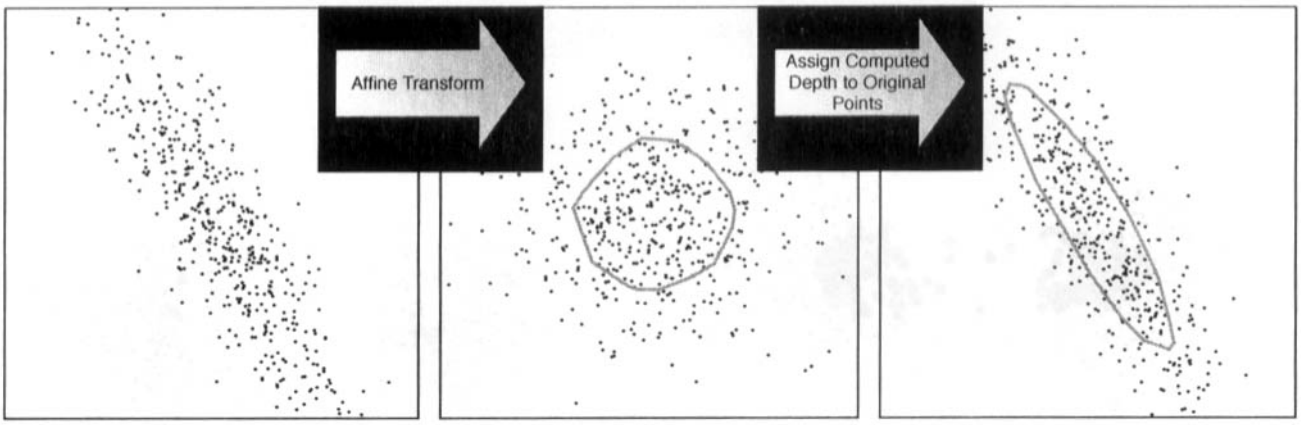


Figure 5: How Affine Scaling Improves the L_1 Depth Results

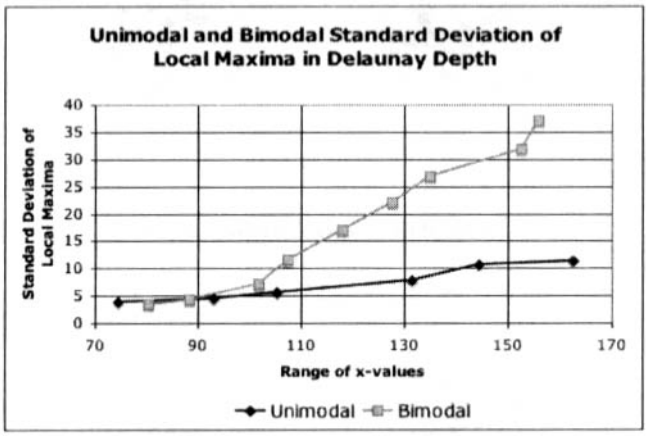


Figure 7: Computation of seeds using the Delaunay depth function for unimodal and bimodal point sets. For every x-value X the unimodal point set was constructed to have comparable width to the width of the bimodal point set, whose separation is X. The standard deviation of the x-values of the local maxima were computed. The results support our assumption, that in a bimodal point set we expect to find wider gaps between the points of local maxima and therefore larger standard deviation compared to unimodal point sets.

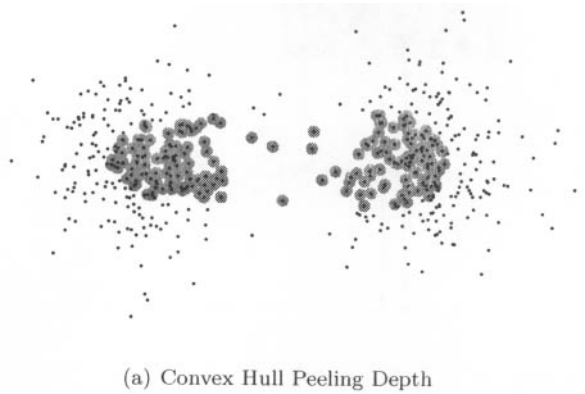
whether it is deeper or of the same depth as its neighbors (*compare(p)*).

One method of computation searches the list of points to locate those that are deeper or of the same depth as their neighbor. In this case, every connected grouping of points must be checked and there can be $O(n)$ such groupings. The *compare(p)* process is called recursively for each point p in a grouping exactly once and obtains a yes/no answer for that point before returning. Because *compare(p)* eliminates points of lower depth than p, it is called at most n times. Each call iterates through some portion of p's adjacency list. Each edge in the graph represents two entries in adjacency lists, and is considered exactly twice, producing an algorithm with a running time that is linear in the size of the proximity graph. The size of the graph is linear in two dimensions and up to quadratic in higher dimensions. The process does not add to the overall complexity, because construction of the graphs requires at least as much time.

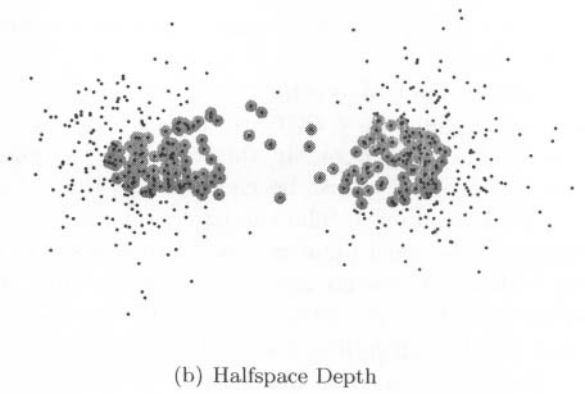
Additional improvements allow the number of seeds to be decreased further (see [42, 43]). For example, only those seeds with **CH-value** great than 2 and over half of the maximum (**CH-point seeds**). The *CH-value* of a seed T is the number of convex hull vertices that can be reached by a path originating at T for which the depths of points visited along the path is strictly decreasing.

5.3 Proximity Graph Depth Function Family

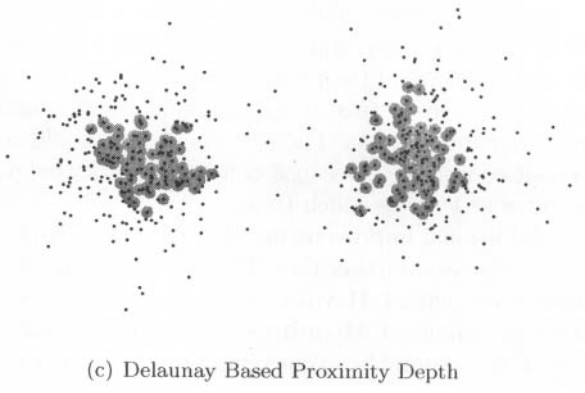
We compared the family β-skeleton depth functions with the DT depth visually, using the **Depth Explorer** (see Figure 9), and quantitatively, with a separate code written in C++, using the LEDA library [32]. For the quantitative analysis 300 normally distributed data sets with 400 points each were generated and the average number of seeds produced was calculated.



(a) Convex Hull Peeling Depth



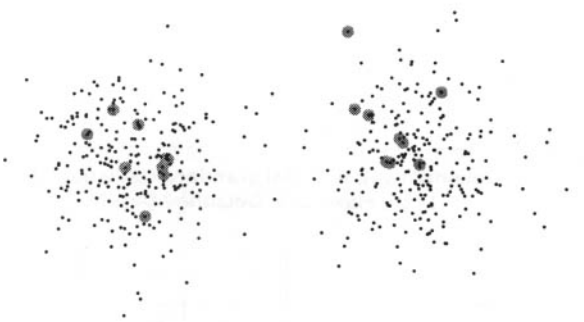
(b) Halfspace Depth



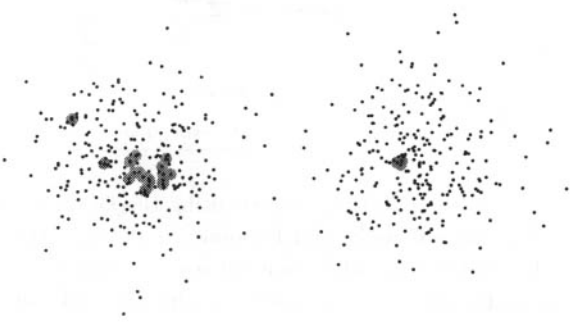
(c) Delaunay Based Proximity Depth

Figure 8: Deepest 30% of the points of a bimodal distribution highlighted with different depth measures. The traditional depth measures' depth is biased towards the space between distributions, while proximity depth measures recognize bimodality. The distribution is two normal clouds with 250 points separated horizontally by 6 standard deviations.

(a) Delaunay-Based Proximity Depth



(b) Gabriel-Based Proximity Depth



(c) β-Based Proximity Depth ($\beta = .970$)

Figure 9: Highlighted seeds for a bimodal data set computing with three proximity depth measures. The distribution is two normal clouds with 250 points separated horizontally by 6 standard deviations.

The average number of CH-point seeds was plotted against the average deepest depth attained by the point set (Figure 10). The locations of the points indicate that the values $\beta = .962$ and $\beta = .970$ best approximate the performance of Delaunay Depth for these two characteristics. The weakness of the GG (1-skeleton) is also visible, as it finds many more seeds in a unimodal data set than the other graphs.

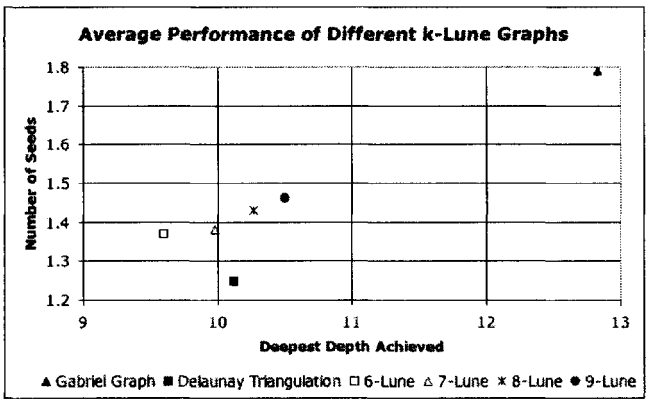


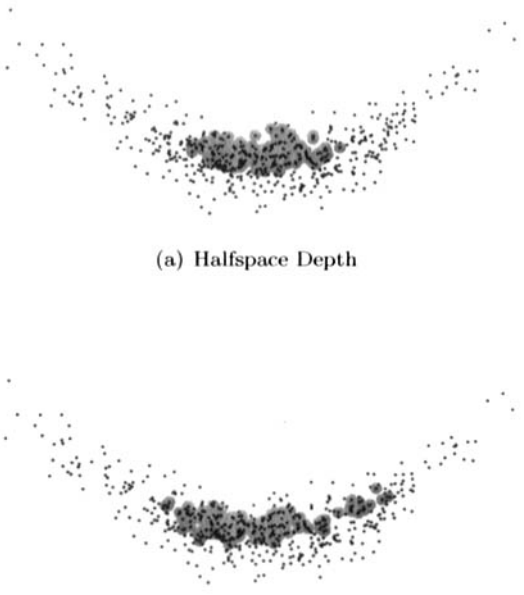
Figure 10: Performance comparison of proximity-depth schemes for 300 normally distributed unimodal point sets with 400 points. A low average number of seeds and a high average number of deepest depth values is desirable. The .962-skeleton and .970-skeleton perform most similarly to the DT The weakness of the GG (1-skeleton) is also visible here, as it finds many more seeds than the other graphs.

6 Non-Elliptic Underlying Distributions

We tested the behavior of depth functions on point sets that did not follow an elliptic distribution. It is desirable that the depth contours will be indicative of the shape of the data cloud. However, for most depth functions, this does not hold. In fact, the *four desirable properties of depth functions*, as suggested by statisticians as a tool to analyze and evaluate depth functions [34, 60, 61, 20] assume that the underlying distribution is unimodal and that the ordering of the points is center-outward.

When the cloud is not round, as in the square uniform plot from Figure 1(a), the depth contours are still round, when they should be more rectangular to match the shape of the data.

If the data points are not in a convex position, but follow a *banana-like* shape, as in Figures 11(a) and 11(b), none of the depth functions capture this shape, not even the proximity graph depth functions which are not convex. We are currently working on a version of the proximity graph depth that will handle this type of distribution.



(a) Halfspace Depth

(b) Delaunay-Based Proximity Depth

Figure 11: 30% deepest points highlighted where the underlying data set is distributed in non-convex position. Traditional depth measures bias towards the inside center of the curve, while proximity-based depth measures have much less bias, and can better represent shape.

7 Future Work

Depth Explorer has great potential as a tool for evaluation and analysis of depth functions, and the work presented here just scratches the surface. Our goal is to continue exploration of depth functions using this tool and to make it publicly available and more user-friendly, such that other researchers, especially in the statistics community, can make use of it.

7.1 Depth Explorer for Analysis of Real Life Data

Application of the data-depth concept to real-life data have been suggested by statisticians for a while. Many of them are two-dimensional graphs that visualize statistical properties of high-dimensional data sets. The flexible architecture of **Depth Explorer** can be augmented to compute and display these graphs

- The multivariate nature of data depth yields simple two dimensional graphs for high dimensional data sets, that can be easily visualized and interpreted [36]. Examples include *scale curves* as a measure of scale/dispression, tracking how the depth contours

expand; *shrinkage plots* and *fan plots* as a measure of *kurtosis*, the overall spread relative to the spread in the tail; and *depth vs. depth (DD) plot* to compare variation between two sample distributions.

- The *bagplot* [50] [3] is a visual tool based on the half-space depth function that is indicative of the shape of a two dimensional data set. It includes a *bag*, the contour containing the $n/2$ observations with largest depth. Magnifying the bag by a factor 3 yields the *fence*, where observations outside the fence are flagged as outliers. The bagplot is an affine invariant and robust visualization tools for the location, spread, correlation, skewness, and tails of a two dimensional data set.

- The robust nature of the data depth concept makes it appropriate to serve as a robust classification and cluster analysis tool. Rousseeuw *et al.* [52] suggest using the volumes of the depth contours that contain α of the points of each cluster (for a constant α) to classify a data point to a given cluster or to partition a given set into a number of clusters. Jörnsten, Vardi and Zhang [27] introduced the *Relative Data Depth, ReD*, based on the L_1 depth as a validation tool for selecting the number of clusters and identifying outliers, in conjunction with an exact K-median algorithm. The concept was tested on several real-life data sets [26].

7.2 Technical Enhancements

On the technical side, work will concentrate on the following directions:

- **Depth Explorer** is currently limited to two dimensional visualizations and data. Since many of the computations extend trivially to three or more dimensions, support for two dimensional visualizations on higher dimensional data is under development. Later extending the software to support three dimensional visualizations will be a priority, however, a three dimensional interface will require careful consideration and much development. This will be a major focus in the ongoing development of Depth Explorer.

- Massive improvements to the editor interface including direct integration with online help and automatic highlighting of XML errors. A live preview of the data distribution will be added, eliminating the current back and forth nature of constructing a scene.

- The process to add a new data generator, modifier or visualizer is currently very simple. Steps need to be taken to further simplify this process and fully document it. The goal is to make it nearly trivial to expand the functionality of **Depth Explorer** within the render tree framework.

- Additional visualization types, For example: coloring points for clustering visualizations according to membership; shading regions of depth rather than outlinining or highlighting them; points displaying their numerical depth value as they are moused over.

- The current tool is single-threaded and cannot take advantage of the recent push for dual core processors on the desktop. Making the render process parallelizable will decrease render time, especially as the size of the render increases.

- Support of additional platforms other than Mac OS X including Microsoft Windows. Depth explorer has been developed with portability in mind, however, as development resources are limited, development and testing on additional platforms may be a secondary priority for the immediate future.

[3] The *Sunburst plot* [36] is a similar visual tool.

References

[1] P. K. Agarwal, M. Sharir, and E. Welzl. Algorithms for center and Tverberg points. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 61–67, 2004.

[2] G. Aloupis, C. Cortes, F. Gomez, M. Soss, and G. Toussaint. Lower bounds for computing statistical depth. *Computational Statistics & Data Analysis*, 40(2):223–229, 2002.

[3] N. Amenta, M. Bern, D. Eppstein, and S.-H. Teng. Regression depth and center points. *Discrete & Computational Geometry*, 23(3):305–323, 2000.

[4] V. Barnett. The ordering of multivariate data. *J. Roy. Statist. Soc. Ser. A*, 139(3):318–355, 1976.

[5] BLAS. Basic linear algebra subprograms. `http://www.netlib.org/blas/`.

[6] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, 16(4):369–387, 1996. Eleventh Annual Symposium on Computational Geometry (Vancouver, BC, 1995).

[7] T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proceedings of 15th ACM-SIAM Symposium on Discrete Algorithms (SODA04)*. ACM Press, 2004.

[8] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, 31(4):509–517, 1985.

[9] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10(4):377–409, 1993.

[10] COCOA. Apple cocoa. `http://developer.apple.com/cocoa/`.

[11] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM Journal on Computing*, 15(1):61–77, 1987.

[12] D. L. Donoho and M. Gasko. Breakdown properties of location estimates based on halfspace depth and projected outlyingness. *Ann. Statist.*, 20(4):1803–1827, 1992.

[13] W. Eddy. Convex hull peeling. In H. Caussinus, editor, *COMPSTAT*, pages 42–47. Physica-Verlag, Wien, 1982.

[14] H. Edelsbrunner. *Algorithms in Computational Geometry.* Springer-Verlag, 1978.

[15] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, 1:25–44, 1986.

[16] S. Fortune. Voronoi diagrams and Delaunay triangulations. In *Handbook of discrete and computational geometry*, CRC Press Ser. Discrete Math. Appl., pages 377–388. CRC Press, Inc., Boca Raton, FL, USA, 1997.

[17] K. Fukuda and V. Rosta. Exact parallel algorithms for the location depth and the maximum feasible subsystem problems. In *Frontiers in global optimization*, volume 74 of *Nonconvex Optim. Appl.*, pages 123–133. Kluwer Acad. Publ., Boston, MA, 2004.

[18] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[19] Pdflib gmbh - main page, 2005.

[20] X. He and G. Wang. Convergence of depth contours for multivariate datasets. *Ann. Statist.*, 25(2):495-504, 1997.

[21] J. Hodges. A bivariate sign test. *The Annals of Mathematical Statistics*, 26:523–527, 1955.

[22] J. Hugg. Depth explorer website. `www.cs.tufts.edu/r/geometry/depthexplorer/`.

[23] F. Hurtado, G. Liotta, and H. Meijer. Optimal and suboptimal robust algorithms for proximity graphs. *Comput. Geom.*, 25(1-2):35–49, 2003. Special issue on the European Workshop on Computational Geometry—CG01 (Berlin).

[24] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 80(9):1502–1517, sep 1992.

[25] T. Johnson, I. Kwok, and R. Ng. Fast computation of 2-dimensional depth contours. In *Proc. 4th International Conference on Knowledge Discovery and Data Mining*, pages 224–228, 1998.

[26] R. Jörnsten. Clustering and classification based on the L_1 data depth. *J. Multivariate Anal.*, 90(1):67–89, 2004.

[27] R. Jörnsten, Y. Vardi, and C.-H. Zhang. A robust clustering method and visualization tool based on data depth. In *Statistical data analysis based on the L_1-norm and related methods (Neuchâtel, 2002)*, Stat. Ind. Technol., pages 353–366. Birkhäuser, Basel, 2002.

[28] D. G. Kirkpatrick and J. D. Radke. A framework for computational morphology. In G. Toussaint, editor, *Computational geometry*, pages 217–248. North-Holland, 1985.

[29] S. Krishnan, N. H. Mustafa, and S. Venkatasubramanian. Hardware-assisted computation of depth contours. In *13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[30] S. Langerman and W. Steiger. Optimization in arrangements. In *Proceedings of the 20th International Symposium on Theoretical Aspects of Computer Science (STACS 2003)*, 2003.

[31] LAPACK. Linear algebra package. `http://www.netlib.org/lapack/`.

[32] LEDA. Library of efficient data structures and algorithms. `www.ag2.mpi-sb.mpg.de/LEDA`.

[33] A. Lingas. A linear-time construction of the relative neighborhood graph from the Delaunay triangulation. *Comput. Geom.*, 4(4):199–208, 1994.

[34] R. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, 18:405–414, 1990.

[35] R. Liu. Data depth: center-outward ordering of multivariate data and nonparametric multivariate statistics. In M. Akritas and D. Politis, editors, *Recent Advances and Trends in Nonparametric Statistics*, pages 155–168. Elsevier Science, 2003.

[36] R. Liu, J. Parelius, and K. Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *The Annals of Statistics*, 27:783–858, 1999.

[37] J. Matoušek. Computing the center of planar point sets. *DIMACS Series in Disc. Math. and Theoretical Comp. Sci.*, 6:221–230, 1991.

[38] K. Miller, S. Ramaswami, P. Rousseeuw, T. Sellarés, D. Souvaine, I. Streinu, and A. Struyf. Efficient computation of location depth contours by methods of combinatorial geometry. *Statistics and Computing*, 13(2):153–162, 2003.

[39] T. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating extreme points in higher dimensions. In *Symposium on Theoretical Aspects of Computer Science*, pages 562–570, 1995.

[40] F. Preparata and S. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, 1977.

[41] F. P. Preparata and M. I. Shamos. *Computational geometry.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985. An introduction.

[42] E. Rafalin, K. Seyboth, and D. Souvaine. Path length in proximity graphs as a data depth measure. *Tufts CS Technical Report 2005-5*, Tufts University, Nov. 2005. Abstract appeared in *Proceedings of the 15th Annual Fall Workshop on Computational Geometry*, UPenn, 2005, pages 11-12.

[43] E. Rafalin, K. Seyboth, and D. Souvaine. Proximity graph depth, depth contours, and a new multimodal median, 2005. submitted for publication *22nd Annual*

ACM Symposium on Computational Geometry.

[44] P. Rousseeuw and M. Hubert. Depth in an arrangement of hyperplanes. *Discrete & Computational Geometry*, 22:167–176, 1999.

[45] P. Rousseeuw and I. Ruts. Bivariate location depth. *Applied Statistics-Journal of the Royal Statistical Society Series C*, 45(4):516–526, 1996.

[46] P. J. Rousseeuw. Introduction to positive-breakdown methods. In *Robust inference*, volume 15 of *Handbook of Statist.*, pages 101–121. North-Holland, Amsterdam, 1997.

[47] P. J. Rousseeuw. Introduction to positive-breakdown methods. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, Discrete Mathematics and its Applications (Boca Raton), pages xviii+1539. Chapman & Hall/CRC, Boca Raton, FL, second edition, 2004.

[48] P. J. Rousseeuw and M. Hubert. Regression depth. *J. Amer. Statist. Assoc.*, 94:388–433 (with discussion), 1999.

[49] P. J. Rousseeuw and I. Ruts. Constructing the bivariate tukey median. *Statistica Sinica*, 8:827–839, 1998.

[50] P. J. Rousseeuw, I. Ruts, and J. W. Tukey. The bagplot: A bivariate boxplot. *The American Statistician*, 53:382–387, 1999.

[51] P. J. Rousseeuw and A. Struyf. Computing location depth and regression depth in higher dimensions. *Statistics and Computing*, 8:193–203, 1998.

[52] I. Ruts and P. J. Rousseeuw. Computing depth contours of bivariate point clouds. *Comp. Stat. and Data Analysis*, 23:153–168, 1996.

[53] A. Struyf and P. Rousseeuw. High-dimensional computation of the deepest location. Manuscript, Dept. of Mathematics and Computer Science, University of Antwerp, Belgium, 1999.

[54] G. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980.

[55] J. Tukey. Mathematics and the picturing of data. In *Proceedings of the International Congress of Mathematics*, pages 523–531, 1974.

[56] J. W. Tukey. Mathematics and the picturing of data. In *Proc. of the Int. Cong. of Math. (Vancouver, B. C., 1974), Vol. 2*, pages 523–531. Canad. Math. Congress, Montreal, Que., 1975.

[57] Y. Vardi and C.-H. Zhang. The multivariate L_1-median and associated data depth. *Proc. Nat. Acad. Sci. USA.*, 97:1423–1426, 2000.

[58] K. Verbarg. Approximate center points in dense point sets. *Inform. Process. Lett.*, 61(5):271–278, 1997.

[59] E. W. Weisstein. Mode. From MathWorld, `http://mathworld.wolfram.com/Mode.html`.

[60] Y. Zuo and R. Serfling. General notions of statistical depth function. *Ann. Statist.*, 28(2):461–482, 2000.

[61] Y. Zuo and R. Serfling. Structural properties and convergence results for contours of sample statistical depth functions. *Ann. Statist.*, 28(2):483–499, 2000.

Keep Your Friends Close and Your Enemies Closer: The Art of Proximity Searching

David Mount
University of Maryland, College Park
University of Maryland Institute for Advanced Computer Studies

Proximity searching is the general term used for various distance-based search problems in geometric and metric space settings. It includes the well known nearest neighbor problem and its relatives, such as range searching, distance selection, and point location. In spite of many years of research, this field remains a fruitful source of new ideas, new problems, and new computational challenges. It is also one of the success stories of algorithm design, where theory has informed the design of the latest software innovations, and algorithm experimentation has led to new theoretical insights. In this talk, we will survey some recent results in this area and present directions for future research and challenges.

Implementation and Experiments with an Algorithm for Parallel Scheduling of Complex Dags under Uncertainty*

(Extended Abstract)

Grzegorz Malewicz†

Abstract

Our earlier paper introduced a parallel scheduling problem where a directed acyclic graph modeling t tasks and their dependencies needs to be executed on n unreliable workers. Worker i executes task j correctly with probability $p_{i,j}$. The goal is to find a regimen Σ, that dictates how workers get assigned to tasks (possibly in parallel and redundantly) throughout execution, to minimize the expected completion time. The paper provided a polynomial time algorithm for the problem restricted to the case when dag width and the number of workers are at most a constant, and showed necessity of these restrictions, unless P=NP. The current paper describes algorithm engineering approaches used to produce an efficient implementation of the algorithm, and experiments demonstrating how the algorithm scales.

Key words: Parallel scheduling, combinatorial optimization, algorithm implementation, application of perfect hashing, grid computing, project management.

1 Introduction

Grid computing infrastructures have been developed over the past several years to enable fast execution of computations using collections of distributed computers [12]. Among the most important remaining challenges is to achieve efficiency of executing large-scale, sophisticated computations using unreliable computers. These problems are manifested, for example, in the Sloan Digital Sky Survey computations [1] that have sophisticated task dependencies. When a computer fails to correctly execute an assigned task, then the progress of execution may be delayed because dependent tasks cannot be executed pending successful execution of the task. It is conceivable that task dependencies and worker reliabilities play a significant role in the ability to execute a computation quickly. Therefore, one would like to determine relationships among these factors, and develop algorithms for quick execution of complex computations on unreliable computers.

A similar problem arises when managing projects [17] such as production planning or software development. Here a collection of activities and precedence constraints are given. Workers can be assigned to perform the activities. In practice, a worker assigned to an activity may fail to perform it. For example, if an activity consists of writing a piece of code and testing it, it could happen that the test fails. The manager of the project may be able to estimate the success probability of a worker assigned to an activity based on prior experience with the worker. The manager may be able to redundantly assign workers to an activity. For example, two workers may independently write a piece of code and test it; if at least one test succeeds, the activity is completed. Thus the manager faces a problem of how to assign workers to activities, possibly in parallel and redundantly, over the course of the project, so as to minimize the total time of conducting the project.

These two application areas motivate the study of the following fundamental parallel computing scheduling problem. A directed acyclic graph (dag) is given representing t *tasks* and their dependencies. There are n *workers*. At any given unit of time workers are assigned in some way to the tasks that are "eligible" based on precedence constraints and tasks executed thus far (more than one task can be assigned a worker; more than one worker can be assigned to a task; workers can idle). The workers then *attempt* to execute the assigned tasks. The attempt of worker i to execute task j *succeeds* with probability $0 \leq p_{i,j} \leq 1$. In the next unit of time workers are again assigned to tasks. The execution proceeds in this manner until all tasks have been executed. The goal is to determine a regimen Σ, that dictates how workers get assigned to tasks throughout execution, that minimizes the expected completion time.

*Research performed in part during a visit to the Division of Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL 60439 supported by NSF grant ITR-800864, and a stay with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487. Patent pending [21].

†Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA, E-mail: malewicz@google.com

Note that tasks are *not* unit size; instead we are modeling geometric distribution of task duration.

Our prior paper [22] introduced this parallel scheduling problem. The paper shows a polynomial time algorithm that finds an optimal regimen when dag width is at most a constant and the number of workers is also at most a constant. These two restrictions may appear to be too severe. However, they are fundamentally required. Specifically, the paper demonstrates that the problem is NP-hard with constant number of workers when dag width can grow, and is also NP-hard with constant dag width when the number of workers can grow. When both dag width and the number of workers are unconstrained, then the problem is inapproximable within factor less than 5/4, unless P=NP.

Let us sketch how the algorithm works. Say that we are given a dag of task dependencies, and the probabilities of success of workers (an example is in Figure 1(a)). A subset of tasks Y *satisfies precedence constraints* when for any task i in Y, every task j on which i depends is also in Y. We enumerate all subsets $Y_1, \ldots, Y_m$ of tasks that satisfy precedence constraints, and determine pairs of subsets such that Y_j can be obtained from Y_i by executing some eligible tasks in Y_i. These subsets and the "obtainability" relation form, it turns out, a dag, which we call admissible evolution of execution, denoted $\mathcal{A}$ (see Figure 1(b)). The expected time to completion of a regimen starting with tasks Y already executed depends recursively on expectations when starting with the subsets that are the children of Y in $\mathcal{A}$. This suggests a dynamic programming approach. We process a topological sort of $\mathcal{A}$ in the reverse order. When processing a subset Y, we consider all ways in which workers can be assigned to eligible tasks, since it can be shown that workers do not have to idle, and pick an assignment that minimizes expectation according to our recursive equation. After the sort has been processed, we have found a regimen Σ that minimizes expected completion time (see Figure 1(b) and (c)).

It is important to better understand the conditions under which this algorithm produces an optimal regimen in a reasonable amount of time. Indeed, certain critical production planning projects for the military, emergency response team activities, or computational projects may favor an optimal regimen, even when derived at a considerable cost, over a suboptimal one, since the latter may lead to a significant waste of resources that may be costly, or priceless, such as human lives during crisis operations. In such applications, an approximation scheduling algorithm may be inappropriate even though the algorithm may derive a regimen much quicker than an optimal scheduling algorithm. Therefore, it is of practical significance to engineer an efficient implementation of our algorithm, and study how its running time depends on the problem instance.

Contributions. This paper describes an efficient implementation of the algorithm and an evaluation of its scalability. Our implementation first produces a topological sort of $\mathcal{A}$, and then processes the sort. The processing consists of triple nested loops: the outer loop traverses the sort, the middle loop enumerates assignments, and the inner loop evaluates the recursive equation.

Our first contribution is a range of practical approaches that we applied when developing our implementation, with the goal of making the implementation efficient. Our first approach is a method for producing a topological sort of $\mathcal{A}$ in a memory-efficient way, exploiting the structure of $\mathcal{A}$. Specifically, we observe that each arc leads from a subset to a subset with strictly more tasks. Hence we can list subsets of executed tasks of cardinality zero, then of cardinality one, and so on. We show how to create the list without explicitly maintaining arcs, which saves on memory. Our second approach is a fast implementation of the body of the inner loop. That body is the most often performed part of the code. There, a significant contributor to the running time is a static dictionary lookup. We observe that even though the dictionary keys may be large, they are similar in a precise sense. We take advantage of the similarity and design a perfect hash table using a "linear" hash function, so that the hash of similar parts of the keys can be precomputed before the inner loop starts, and only the hash of the differences is computed within the body, thus saving on the time of lookups.

Our second contribution is an experimental evaluation of scalability of the implementation. A user of the implementation may want to judge which instances can be quickly solved. We give one judgment method based on three very simple parameters of the instance: dag width, the number of workers, and the number of tasks. We say that the implementation succeeds on an instance, when the implementation running on a specific dedicated computer can construct an optimal regimen for the instance quickly enough. We explore the landscape of the parameters and estimate where chances of success are high and where they are low, thus outlining the limits of scalability of the implementation. The landscape has subtle shape. Indeed, our implementation can succeed on problem instances with hundreds of tasks, provided that dag width and the number of workers are small; and can succeed on medium size dags with larger width but small number of workers, or with larger number of workers but small width. Note that it is possible that instances with such parameters have occurred

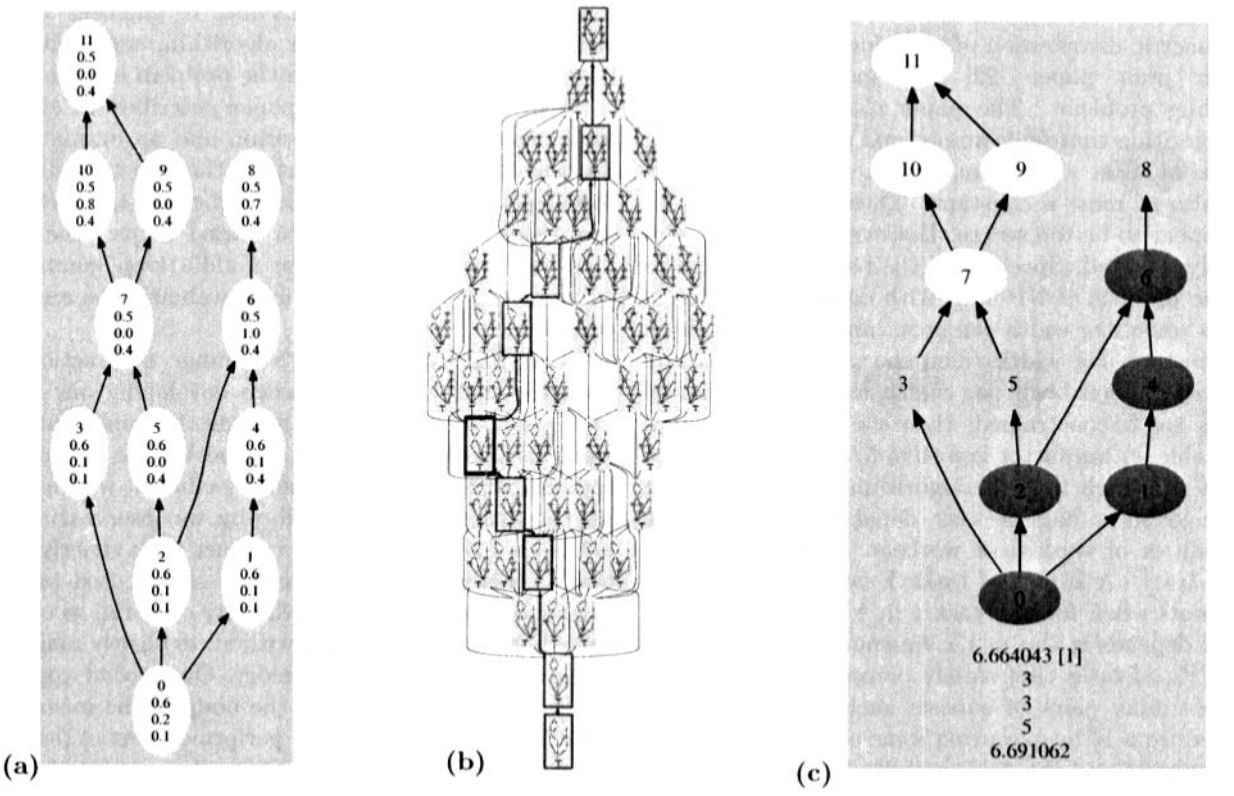


Figure 1: An instance of the scheduling problem and an optimal solution. From left to right: **(a)** 12 dependent tasks with 3 workers and probabilities of success, **(b)** admissible evolution of execution listing, for each node, the minimum expected time to completion and a corresponding assignment of workers, **(c)** a set of executed tasks (darker) and the resulting eligible tasks (lighter); the minimum expected time to completion is 6.664043; worker assignment is: one to 3, two to 3, three to 5; for any other assignment expectation is at least 6.69; this set can be encountered during execution (the darker frame on a path of lighter frames in (b)). Visualized with [15].

or will occur in practice, for example in project management for small businesses. We could, then, claim that our implementation has the potential of being practical.

Related and prior work. Operations Research community studies scheduling under constrained resources [26, 24, 25, 30] and typically offers heuristics for NP-hard problems. In project scheduling under uncertainty [18, 10, 11] a task can be executed by one worker at a time. The restriction can be relaxed and a heuristic approach is offered [31]. Theoretical Computer Science community studies stochastic scheduling where one wants to minimize the expected completion time when task durations are random variables [19, 16, 28]. Any task can be executed by one worker only. There are other sequencing and scheduling problems, with ([14] ND20 and ND21) and without [6] probabilistic failures. Systems community has implemented scheduling algorithms in grid computing (e.g., [29]) and project management fields (e.g., [23]). These implementations often use basic algorithms that may not model the problems accurately, or be less resilient to certain failures. Our implementation of the scheduling algorithm uses a static dictionary. Efficient implementations can use a perfect hash table [13, 7, 9] or a trie data structure (cf. [20]). Our implementation uses a special perfect hash table that exploits algorithm features that may be difficult to achieve with a trie.

2 Background

2.1 Model A computation is modeled by a directed acyclic graph $\mathcal{G}$ with nodes $N_{\mathcal{G}} = \{1, \ldots, t\}$, called tasks, and arcs $A_{\mathcal{G}}$, each having the form $(u \rightarrow v)$, where $u, v \in N_{\mathcal{G}}$. We denote the set $\{1, \ldots, t\}$ by $[t]$. Arcs specify dependencies between tasks: given an arc $(u \rightarrow v)$, v cannot be executed until u has been; task u is a *parent* of v. A set of tasks *satisfies precedence constraints* if, for every task in the set, all parents of the task are also in the set. Given such set X, we

denote by $E(X)$ the set of tasks not in X all whose parents are in X; tasks in this set are called *eligible* when tasks X have been executed (any non-executed source is eligible). A *chain* is a sequence of tasks $u_1, \ldots, u_k$ such that $(u_i \rightarrow u_{i+1})$ is in $A_{\mathcal{G}}$, for $1 \leq i < k$. A set of chains is said to *cover* the dag if every task of the dag is a task in at least one of the chains (chains may "overlap"). An *antichain* is a set of tasks such that no two are "comparable" i.e., for any two distinct tasks u and v from the set, there is no chain from u to v nor from v to u. The largest cardinality of an antichain is called *width* of the dag. A famous Dilworth's Theorem [8] states that dag width is equal to the minimum number of chains that cover the dag.

The execution of tasks is modeled by the following game. There are n *workers* identified with elements of $[n]$. Let X be a set of tasks that satisfies precedence constraints. The game starts with $Y = X$, and proceeds in *rounds*. During a round, workers are assigned to tasks in $E(Y)$ according to a regimen Σ. The regimen specifies an assignment $\Sigma(Y)$ that maps each worker to an element of the set $E(Y) \cup \{\perp\}$ i.e., either to a task that is eligible in this round, or to a distinguished element $\perp$. Note that the assignment is determined by the set of tasks Y. The assignment enables directing multiple workers to the same task, or to different tasks; workers can also idle. Then each worker that was assigned to a task *attempts* to execute the task. The attempt of worker i assigned to task j *succeeds* with probability $0 \leq p_{i,j} \leq 1$ independently of any other attempts. We assume that there is at least one worker that has non-zero probability of success, for any given task. A task is *executed* in this round if, and only if, at least one worker assigned to the task has succeeded. All executed tasks are added to Y, and the game proceeds to the next round. It could be the case that all attempts have failed; then the set Y remains unchanged, and in the next round worker assignment remains unchanged, too. Notice that we are not assuming that each task takes the same amount of time to execute. Rather, we are assuming that each attempt to execute a task takes the same amount of time. Since attempts by different workers may fail with different probabilities, we are modeling a geometric distribution of task durations. Formally, a regimen Σ is a function $\Sigma : 2^{N_{\mathcal{G}}} \rightarrow ([n] \rightarrow (N_{\mathcal{G}} \cup \{\perp\}))$, such that for any subset Z of tasks that satisfies precedence constraints, the value $\Sigma(Z)$ is a function from $[n]$ to the set $E(Z) \cup \{\perp\}$. The game proceeds until a round when all sinks of $\mathcal{G}$ are in Y. We say that the game *ends* in such round.

The quality of the game is determined by how quickly the game ends. Specifically, the number of rounds of the game, beyond the first round, until the round when the game ends is called *time to completion* of regimen Σ starting with tasks X already executed. This time is a random variable. When X is empty (i.e., the game starts with an empty set of executed tasks), we call the time simply *completion time*. Our goal is to find a regimen $\Sigma^\star$ that minimizes the expected completion time. We call this goal the Recomputation and Overbooking allowed Probabilistic dAg Scheduling Problem (ROPAS).

2.2 Algorithm This section outlines an algorithm, called OPT (a pseudocode is given in Figure 2), that solves ROPAS [22]. The algorithm finds an optimal regimen using a dynamic programming approach.

We begin by constructing a topological sort $Y_1, \ldots, Y_m$ of a certain dag $\mathcal{A} = (N_{\mathcal{A}}, A_{\mathcal{A}})$ called *admissible evolution of execution* for $\mathcal{G}$. The dag is produced inductively. Each node of $\mathcal{A}$ will be a subset of nodes of $\mathcal{G}$. We begin with a set $N_{\mathcal{A}} = \{\emptyset\}$. For any node $X \in N_{\mathcal{A}}$ that does not contain all sinks of $\mathcal{G}$, we calculate the set of eligible tasks $E(X)$ in $\mathcal{G}$. We then take all non-empty subsets $D \subseteq E(X)$, add to $N_{\mathcal{A}}$ a node $X \cup D$, if it is not already there, and add to $A_{\mathcal{A}}$ an arc $(X, X \cup D)$, if it is not already there. The dag $\mathcal{A}$ has a single source $\emptyset$ and a single sink $N_{\mathcal{G}}$.

We process the topological sort in the reverse order, while defining a regimen called $\Sigma^\star$. We initialize the regimen arbitrarily. When we process a node X of $\mathcal{A}$, we define two values: a number T_X and an assignment $\Sigma^\star(X)$. Specifically, we begin by setting T_{Y_m} to 0, and $\Sigma^\star(Y_m)$ so that each worker is assigned to $\perp$. Now let $1 \leq h < m$, and let us discuss how T_{Y_h} and $\Sigma^\star(Y_h)$ are defined. Let $D_0, \ldots, D_k$ be all the distinct subsets of $E(Y_h)$, such that $D_0 = \emptyset$. We consider all possible $|E(Y_h)|^n$ assignments of the n workers to the tasks of $E(Y_h)$ (but not to $\perp$). For any assignment, we calculate the probability a_i that D_i is exactly the set of tasks executed by workers in the assignment. Node $Y_h \cup D_i$ has already been processed, for any $i \geq 1$, and so $T_{Y_h \cup D_i}$ is defined. If $a_1 + \ldots + a_k > 0$, then we compute the weighted sum $1/(a_1 + \ldots + a_k) \cdot (1 + \sum_{i=1}^{k} a_i \cdot T_{Y_h \cup D_i})$. We pick an assignment that minimizes the sum. We set T_{Y_h} to the minimum, and $\Sigma^\star(Y_h)$ to the assignment that achieves the minimum. Then we move back in the topological sort to process another node, by decreasing h. After the entire sort has been processed, the regimen $\Sigma^\star$ has been determined. This regimen minimizes the expected completion time.

THEOREM 2.1. ([22]) *The algorithm OPT solves the ROPAS Problem in polynomial time when width of $\mathcal{G}$ is bounded by a constant and the number of workers is also bounded by a constant.*

Data structure: T is a dictionary that maps nodes of $\mathcal{A}$ to distinct floating point variables

```
OPT(t, G, n, (p_{i,j}))
01 let Y_1, ..., Y_m be a topological sort of A
02 T_{Y_m} = 0
03 for h = m − 1 downto 1 do
04     min = ∞
05     for all assignments S of workers to E(Y_h)
06         let I ⊆ E(Y_h) be the assigned tasks
07         sum = 0
08         for all nonempty subsets D ⊆ I
09             let a = Pr[S executes exactly D]
10             sum = sum + a · T_{Y_h ∪ D}
11         let b = Pr[ every assigned worker fails ]
12         if (1 + sum)/(1 − b) < min, then
13             min = (1 + sum)/(1 − b)
14             Σ*(Y_h) = S
15 T_{Y_h} = min
```

Figure 2: An algorithm for constructing an optimal regimen $\Sigma^\star$, for a dag $\mathcal{G}$ describing dependencies among t tasks, and n workers such that worker i executes task j successfully with probability $p_{i,j}$.

THEOREM 2.2. ([22]) *The ROPAS Problem is inapproximable to within less than 5/4 factor, unless P=NP. When restricted to the dag with two independent tasks (where the number of workers may grow), the problem is NP-hard. The problem is also NP-hard when restricted to two workers (where the dag width may grow).*

2.3 Hashing We summarize relevant background on hash functions. Given two sets A and B, a family of *hash functions* (transformations) from A to B is called *c-universal* [3, 27], if for any distinct $x, y \in A$, at most a $c/|B|$ fraction of the hash functions map x and y to the same element of B. If a family is 1-universal, it is called *universal*. We will use three families of hash functions. Any hash function from the first two families maps any subset of $[t]$ to an integer. The families differ by the range of their hash functions. Let $r, s \geq 1$ be fixed integers. Hash functions are indexed by vectors v of length t with coordinates from $V = \{0, \ldots, r \cdot s - 1\}$. For the first family $\mathcal{F}$, each hash function f_v maps any subset to an element of $V = \{0, \ldots, r \cdot s - 1\}$ as prescribed by $f_v(Y) = \left(\sum_{i \in Y} v_i \mod r \cdot s\right)$. For the second family $\mathcal{F}'$, each hash function f'_v maps any subset to an element of $V' = \{0, \ldots, s - 1\}$ as prescribed by $f'_v(Y) = \left(\sum_{i \in Y} v_i \mod s\right)$. Note that $f'_v(Y) = (f_v(Y) \mod s)$. The third family $\mathcal{F}''$ consists of hash functions that map integers to integers. Let p be a prime number at least $r \cdot s$, and $1 \leq b \leq p$. Hash functions from $\mathcal{F}''$ are indexed by numbers g from $\{0, \ldots, p - 1\}$. A hash function f''_g maps elements of V to elements of $\{0, \ldots, b - 1\}$ and is defined as $f''_g(x) = ((g \cdot x \mod p) \mod b)$.

THEOREM 2.3. ([3]) *The families $\mathcal{F}$ and $\mathcal{F}'$ are universal and the family $\mathcal{F}''$ is 2-universal.*

THEOREM 2.4. ([13]) *Fix a c-universal family of hash functions from A to B, such that B has cardinality at least $d \cdot m^2$, and fix a subset of A of cardinality m. Then at least a $(1 - c/(2d))$ fraction of the functions map the elements of the subset to distinct elements of B.*

THEOREM 2.5. ([13]) *Fix a universal family of hash functions from A to B, and a subset of A of cardinality at most $|B|$. For a given function, let c_i be the number of elements of the subset mapped by the function to $i \in B$. Then at least half of the functions have the property that $\sum_{i \in B} c_i^2 \leq 4|B|$.*

3 Engineering the algorithm

This section presents several practical approaches that we used to engineer an efficient implementation of the algorithm OPT.

The first phase of the algorithm constructs a topological sort of $\mathcal{A}$. One approach is to explicitly construct $\mathcal{A}$, and then use a linear time topological sort algorithm (cf. [5]). However, $\mathcal{A}$ often has significantly more arcs than nodes, which slows down the construction, and may cause a memory overflow (thus necessitating an out of core algorithm). We notice that a topological sort can be obtained by listing subsets of executed tasks in the order of cardinalities of the subsets. Hence there is no need to explicitly represent arcs. When listing subsets, we can produce duplicates, but these can be removed in a space-efficient way with the help of a dynamic hash table. We omit details of the construction from this extended abstract.

The second phase processes the sort. The processing typically takes significantly longer than the construction of the sort, especially when the dag width or the number of workers are large. This is due to the triple nested loops in lines 03 to 15 and dictionary accesses in the inner loop. We focus on keeping the number of iterations of the inner loop low, and also keeping low the running time of the body of the inner loop. We outline how we achieve our goal.

The number of iterations of the inner loop may sometimes be reduced. By inspecting the $p_{i,j}$, we can determine which tasks must be executed and which cannot be through S, and only consider the sets D that do not contain such tasks.

The final improvement is to implement the body of

the inner loop well. Here the main cost appears to be the dictionary lookup $T_{Y_h \cup D}$, where $Y_h \cup D$ is a subset of nodes of $\mathcal{G}$. The lookups can be accomplished quickly when we use a carefully crafted perfect hash table, so that the lookup time is on the order of the cardinality of D, rather than the cardinality of $Y_h \cup D$, the latter of which may be quite large, while the former may be fairly small.

We outline a design of a two-level perfect hash table. We use the family $\mathcal{F}$ of hash functions to find a function f_v that maps the sets $Y_1, \ldots, Y_m$ to distinct numbers from $\{0, \ldots, rs-1\}$—s is the smallest power of 2 that is at least m, and $r = 2s$—and the family $\mathcal{F}'$ to find f'_v that maps the numbers to slots of total squares of sizes linear in s. Theorems 2.5 and 2.4 ensure that we can find a desired v. Note that the fact that s is a power of 2 and so is r simplifies the computation of $(x \mod s)$ and also $(x \mod rs)$, since we merely need to take logical "and" of x with appropriate masks. We construct the second level hash table using the family $\mathcal{F}''$. We pick b_i as the smallest power of 2 at least $2 \cdot c_i^2$, and p as the smallest prime at least b_i, for any i, and at least rs. By Theorem 2.4, we can select g_i from $\{0, \ldots, p-1\}$ to obtain $f''_{g_i}(x) = (g_i x \mod b_i)$ that distributes content of the slot i into distinct slots of the second level hash table.

The body of the inner loop is implemented as follows. The hash function f_v is *linear* in the sense that for disjoint X and Y, $f_v(X \cup Y)$ is equal to $(f_v(X) + f_v(Y)) \mod rs$. We can use this fact to reduce the amount of computation inside the loop. Right before we begin executing the inner loop, we compute $f_v(Y_h)$. The loop now considers all subsets D of I, except for the empty one. For a given subset D, the value $f_v(D)$ is computed, then the value of $f_v(Y_h \cup D)$ is computed using linearity, and from this the slot in the first level hash table and then the slot in the second level hash table are determined. We also need to compute the probability a that a given D considered within the body will be executed by S. This computation can be accelerated using a scratch table that for any given task j from I contains the probability $1 - q_j$ that this task will be executed, and another scratch table containing the probability q_j that this task will not be executed.

We observe that the probability a and the value $f_v(Y_h \cup D)$ could be computed "incrementally" from their values from the previous iteration, or the value *sum* resulting from the loop of lines 08 to 10 in a "bottom-up" manner. However, such approach may lead to the accumulation of numerical errors or a more complex control structure.

The algorithm was implemented in C++. The source code contains over 5,000 lines (over 110KB). A discussion of performance improvements due to our algorithm engineering approaches has been omitted from this extended abstract.

4 Scalability experiments

A user of the implementation may want to judge which instances can be quickly solved. This section develops one judgment method. The method is based on three very simple parameters of the instance: dag width, the number of workers, and the number of tasks. We say that the implementation succeeds on an instance, when the implementation running on a specific dedicated computer can construct an optimal regimen for the instance quickly enough. We explore the landscape of the parameters and estimate where chances of success are high and where they are low, thus outlining the limits of scalability of the implementation.

The implementation has exponential running time, and so one could claim that it cannot tackle large problem instances quickly enough. We show here, however, that a response to this claim is subtle. Our implementation can actually succeed on problem instances with hundreds of tasks, provided that dag width and the number of workers are small; and can succeed on medium size dags with larger width but small number of workers, or with larger number of workers but small width. Note that it is possible that instances with such parameters have occurred or will occur in practice, for example in project management for small businesses. We could, then, claim that our implementation has the potential of being practical.

4.1 Experimental setup We overview our judgment approach. Let us fix parameters w, n and t. Certain dags of width w on t tasks may be more typical than others in practice, and so assume that we have a probability distribution of dags with the parameters. The resource consumption of the implementation does not depend much on the $p_{i,j}$, so we assume that $p_{i,j} = 1/2$ for every instance. In this setting, we can find the probability that the implementation running on a randomly selected instance will produce an optimal regimen within a given bound on space and time. We say that our implementation scales within the landscape of triples (w, t, n) where the probability is high. We next discuss details of the approach.

Pseudorandom dags are generated by picking a chain of length t and breaking it in $w-1$ pseudorandom places to obtain w chains. We selected this probability distribution on dags because it is conservative in our setting (see [2] for other distributions). Indeed, for dags occurring in practice (e.g., the AIRSN family [32]) when the width of a dag is w, then the dag can usually be

covered by w chains that overlap. However, our process produces dags with disjoint chains. When chains are disjoint, the graph of admissible evolution of execution tends to have more nodes, and each node X usually has the property that $E(X)$ has about as many tasks as the width of the dag. As a result, the space and time consumption of the implementation should be larger on dags generated through our process compared to practical dags. Hence our estimate of the limits of scalability of the implementation is conservative.

We define the success probability of our implementation. Let $p_{w,n,t}$ be the probability that an instance with parameters w, n and t generated through our process can be computed by our implementation within 120 seconds on our dedicated hardware platform Dell Optiplex GX280 with a 3.4GHz Intel Pentium 4 processor and 4GB of memory and virtual memory disabled, running Windows XP Professional. We compiled the implementation in Visual Studio .NET environment in a release configuration with default optimizations. Our choice of the definition seems to capture the notion of efficiency well given the current state of technology. It may be interesting to define efficiency in a platform-independent manner for the sake of subsequent comparisons with other implementations and algorithms. However, it is also quite important to demonstrate how quick an implementation is on a real hardware and software platform. Hence we focus on our definition.

Our goal is to estimate where the probability is high within the landscape of triples and where it is low. We consider all pairs of w and n with the parameters ranging from 2 to 29. For a fixed pair, we try to find the largest t_{lb} for which $p_{w,n,t}$ is large, and the smallest t_{ub} for which $p_{w,n,t}$ is small. Clearly the probability is monotonically decreasing with t. Therefore, the interval $[t_{lb}, t_{ub}]$ intuitively denotes the values of the number of tasks where a transition occurs i.e., the implementation becomes inefficient.

The search for t_{ub} and t_{lb} is carried out experimentally as follows. We generate 9 pseudorandom instances with parameters w, n and t. If the implementation succeeds on each, then with confidence at least 95% the probability $p_{w,n,t}$ is at least 0.7. Indeed under the null hypothesis that $p_{w,n,t} < 0.7$, 9 Bernoulli trials succeed with probability less than $0.7^9 < 0.05$. Similarly, if 9 pseudorandomly selected instances fail, then with confidence at least 95% $p_{w,n,t}$ is at most 0.3 (see [4] for a discussion of hypothesis testing). The search begins with $t = w$ tasks, and uses expanding and then contracting binary search to determine t_{ub} and t_{lb}.

4.2 Results The results of the experiments are depicted in Figure 3. In the extreme cases, the implementation often succeeds when presented with a small and slim dag of width 2 on 2 tasks but with as many as 28 workers, and also often succeeds with just 2 workers but on a fairly wide dag of width 20 with 20 tasks. The middle range is perhaps more interesting, because it is there where we believe many practical instances will reside. Here success often occurs with fairly large graphs on 351, 120, 45, and 21 tasks, as width and the number of workers increase together from 3 to 6. Width of up to 4 also often admits success for quite large dags when the number of workers is moderate, and when the number of workers is up to 4, then again fairly large dags often succeed with moderate width.

Acknowledgements. The author thanks Ian Foster (Argonne) for hosting his visit at Argonne, Frederica Darema (NSF) for support that facilitated that visit, and David Cordes (UAlabama) for providing an environment that initiated this research.

References

[1] Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S., Foster, I.: Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey. 15th Conference on High Performance Networking and Computing (SC) (2002) 56

[2] Brightwell, G.: Models of random partial orders. Surveys in combinatorics, Cambridge University Press (1993) 53–83

[3] Carter, J.L., Wegman, M.N.: Universal classes of hash functions. Journal of Computer and System Sciences, Vol. 18(2) (1979) 143–154

[4] Cohen, P.R.: Empirical Methods for Artificial Intelligence. Cambridge: MIT Press (1995)

[5] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (2nd Edition). MIT Press (2001)

[6] Crescenzi, P., Kann, V. (eds.): A compendium of NP optimization problems. `http://www.nada.kth.se/~viggo/wwwcompendium/node173.html`

[7] Czech, Z.J., Havas, G., Majewski, B.S.: Perfect hashing, Fundamental Study. Theoretical Computer Science, Vol.182 (1997) 1–143

[8] Dilworth, R.P.: A decomposition theorem for partially ordered sets. Annals of Mathematics, Vol. 51 (1950) 161–166

[9] Farach, M., Muthukrishnan, S.: Perfect Hashing for Strings: Formalization and Algorithms. Combinatorial Pattern Matching (1996) 130–140

[10] Fernandez, A., Armacost, R., Pet-Edwards, J.: Understanding Simulation Solutions to Resource constrained Project Scheduling Problems with Stochastic task Durations. Engineering Management Journal, Vol. 10(4) (1998) 5–13

workers \ width	2	3	4	5	6	7	8	9	10	11
2	1282/1664	354/961	158/334	112/187	79/120	61/101	48/76	43/58	35/50	33/44
3	1288/2328	351/748	164/450	88/237	55/124	49/85	38/56	33/45	28/37	27/36
4	1280/4104	351/612	120/302	58/162	42/75	33/50	26/46	27/33	20/25	19/24
5	1274/1902	313/667	88/258	45/102	31/49	22/35	18/30	16/22	15/17	14/16
6	1278/2046	208/526	62/96	30/62	21/34	15/21	12/18	11/15	10/12	11/11
7	1300/2694	150/321	48/80	22/38	13/21	12/15	10/12	9/9	10/10	
8	1274/2284	105/246	28/56	17/32	10/15	8/10	8/8			
9	1276/2692	69/231	18/42	10/15	7/9	7/7				
10	1120/4482	48/169	12/22	7/10	6/6					
11	800/3078	31/100	8/16	5/7						
12	572/1706	24/48	6/10	5/5						
13	412/1080	16/39	4/6							
14	284/1346	10/19	4/4							
15	196/508	7/12								
16	138/454	4/7								
17	96/198	3/4								
18	68/128	3/3								
19	46/126									
20	32/64									
21	22/54									
22	16/58									
23	10/20									
24	8/18									
25	4/10									
26	4/6									
27	2/4									
28	2/4									
29	2/2									

workers \ width	12	13	14	15	16	17	18	19	20	21
2	31/42	29/33	26/33	26/30	24/28	23/27	22/24	21/23	20/22	21/21
3	24/28	22/26	21/26	20/22	18/20	17/19	18/18	19/19	20/20	
4	16/22	16/19	15/17	15/15	16/16	17/17				
5	13/15	13/13	14/14							
6	12/12									

Figure 3: Scalability of the algorithm demonstrated through confidence intervals for a range of problem parameters. Each cell contains two numbers. The former (if darker) is the number of tasks that yields at least 0.7 probability of success with 95% confidence, unless (if lighter) no success was achieved with that parameter value; the latter at most 0.3 probability of success with the same confidence. Success means that computation takes at most 3 minutes on a dedicated 3.4GHz Pentium processor with 4GB of memory.

[11] Fernandez, A., Armacost, R.L., Pet-Edwards, J.: A Model for the Resource Constrained Project Scheduling Problem with Stochastic Task Durations. 7th Industrial Engineering Research Conference Proceedings (1998)

[12] Foster, I., Kesselman, C. [eds.]: The Grid: Blueprint for a New Computing Infrastructure, 2nd ed. Morgan-Kaufmann, San Francisco, CA (2004)

[13] Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with O(1) worst case access time. Journal of the ACM, Vol. 31(3) (1984) 538–544

[14] Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)

[15] Graphviz - Graph Visualization Software, AT&T Research `http://www.graphviz.org`

[16] Goel, A., Indyk, P.: Stochastic load balancing and related problems. 40th Annual Symposium on Foundations of Computer Science (FOCS) (1999) 579–586

[17] Hillier, F.S., Lieberman, G.J.: Introduction to Operations Research, 8th ed. McGraw-Hill (2004)

[18] Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research, Vol. 165(2) (2005) 289–306

[19] Kleinberg, J., Rabani, Y., Tardos, E.: Allocating Bandwidth for Bursty Connections. SIAM Journal on Computing, Vol. 30(1) (2000) 191–217

[20] Knuth, D.E.: The Art of Computer Programming, Volume 3, Second Edition. Addison-Wesley (1998)

[21] Malewicz, G. (inventor), University of Alabama (assignee): Method and System for Parallel Scheduling of Complex Dags under Uncertainty. Patent pending (2005)

[22] Malewicz, G.: Parallel Scheduling of Complex Dags under Uncertainty. 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA) (2005) 66–75

[23] Microsoft Project 2003 http://www.microsoft.com/office/project/default.asp

[24] Mori, M., Tseng, C.: A Resource Constrained Project Scheduling Problem with Reattempt at Failure: A Heuristic Approach. Journal of the Operations Research Society of Japan, Vol. 40(1) (1997) 33–44

[25] Narasimhan, M., Ramanujam, J.: A fast approach to computing exact solutions to the resource-constrained scheduling problem. ACM Transactions on Design Automation of Electronic Systems, Vol. 6(4) (2001) 490–500

[26] Özdamar, L., Ulusoy, G.: A survey on the resource-constrained project scheduling problem. IIE Transactions, Vol. 27 (1995) 574–586

[27] Pagh, R.: Hash and Displace: Efficient Evaluation of Minimal Perfect Hash Functions. Workshop on Algorithms and Data Structures (WADS) (1999) 49–54

[28] Skutella, M., Uetz, M.: Stochastic Machine Scheduling with Precedence Constraints. SIAM Journal on Computing, Vol. 34(4) (2005) 788–802

[29] Thain, D., Tannenbaum, T., Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, Vol. 17(2-4) (2005) 323–356

[30] Tseng, C.C., Mori, M., Yajima, Y.: A project scheduling model considering the success probability. Proceedings on the Association of Asian Pacific Operational Research Societies (APROS) (1994) 399–406

[31] Turnquist, M.A., Nozick, L.K.: Allocating Time and Resources in Project Management Under Uncertainty. 36th Annual Hawaii International Conference on System Sciences (HICSS) (2003) 250c

[32] Zhao, Y., Dobson, J., Foster, I., Moreau, L., Wilde, M.: A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data. SIGMOD Record (2005)

Using Markov Chains To Design Algorithms For Bounded-Space On-Line Bin Cover

Eyjolfur Asgeirsson * Cliff Stein †

Abstract

We show how the on-line bounded-space bin cover problem can be modeled with a Markov chain. We then use this Markov chain formulation to derive an algorithm for the on-line bounded-space bin cover problem. Our algorithm is designed to perform well in a restrictive environment where it can utilize only very few open bins at each time. We analyze the performance of our algorithm and compare it to the Sum-of-Squares with Threshold algorithm. The experimental results show that our algorithm compares favorably with the Sum-of-Squares with Threshold algorithm, and the average waste incurred by our algorithm is very small even when it is forced to use only a handful of open bins.

1 Introduction

The bin cover problem is, in a sense, a dual problem to the classic bin packing problem. Bin covering takes as input a list of items $L = \{a_1, a_2, \ldots, a_n\}$ with sizes in $(0, 1)$ and places them into bins of unit demand so as to maximize the number of bins that are filled to at least 1. The problem of finding an optimal bin cover is NP-hard so our emphasis will be on designing a good approximation algorithm. We will focus on a restricted class of algorithms, which are *on-line* and *bounded-space*. In an *on-line* setting, the items arrive one at a time and must be assigned to a bin on arrival, without any knowledge of the items that are yet to arrive. An on-line bin cover algorithm is *K-bounded-space* if at no time during its operations does the number of open bins exceed K.

This bounded-space on-line restriction has many practical applications. For example, consider the problem of packing irregular candy pieces into boxes where each box must contain at least 1 lb of candy. The candy pieces arrive one at a time and we must decide in which box to put each piece, while the flowline allows only K open boxes at each time.

Given a list L and an algorithm A, let $A(L)$ be the number of bins filled to at least unit level by A, $OPT(L)$ be the optimal number of bins that can be filled using the items in L and $s(L)$ be the sum of the item sizes in L. Then we have that $s(L) \geq OPT(L) \geq A(L)$.

The worst case performance of bin cover algorithms is usually measured using the following formulas:

$$R^A = \min\left\{\frac{A(L)}{OPT(L)} : \text{ all lists } L\right\}$$

Asymptotic worst case ratio:

$$R^A_\infty = \liminf_{n\to\infty}\left(\min\left\{\frac{A(L)}{OPT(L)} : OPT(L) = n\right\}\right)$$

The R^A_∞ ratio is a better measure of performance for typical applications than R^A. No polynomial-time approximation algorithm A can have $R^A > 1/2$ because of the NP-completeness of distinguishing between instances that can fill two bins opposed to one, whereas in typical applications the number of filled bins is likely to be large.

Another way to measure the performance of bin cover algorithms is to look at the average waste. The objective of minimizing the average waste is equivalent to the objective of maximizing the asymptotic worst case ratio. We define the average waste of algorithm A is defined as:

$$AW^A = \lim_{n\to\infty}\left\{\frac{s(L)}{A(L)} : \text{ all lists } L\right\}$$

When we talk about the expected performance of algorithm A, the average waste is often called *Expected Average Waste* of algorithm A (EAW^A).

A key average-case metric for both bin cover and bin packing is the Expected Waste Rate (EW). Let $L_n(F)$ be a list of n items, where the size of each item is chosen independently according to an item-size distribution F. The Expected Waste Rate for bin covering is defined as:

$$EW^A_n(F) = E[s(L_n(F) - A(L_n(F))]$$

while for bin packing, the Expected Waste Rate is defined as:

$$EW^A_n(F) = E[A(L_n(F)) - s(L_n(F)]$$

*Department of IEOR, Columbia University, New York, NY. Research partially supported by NSF Grant DMI-9970063. ea367@columbia.edu.

†Department of IEOR, Columbia University, New York, NY. Research partially supported by NSF Grant DMI-9970063. cliff@ieor.columbia.edu.

The on-line bounded-space bin cover problem has not been covered extensively in the literature, but less restrictive variants of the bin cover problem have been studied [6, 14]. However, results for bin covering have not been as forthcoming as results for the closely related, but more famous, bin packing problem [1, 4, 3]. Fernandez de la Vega and Lueker [11] introduced an asymptotic PTAS for the off-line bin packing problem in 1981 and in 1982, Karmarkar and Karp [12] gave an improved algorithm. However, for a long time afterwards it was still an open question whether a similar approximation scheme existed for the bin cover variant. This question wasn't answered until 2001, when Csirik, Johnson and Kenyon, [7], gave a PTAS for the off-line bin covering problem. They also looked at the on-line version of bin cover and introduced algorithms, based on a well known Sum-Of-Squares algorithm (SS) for bin packing [8, 9]. These modified SS algorithms perform well for on-line bin cover. In section 4, we will look closely at one of these algorithms, an algorithm called *Sum-Of-Squares with Threshold* (SST), and use it for comparison with our algorithm.

The best known on-line bounded-space bin cover algorithm is also one of the most simple ones, Next Fit (NF). NF has, at all times, only one open bin to which all items are assigned. When the demand of the bin is satisfied, NF closes the bin and opens a new empty bin. Hence, NF is an on-line 1-bounded-space bin cover algorithm. Csirik and Totik [10] have shown that every on-line bin cover algorithm, A, must have $R_\infty^A \leq 1/2$. It is easy to verify that even though NF is a very simple algorithm, it achieves that ratio. However, due to the simplicity of NF, the average case performance of NF is equal to its worst case performance. More sophisticated algorithms can achieve much better average case performance, even though the worst case performance of NF cannot be improved on.

In this paper, we will show how we can model the on-line bounded-space bin cover problem using a Markov chain. We will use this Markov chain to derive an algorithm for the on-line bounded-space bin cover problem. Our focus is on designing an algorithm with a good average case performance while using approximations to help control the exploding size of the state space. To measure the average case performance of our algorithm, we compare it to the Sum-of-Squares with Threshold (SST) algorithm. Our algorithm is designed to perform well in very restricted settings that are similar to real life situations. However, even with the restrictions of bounded-space that our algorithm must satisfy, experimental results show that it does very well when compared to SST.

The input lists that we will focus on have item sizes that are drawn randomly from discrete distributions. The bins have unit demand and the item sizes are integral multiples of $1/B$ for some integer B and the probabilities are rational numbers. By scaling up both the demand of the bins and the item sizes we can equivalently assume that the bins have demand B and the item sizes are integers. Most real-world applications can be scaled to fit this model. The convention is that "polynomial time" can include polynomials in n and B [7]. We will also assume that the item distributions are nontrivial, i.e. distributions where at least two items have positive probabilities.

Of special interest among the discrete distributions are a class of distributions called called perfect packing distributions [2]. These distributions satisfy the property that $EW_n^{OPT} = o(n)$. Courcoubetis and Weber [5] showed that this implies that $EW_n^{OPT} = O(\sqrt{n})$ for bin packing. This property is called the *perfect packing* property. In bin packing, if a distribution F has the perfect packing property then the asymptotic expected ratio of $OPT(L_n(F))$ to $s(L_n(F))$ is 1. It's easy to see that this property holds for bin covering if and only if it holds for bin packing.

2 Definitions and connection to Markov chains

In this section we will define the terms that we use to describe our algorithm and show how we can use Markov chains to analyze the performance of many on-line bounded-space bin cover algorithms. In the next section we will then use the concepts from this section to design such an algorithm.

DEFINITION 2.1. *The* **level** *of a bin is the sum of the sizes of all items in the bin.*

DEFINITION 2.2. *An* **open bin** *is a bin in which items can be placed and whose level is less than its demand. When the level of a bin is greater than or equal to the demand of the bin, the bin becomes* **closed**. *Closed bins cannot accept further items. The difference between the total size of items in a closed bin and the demand of the bin is called* **waste**.

Since we are looking at a K-bounded-space algorithms, we can assume that we have, at all times, K open bins and write the level of these bins as a vector, called *bin-state.*

DEFINITION 2.3. *Assume we have K open bins $b_1, \ldots, b_K$, and bin i has level l_i. Then the vector of bin levels $\sigma = [l_1, \ldots, l_K]$ is called a* **bin-state**. *Since we will only look at bins that are identical, we can reduce the number of possible bin-states by assuming that the bin-states are sorted, i.e. for any bin state $\sigma = [l_1, \ldots, l_K]$*

we have that $l_i \leq l_{i+1}$ *for* $i = 1, \ldots, K-1$. *If the bin-state is not sorted, we will call it an* **unsorted bin-state**. *If all bin levels are less than the demand, the bin-state is called* **open-bin-state**, *otherwise we refer to it as a* **closed-bin-state**.

When an algorithm places an item into a bin such that the demand of the bin is satisfied, the bin is closed and replaced by an empty bin. This means that the closed-bin-states that we will use will have exactly one closed bin. Since the bin-states are sorted, the closed bin will always be the last bin in the bin-state.

We can use the following lemma to determine the total number of different bin-states.

LEMMA 2.1. *Let the demand of each bin be* B *and the sizes of the items be integers. Then for a* K*-bounded space algorithm, there are at most* $M = \binom{B+K-1}{K} + (B-1)\binom{B+K-2}{K-1}$ *possible bin-states.*

Proof. Let $\mathcal{S}$ be the be the set of all possible bin-states. Then we have $\mathcal{S} = \{\sigma_0, \ldots, \sigma_{M-1}\}$ where $\sigma_i = [l_1^i, \ldots, l_K^i]$, l_j^i is the level of the j-th bin in the sorted bin-state σ_i and $l_j^i \leq l_{j+1}^i$ for all $j = 1, \ldots, K-1$. The total number of different open-bin-states is equal to the number of different multisets of size K from a set of B elements, or $\binom{B+K-1}{K}$.

Since the size of the largest item is B, the last bin in a sorted bin-state has $B-1$ possible bin levels, i.e. bin levels from B to $2B-1$. The number of possible bin-level combinations for the first $K-1$ bins is equal to the number of different multisets of size $K-1$ from a set of B elements, or $\binom{B+K-2}{K-1}$. The total number of closed-bin-states is then $(B-1)\binom{B+K-2}{K-1}$ and the total number of bin-states is $\binom{B+K-1}{K} + (B-1)\binom{B+K-2}{K-1}$. ■

Of all these bin-states, the bin-state σ_0 has special significance since it is the initial bin-state where all bins are empty.

Since there are only a fixed number of possible bin-states for on-line bounded-space bin cover algorithms, we can use Markov chains to analyze the performance of such algorithms when the item distribution is known, provided that the item distribution and the algorithms satisfy the following conditions:

1. The algorithm does not change behavior based on the number of closed bins or on the items seen, i.e. it only depends on the current bin-state and the size of the current item.

2. The item distribution is fixed and nontrivial, i.e. it does not change over time and at least two item have positive probabilities.

If these conditions are satisfied for an algorithm A we can write A as a Markov chain. Each bin-state that algorithm A can encounter becomes a state in the Markov chain. We will overload the syntax and let the bin-state σ_i denote both the bin-state of algorithm A and the corresponding state in the Markov chain.

Let $\gamma^A(\sigma_i, j, \sigma_{i'})$ be an indicator variable that is equal to 1 if algorithm A moves to bin-state $\sigma_{i'}$ whenever it is in an open-bin-state σ_i and receives an item of size j. Then the transition probability from open-bin-state σ_i to state $\sigma_{i'}$, $p(\sigma_i, \sigma_{i'})$, is calculated as follows:

$$p(\sigma_i, \sigma_{i'}) = \sum_{j \in J} p_j \, \gamma^A(\sigma_i, j, \sigma_{i'})$$

where J is the set of possible item sizes, $J = [1, 2, \ldots, B-1]$ and p_j is the probability of item of size j.

The states in the Markov chain that corresponds to closed-bin-states have a transition probability of 1 to the open-bin-state where the closed bin has been replaced with an empty bin. Since the state space is finite, there will be at least one recurrent class in the Markov chain. If there are some transient states then we can simply remove them since we are only interested in the long run performance of the algorithm. The algorithms for on-line bounded-space bin-cover are not likely to create Markov chains with multiple recurrence classes, but if that happens we can look at each recurrence class in isolation and then look at the probability of the Markov chain being in each recurrence class.

We now focus on the positive recurrent class of the Markov chain. Since any algorithm with only one open bin is trivially Next Fit, we will only consider instances where $K > 1$. The item distribution is nontrivial so this class will be aperiodic. Markov chains that have only one class, positive recurrent and aperiodic have limiting probabilities [13] that determine the mean time spent in any state. These limiting probabilities allow us to calculate exactly the expected waste of the algorithm. Let π_i be the long-run proportion of time that the Markov chain is in state i and let S_c be the set of states in the Markov chain that correspond to closed-bin-states. Then the expected average waste of algorithm A is:

$$EAW^A = \frac{\sum_{i \in S_c} \pi_i W_i}{\sum_{i \in S_c} \pi_i}$$

where W_i is the actual waste of the closed bin in the closed-bin-state i.

If the Markov chain has multiple recurrence classes, we can calculate the expected average waste of the algorithm by taking a weighted average of the expected

average waste for each recurrence class, where the weight of each such class is based on the probability that the Markov chain will end up in that class.

3 Using Markov chains to design an algorithm

Analyzing the expected performance of an on-line bounded-space bin cover algorithm for a specific item distribution using Markov chains might be useful in some situations, but we would like to use the idea behind this method to design such algorithms. Observe that we can completely describe an on-line bounded space bin cover algorithm by the parameters $\gamma^A(\sigma_i, j, \sigma_{i'})$, i.e. the parameters that effectively determine into which bin we place an item of size j for all possible bin-states that can occur. A helpful notation is to let $\sigma_i^{j\to k}$ be the bin state that we get if the current bin-state σ_i and we place an item of size j into bin number k. (Recall that the bins are stored in sorted order.)

We can now write the formulation for an on-line K-bounded-space bin cover algorithm as:

$$\begin{aligned}
\min \quad & EAW = \frac{\sum_{i\in S_c} \pi_i W_i}{\sum_{i\in S_c} \pi_i} \\
\text{s.t.} \quad & \sum_{k=1}^{K} \gamma(\sigma_i, j, \sigma_i^{j\to k}) = 1 \quad \forall \sigma_i \in S, j \in J \\
& 0 \le \gamma(\sigma_i, j, \sigma_{i'}) \le 1 \\
& \gamma(\sigma_i, j, \sigma_{i'}) = 0 \qquad \forall \sigma_{i'} \notin \{\sigma_i^{j\to 1}, \ldots, \sigma_i^{j\to K}\}
\end{aligned}$$

where S is the set of all states in the Markov chain and J is the item distribution. However, in general, this formulation does not immediately yield an algorithm. Even though the optimal solution to this problem would give us an optimal algorithm for the on-line K-bounded-space bin cover problem, this formulation cannot be solved efficiently because of the complicated connection between the $\gamma(\sigma_i, j, \sigma_{i'})$ variables and the limiting probabilities π.

3.1 Approximate solution Let us assume that we have a ranking criteria for all the closed-bin-states, i.e. some cost that determines how desirable each closed-bin-state is. We will define this in detail later but for now, let's just call this cost *bin-state-cost*, or *bsc*.

Given the bin-state-cost for all closed bin-states, we can calculate the expected bin-state-cost for any open-bin-state σ_i in a recursive manner:

$$E[\text{bsc}(\sigma_i)] = \sum_{j\in J} p_j E[\text{bsc}(\sigma_i^{j\to k^*})]$$

where p_j is the probability of item of size j, k^* is the bin that we will place an item of size j into if the current bin-state is σ_i and $\sigma_i^{j\to k^*}$ is the resulting bin-state after the item has been placed in bin k^*.

ALGORITHM 3.1. BSC
Input: Sorted list of all states, probability of each item size, list of items.
Output: Packing of all the items into bins.

Initialize:
For state $\sigma_i = \sigma_M \ldots \sigma_0$
Calculate $EBSC(\sigma_i)$

Packing items:
$\sigma_{current} \leftarrow \sigma_0$
For each item j in the list of items
  $k^* \leftarrow -1$
  $c_{\min} \leftarrow \infty$
  for each bin k
    if $(EBSC(\sigma_{current}^{j\to k}) < c_{\min})$
      $c_{\min} \leftarrow EBSC(\sigma_{current}^{j\to k})$
  $k^* \leftarrow k$
  place item j in bin k^*
  $\sigma_{current} \leftarrow \sigma_{current}^{j\to k^*}$

Figure 1: The Min-Bin-State-Cost (BSC) algorithm. The parameter k^* is used both in the initialization step and when we are packing the items. We could store the values of k^* for each bin-state and each item size during the initialization. However, due to the storage requirements of such a table and since it is easy to find k^*, we use a simple comparison of k bin-states to find k^* for each item when we are packing the items.

LEMMA 3.1. *For any bin-state σ_i and any item j and any bin k in σ_i, the bin-state $\sigma_i^{j\to k}$ is always later in a lexicographic order than σ_i. This still holds when $\sigma_i^{j\to k}$ is a closed-bin-state.*

Proof. Let $\sigma_i = [l_1, \ldots, l_k, l_{k+1}, \ldots, l_{k+r}, \ldots, l_K]$ and $\sigma_i^{j\to k} = [l_1, \ldots, l_{k+1}, \ldots, l_{k+r}, l_k + j, \ldots, l_K]$. Then since $l_k \le l_{k+1}$ and $l_k + j > l_{k+r}$ for any $r \in \{0, \ldots, K-k\}$, while all other elements are identical, we have that $\sigma_i < \sigma_i^{j\to k}$. ∎

COROLLARY 3.1. *Given the bin-state-cost of all closed-bin-states, we can calculate the bin-state-cost of all bin-states in one pass by iterating through the bin-states in reverse lexicographic order.*

Once we define the bin-state-cost, we can use Corollary 3.1 to create the BSC algorithm shown in Figure 1.

We define the cost for each closed-bin-state in the following manner. The cost has two parts:

- The actual waste that we incur from the closed bin

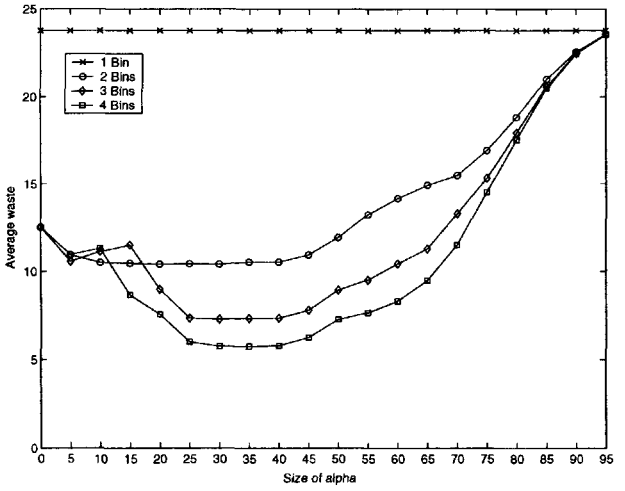


Figure 2: Effect of α on average waste incurred by the BSC algorithm. The bin size was 100, the items sizes were integers obtained by rounding from $N(50, 7)$.

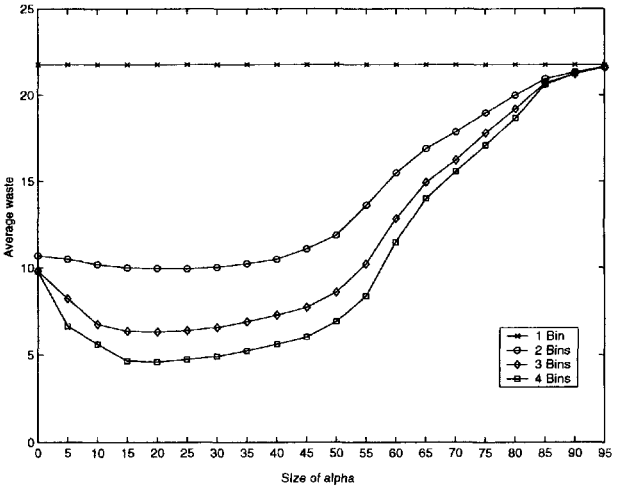


Figure 3: Effect of α on average waste incurred by the BSC algorithm. The bin size was 100 and the item distribution was discrete Uniform $(15, 65)$.

- A penalty for all open bins whose level is too high, i.e. if they are unlikely to be closed without incurring a large waste.

We set the penalty for each open bin as $\max(0, h + \alpha - B)$, where h is the level in the bin and α is a number such that the probability of receiving an item no greater than α is small. The recursive formula for the expected bin-state-cost then becomes

$$E[\text{bsc}(\sigma_i)] = \begin{cases} W_K + \sum_{k=1}^{K-1} \max(0, l_k + \alpha - B)) \\ \quad \text{if } \sigma_i \text{ is a closed-bin-state;} \\ \\ \sum_{j \in J} p_j E[\text{bsc}(\sigma_i^{j \to k^*})] \\ \quad \text{if } \sigma_i \text{ is an open-bin-state.} \end{cases}$$

where W_K is the actual waste of the closed bin (the last bin) in the closed-bin-state. The optimal value for α depends on the item distribution and the number of bins that we can have open. Figures 2 and 3 show how the choice of α affects the performance of the *BSC* algorithm. We often refer to his constant α as a *penalty item*, since we are effectively penalizing the bins that are too full.

3.2 Groups When the number of open bins grows, using a Markov chain to design the algorithm quickly becomes impractical since the number of possible states grows very rapidly with the number of open bins.

To be able to use many bins, we partition the open bins into groups such that each group has a manageable number of bins. We then find a BSC algorithm for each group of bins. Then, when we pack items we must do it in two steps, first we select which group we will use for the item, and then in which bin from that group we will place the item.

We tried two different greedy methods to select into which group we place each item. Let $G = \{1, \ldots, q\}$ be the set of groups and σ_i be the bin-state of group i. Also let $\text{bsc}(\sigma_i)$ be the expected bin-state-cost of σ_i and let k^* be the bin in group i that we will use for an item of size j according to the BSC algorithm. We then tried the two following greedy methods.

Aggressive Greedy:

$$\text{argmin}_{\sigma_i \in G} E[\text{bsc}(\sigma_i^{j \to k^*})]$$

Conservative Greedy:

$$\text{argmax}_{\sigma_i \in G} \left(E[\text{bsc}(\sigma_i)] - E[\text{bsc}(\sigma_i^{j \to k^*})] \right)$$

The aggressive greedy method tries to minimize the best expected bin-state-cost over all the groups while the conservative greedy method tries to maximize the improvement in expected bin-state-cost we can get for any group by adding the item to that group.

Figures 4 and 5 show comparisons between these two methods. In our experiments, the conservative greedy was often much better than the aggressive greedy method whereas in the few instances where the aggressive greedy performed better, the difference was very small. Our results indicate that when choosing into which group we place an item, it is preferable to avoid getting into trouble rather than selecting the most attractive option for each item.

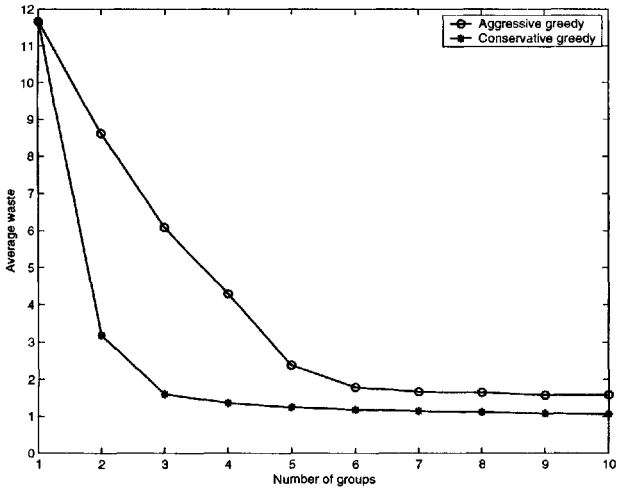


Figure 4: Greedy comparison. Bin size of 600 and an item distribution of Normal(100,15). The number of groups is from 1 to 10 where each group consists of 3 bins.

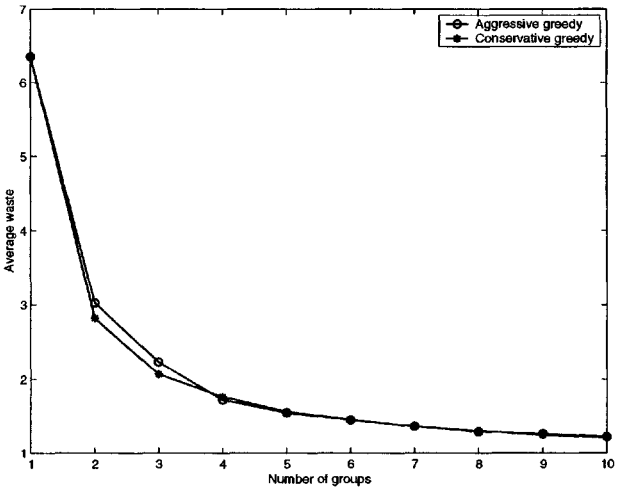


Figure 5: Greedy comparison. Bins of size 100 and the item distribution is Uniform(15,65). The number of groups is from 1 to 10 where each group consists of 3 bins.

3.3 Decreasing the number of states As we mentioned at the beginning of Section 3.2, the number of states grows rapidly with the demand of the bins. Since the demand is an integer, the number of different bin levels that a single bin can have is equal to the demand. We will now show how we can ignore a few bin levels to decrease the number of states in the Markov chain, and thereby reduce the running time of the algorithm.

DEFINITION 3.1. *If bin-states* $\sigma_i = (l_1^i, \ldots, l_K^i)$ *and* $\sigma_j = (l_1^j, \ldots, l_K^j)$ *are such that* $\sum_{k=1}^{K} |l_k^i - l_k^j| = 1$ *then we say that the bin-states* σ_i *and* σ_j *are* **adjacent**.

Figures 6 and 7 show the expected bin-state-cost of all bin-states of the BCS algorithm for two different item distributions and a bin size $B = 100$. The first item distribution is a normal distribution $N(25, 5)$ where we round the items to the nearest integer. The second is a discrete uniform distribution between 10 and 40. Usually, we only look at the sorted bin-states, but here we plot all bin-states to get a better picture.

In Figures 6 and 7, we have 100 bin levels and two bins, for a total of 5050 sorted open-bin-states. The steep edges when either bin has level close to 100 are due to the α-factor kicking in when the bin levels become too high. The size of α for the first one was set to 10 which is 3 standard deviations below the mean, and the second one also had α of size 10 which is equal to the smallest possible item we can get from that distribution.

However, they both support the intuition that, for the majority of the bin-states, there should only be a small difference in the bin-state-cost of adjacent bin-states. When the bin-levels are not at some critical levels based on the item distribution and the bin demand, the expected bin-state-cost will change slowly between adjacent bin-states.

To decrease the number of bin-states we do the following:

- If the item distribution has a smallest item, i.e. no items are smaller than β, we ignore all bin levels between 0 and β. We can do this since it is impossible for any bin to have level between 0 and β.
- Select a subset, L, of the remaining bin-levels. We try to minimize the size of L, while still making sure that L captures the variations in bin-state-costs for the bin-states.

When we now run the BSC-algorithm, we calculate the expected bin-state-cost as before, but in each step, we round the actual bin-state to the nearest bin-state that consists only of bin-levels from L.

To see the effect of using only a subset of the bin-levels, we created four different subsets, with bin demand of 100 and a set of item distributions where the distribution is uniform between 18 and X where $X = 21, \ldots, 40$. We used the same subsets for all 19 distributions. We ran the BSC-algorithm for each subset of bin levels and each item distribution using 3 open bins. The subsets are shown on the left graph in Figure 8. The first subset is as detailed as possible

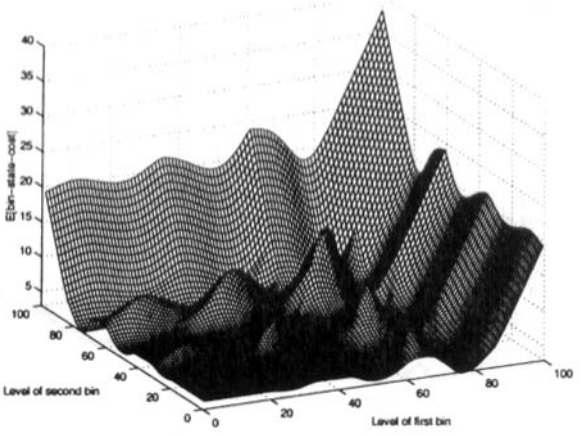


Figure 6: The expected bin-state-cost with two bins of demand 100, an item distribution of $N(25,5)$ rounded to the nearest integer and a penalty item of size 10.

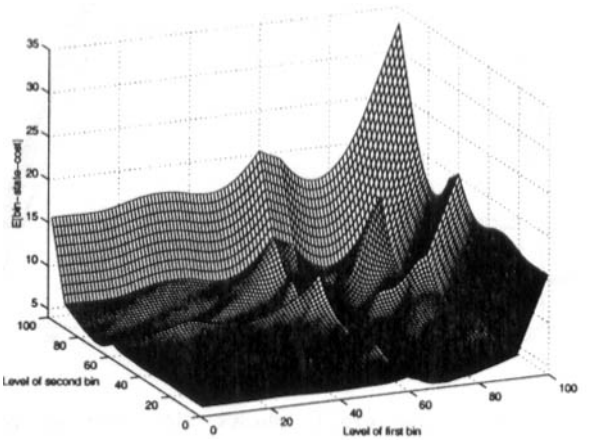


Figure 7: The expected bin-state-cost with two bins of demand 100. The item distribution was discrete uniform between 10 and 40 and a penalty item of size 10.

Size of subsets	3 Bins	4 Bins	5 Bins
100 levels	171,700	4,421,275	91,962,520
83 levels	98,770	2,123,555	36,949,857
42 levels	13,244	148,995	1,370,754
29 levels	4,495	35,960	237,336
17 levels	969	4,845	20,349

Table 1: Number of different sorted open-bin-states for various sized subsets and number of open bins.

with 83 different bin-levels, we only omit the bin levels $1,\ldots,17$, since the smallest item is 18, we will never encounter those. The next subset uses every other bin-level between 18 and 99, this gives us a subset of size 42. The third set has size 29 while the last set uses only 17 bin-levels by jumping in steps of 5 between 18 and 99. Figure 8 show the comparison of the algorithm using these 4 different sets on 19 uniform distributions.

Figure 8 shows that we can decrease the number of bin-states significantly without severely affecting the performance. The average waste incurred does not increase significantly if we use 42 levels rather than 83, however, as Table 1 shows, the complexity decreases by almost an order of magnitude.

Table 1 shows the number of different sorted bin states for 3,4 and 5 open bins, using 5 different sized subsets of the bin levels. As we can see, the number of bin-states decreases dramatically when we decrease the size of the subset. Notice that, for 3,4 and 5 bins, we can reduce the number of bin-states by a factor of 1.7, 2.1 and 2.5 respectively, just by removing the unnecessary

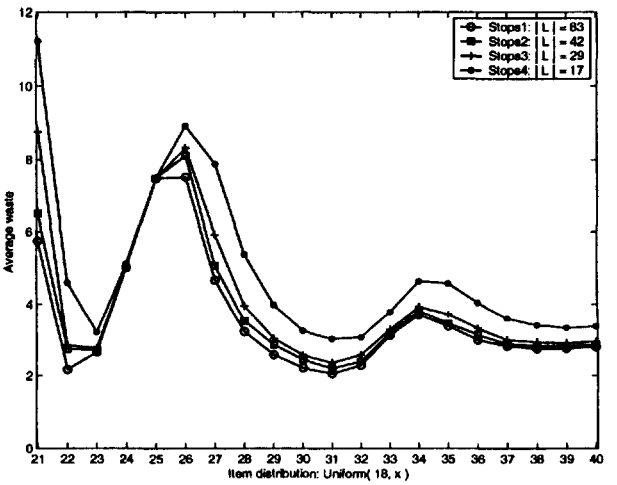


Figure 8: Comparison of using different sized subsets of the bin levels. The sizes of the subsets are 83,42,29 and 17, and the item distribution is uniform between 18 and X, where $X = [21,\ldots,40]$. The size of α was 18 and we used 3 open bins. The results are averages taken over 5 instances.

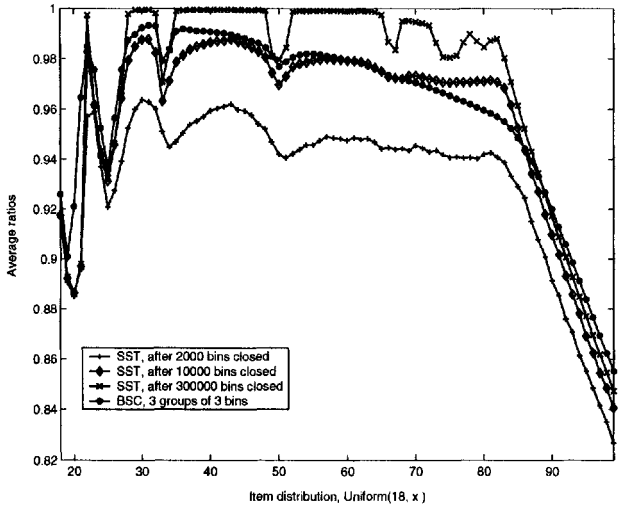


Figure 9: This graph shows the average ratio $A(L_n(F))/s(L_n(F))$ for the SST algorithm and the BSC algorithm. The values for the SST algorithm are shown after it has closed 2000, 10000 and 300000 bins. The BSC algorithm uses 3 groups of 3 bins for a total of 9 open bins. The values for the BSC algorithms are evaluated after closing 100000 bins. The item distributions shown are uniform distribution between 18 and all values from 18 to 99.

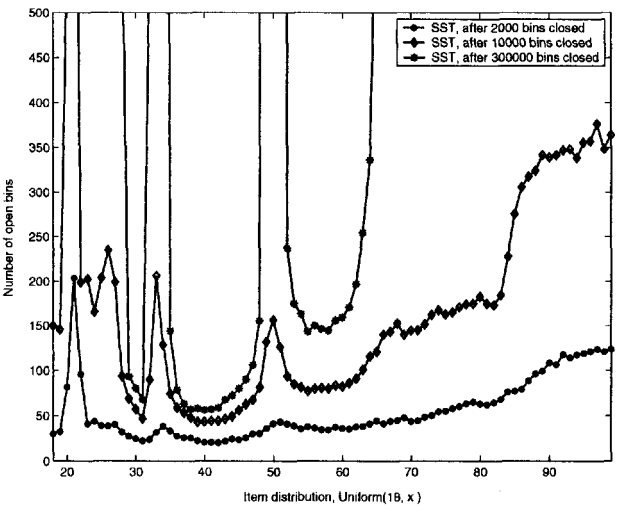


Figure 10: This graph shows how many bins the SST algorithm is using after it has closed 2000, 10000 and 300000 bins. The values for the BSC algorithms are evaluated after closing 100000 bins. Since the BSC algorithm uses only 9 open bins, we only show values between 0 and 500 for the SST algorithm.

bin-levels between 0 and 18. Sorting the bin-states is crucial, the number of unsorted bin states with 100 bin levels is 1,000,000 for 3 open bins, 100,000,000 for 4 open bins and 10,000,000,000 for 5 open bins.

Since the number of possible bin-states is a bottleneck on how many bins we can keep open, using a small set of bin-levels allows us to increase the number of open bins. Using a smaller set of bin-levels will increase the average waste incurred, but having more open bins will decrease the average waste. Hence there is a tradeoff between the size of the set of bin-levels we use and the number of open bins.

4 Comparison with Sum Of Squares with Threshold

The Sum-of-Squares algorithm (SS) is a well known algorithm that performs well for on-line bin packing. The SS algorithm keeps track of how many bins, n_i, have bin-level equal to i in the current packing. When a new item is packed, it is placed in a bin so as to minimize $\sum_{i=0}^{B-1} n_i^2$, subject to the constraint that no bin is overfilled. The SS-algorithm has $EW_n^{SS}(F) = O(\sqrt{n})$ for all perfect packing distributions F, so it performs well on such distributions for the on-line bin cover problem. However, if the distribution does not satisfy the perfect packing property, then SS can fail miserably since it never considers overfilling a bin, which is often necessary for bin covering. Csirik, Johnson and Kenyon [7] showed how to modify the SS algorithm to make it useful for on-line bin covering with general distributions. The version that they introduced and we will consider here is called Sum-of-Squares with Threshold, SST, where they allow the SS algorithm to fill a bin up to a threshold T, where at each time interval, T depends on the number of bins closed so far and the sum of the items seen so far.

The SST algorithm has a definite advantage over our algorithm since it is not restricted by the bounded-space condition. This means that there are no restrictions on how many bins SST can keep open at each time. When the distribution is a perfect packing distribution, the average waste of the algorithm will get arbitrarily close to 0 as the number of items increases. To get a relevant comparison between the BSC algorithm and SST, we will run them both for a long time, and look at how the average waste of both algorithms changes over time and how many bins SST must keep open to achieve its performance.

Figure 9 looks at the ratio of $A(L_n(F))/s(L_n(F))$ for both the algorithms, while Figure 10 shows the number of open bins that the SST algorithm is using.

Figures 9 and 10 indicate that using 9 open bins for the BSC algorithm gives similar results to the

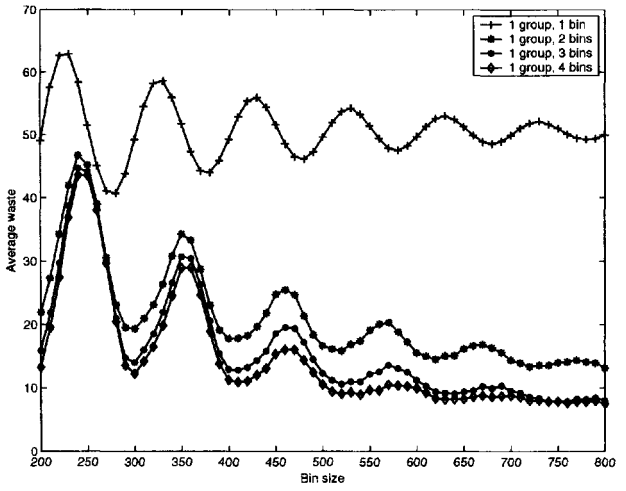


Figure 11: The performance of the BSC algorithm with item distribution N(100,15) for bin sizes from 200 to 800. This graph shows the performance for a single group of 1,2,3 and 4 bins.

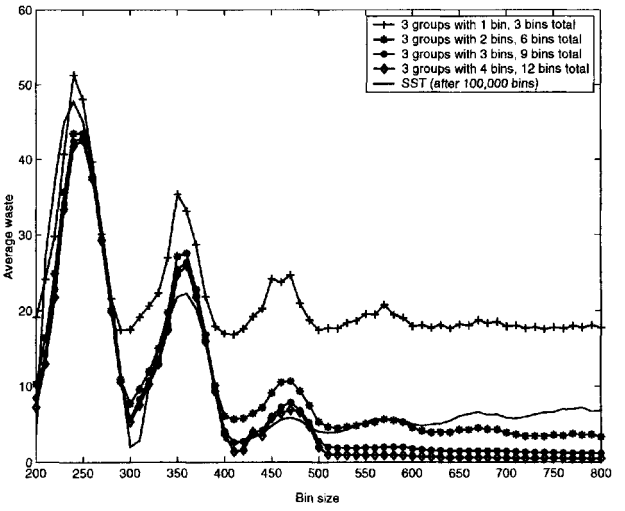


Figure 12: The performance of the BSC algorithm with item distribution N(100,15) for bin sizes from 200 to 800. This graph shows the performance of three groups of 1,2,3 and 4 bins, for a total of 3,6,9 and 12 bins, and the performance of Sum of Squares with Threshold algorithm after closing 100,000 bins.

SST algorithm after closing 10,000 bins, while the SST algorithm uses at least 4 times as many open bins and up to 54 times as many. When the item distribution is very difficult, the SST algorithm takes a long time to reach a good average ratio. For item distributions that are uniform between 18 and an upper limit larger than 82, the item distribution no longer satisfies the perfect packing property and is very difficult to pack. In those cases, our algorithm actually performs better than SST after 300,000 closed bins. In the long run, the SST algorithm performs much better than the BSC algorithm, but for environments where the number of open bins must be relatively small and items cannot wait in open bins indefinitely, the BSC algorithm has definite advantages over the SST algorithm.

5 Results and experiments

We looked at the performance of the BSC algorithm when faced with a number of different bin sizes. We kept the item distribution fixed as normal distribution with mean 100 and standard deviation 15. We chose the normal distribution since the distribution of items in a practical setting, e.g. in the food industry, is often quite close to the normal distribution. We look at bin sizes between 200 and 800, and for each bin size we create a special subset of the bin levels that we use. To speed up the performance, we try to keep the number of bin levels in each such subset no more than 100. Figure 11 shows the performance of using only a single group of 1,2,3 and 4 open bins. Figure 12 shows the results if we use three groups of 1,2,3 and 4 bins. The graph in Figure 13 shows the performance of 1,2,3 and 4 groups of 3 bins, with a total number of open bins as 3,6,9 and 12.

Small average waste using very few open bins. By using only 9 open bins we get the average waste below 2 for bin sizes from 600 to 800 using item distribution N(100,15). With only 12 open bins, we can get an average waste well below 1 for bin sizes from 600 to 800, and less than 0.5 for the largest bin sizes. The ratio $A(L_n(F))/s(L_n(F))$ for bin size 800 and average waste 802 is around 0.9975, and for bin size 600 and average waste 602, the ratio is .9967. Hence, by using only 9 open bins, we consistently get a ratio above 0.996 for bin sizes above 600. With 12 open bins, this ratio is greater than 0.998 for bin sizes above 600.

Using 12 bins gives results that are comparable to SST for small bin sizes When the bin sizes are small, it is often difficult to get small average waste, since we can only fit a few items into each bin. We ran the SST algorithm as comparison with the BSC algorithm, allowing SST to close 100,000 bins. For small bin sizes, between 200 and 500, we got very similar results from the BSC algorithm with 3 groups of 4 bins as the SST algorithm got after closing 100,000 bins. The SST algorithm used from 100 to 14,000 open bins, while the BSC algorithm used only 12. When the bin sizes were above 600, the BSC algorithm was much better

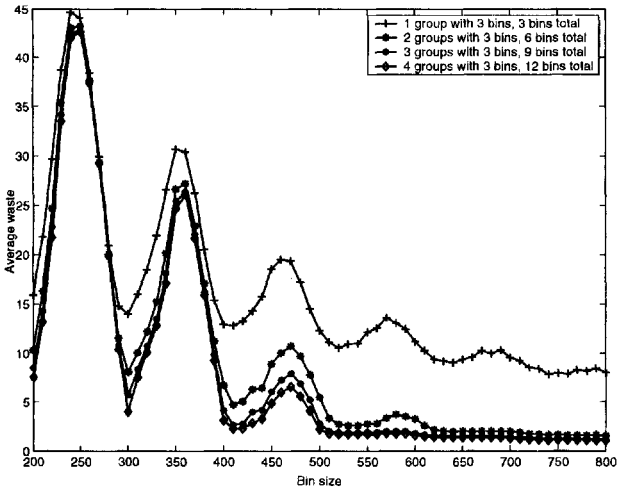


Figure 13: The performance of the BSC algorithm with item distribution N(100,15) for bin sizes from 200 to 800. This graph shows the performance of 1,2,3 and 4 groups where each groups consists of three bins.

than the SST algorithm since the SST algorithm did not converge fast enough when the bin sizes grew larger.

Running time for packing items is small. We ran the instances shown in Figures 11,12 and 13 on a laptop with a 1.7GHz Pentium-M processor and 512MB memory. Running time for BSC when there are 3 bins in a group is around 5 seconds for the smaller instances to 20 seconds for the largest instances. This is the time that it takes to calculate bin-state-cost for all bin-states and fill 100,000 bins. When there are 4 bins in a group, the running time is around 2 minutes for the smaller problem up to 10 minutes for the largest instances. The running time is dominated by the time it takes to set up and calculate the bin-state-cost for all the bin-states. Once the expected bin-state-cost is calculated, packing the items is very fast.

6 Conclusions

Markov chains can be a powerful tool when dealing with algorithms that have a closed and finite search space. By formulating the problem in terms of a Markov chain we can get new insights and ideas on how to solve it or for algorithm design. We showed how ideas from Markov chains helped us design an algorithm for the on-line bounded-space bin covering problem. This algorithm was called BSC-algorithm. We showed that the BSC algorithm performs very well with various item distributions and bin sizes, while using only very few open bins. In experiments designed to simulate practical settings, the BSC algorithm performed much better than the Sum-of-Squares with Threshold algorithm. The BSC algorithm has some very nice qualities and should perform well in a practical setting.

Acknowledgments. The authors thank Agni Asgeirsson and Pall Jensson for helpful discussions.

References

[1] A. B. Assman, D. J. Johnson, D. J. Kleitman, J. Y-T. Leung, *On a dual version of the one-dimensional bin packing problem*, Journal of Algorithms, 5 (1984), pp. 502–525.

[2] E. G. Coffman, Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, M. Yannakakis, *Perfect Packing Theorems and the Average Case Behavior of Optimal and On-line Packings*, SIAM Rev., 44 (2002), pp. 384–402.

[3] E. G. Coffman, Jr., D. S. Johnson, P. W. Shor, R. R. Weber, *Markov chains, computer proofs, and average-case analysis of best fit bin packing*, STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, (1993), pp. 412–421.

[4] E. G. Coffman, Jr., G. S. Lueker, *An Introduction to the Probabilistic Analysis of Packing and Partitioning Algorithms*, Wiley & Sons, New York, (1991).

[5] C. Courcoubetis, R. R. Weber, *Stability of on-line bin packing with random arrivals and long-run average constraints*, Prob. End. Inf. Sci.,4 (1990), pp. 447–460.

[6] J. Csirik, J. B. G. Frenk, G. Galambos, A. H. G. Rinooy Kan, *Probabilistic analysis of algorithms for dual bin packing problems*, Journal of Algorithms, 12(2) (1991), pp. 189–203.

[7] J. Csirik, D. Johnson, C. Kenyon, *Better approximation algorithms for bin covering*, Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms, (2001), pp. 557–566.

[8] J. Csirik, D. Johnson, C. Kenyon, P. Shor, R. Weber, *A self organizing bin packing heuristic*, Proceedings, ACM-SIAM Workshop on Algorithm Engineering and Experiments, (1999), pp. 246–265.

[9] J. Csirik, D. S. Johnson, C. Kenyon, J. B. Orlin, P. W. Shor, R. R. Weber, *On the sum-of-squares algorithm for bin packing*, Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, (2000), pp. 208–217.

[10] J. Csirik, V. Totik, *Online algorithms for a dual version of bin packing*, Discrete Appl. Math., 21(2) (1988), pp. 163–167.

[11] W. F. de la Vega, G. S. Lueker, *Bin Packing can be Solved within* $1+\epsilon$ *in Linear Time*, Combinatorica,1(4) (1981), pp. 349–355.

[12] N. Karmarkar, R. Karp, *An efficient approximation scheme for the one-dimensional bin-packing problem*, Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, (1982), pp. 312–320.

[13] S. M. Ross, *Introduction to Probability Models*, Academic Press, (2000).

[14] T. P. Runarsson, P. Jensson, M. T. Jonsson, *Dynamic dual bin packing using fuzzy objectives*, Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC '96), (1996), pp. 219–222.

Data Reduction, Exact, and Heuristic Algorithms for Clique Cover

Jens Gramm* Jiong Guo† Falk Hüffner† Rolf Niedermeier†

Abstract

To cover the edges of a graph with a minimum number of cliques is an NP-complete problem with many applications. The state-of-the-art solving algorithm is a polynomial-time heuristic from the 1970's. We present an improvement of this heuristic. Our main contribution, however, is the development of efficient and effective polynomial-time data reduction rules that, combined with a search tree algorithm, allow for exact problem solutions in competitive time. This is confirmed by experiments with real-world and synthetic data. Moreover, we prove the fixed-parameter tractability of covering edges by cliques.

1 Introduction

Data reduction techniques for exactly solving NP-hard combinatorial optimization problems have proven useful in many studies [1, 3, 10, 16]. The point is that by polynomial-time executable reduction rules many input instances of hard combinatorial problems can be significantly shrunk and/or simplified, without sacrificing the possibility of finding an optimal solution to the given problem. For such reduced instances then often exhaustive search algorithms can be applied to efficiently find optimal solutions in reasonable time. Hence data reduction techniques are considered as a "must" when trying to cope with computational intractability. Studying the NP-complete problem to cover the edges of a graph with a minimum number of cliques ((EDGE) CLIQUE COVER)[1], we add a new example to the success story of data reduction, presenting both empirical as well as theoretical findings.

Our study problem CLIQUE COVER, also known as KEYWORD CONFLICT problem [12] or COVERING BY CLIQUES (GT17) or INTERSECTION GRAPH BASIS (GT59) [9], has applications in diverse fields such as compiler optimization [19], computational geometry [2], and applied statistics [11, 18]. Thus, it is not surprising that there has been substantial work on (polynomial-time) heuristic algorithms for CLIQUE COVER [12, 14, 19, 18, 11]. In this paper, we extend and complement this work, particularly introducing new data reduction techniques.

Formally, as a (parameterized) decision problem, CLIQUE COVER is defined as follows:

> CLIQUE COVER
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Is there a set of at most k cliques in G such that each edge in E has both its endpoints in at least one of the selected cliques?

CLIQUE COVER is hard to approximate in polynomial time and nothing better than only a polynomial approximation factor is known [5].

We examine CLIQUE COVER in the context of parameterized complexity [7, 17]. An instance of a parameterized problem consists of a problem instance I and a parameter k. A parameterized problem is *fixed-parameter tractable* if it can be solved in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function solely depending on the parameter k, not on the input size $|I|$.

Our contributions are as follows. We start with a thorough mathematical analysis of the heuristic of Kellerman [12] and the postprocessing of Kou et al. [14], which is state of the art [19, 11]. For an n-vertex and m-edge graph, we improve the runtime from $O(nm^2)$ to $O(nm)$. Afterwards, as our main algorithmic contribution, we introduce and analyze data reduction techniques for CLIQUE COVER. As a side effect, we provide a so-called problem kernel for CLIQUE COVER, for the first time showing—somewhat surprisingly—that the problem is fixed-parameter tractable with respect to the parameter k. We continue with describing an exact algorithm based on a search tree. For our experimental investigations, we combined our data reduc-

*Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany, gramm@informatik.uni-tuebingen.de. Supported by DFG project OPAL, NI 369/2.

†Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany, {guo|hueffner|niedermr}@minet.uni-jena.de. Jiong Guo and Falk Hüffner supported by DFG Emmy Noether research group PIAF, NI 369/4.

[1] We remark that covering *vertices* by cliques (VERTEX CLIQUE COVER or CLIQUE PARTITION) is of less interest to be studied on its own because it is equivalent to the well-investigated GRAPH COLORING problem: A graph has a vertex clique cover of size k iff its complement graph can be colored with k colors such that adjacent vertices have different colors.

tion rules with the search tree, clearly outperforming heuristic approaches in several ways. For instance, we can solve real-world instances from a statistical application [18]—so far solved heuristically [18, 11]—optimally without time loss. This indicates that for a significant fraction of real-world instances our exact approach is clearly to be preferred to a heuristic approach which is without guaranteed solution quality. We also experimented with random graphs of different densities, showing that our exact approach works extremely well for sparse graphs. In addition, our empirical results reveal that for dense graphs a data reduction rule that was designed for showing the problem kernel does very well. In particular, this gives strong empirical support for further theoretical studies in the direction of improved fixed-parameter tractability results for CLIQUE COVER, nicely demonstrating a fruitful interchange between applied and theoretical algorithmic research.

Not all details and proofs are given in this extended abstract.

2 Improved Heuristic

Kellerman [12] proposed a polynomial-time heuristic for CLIQUE COVER. This heuristic was improved by Kou, Stockmeyer, and Wong [14] by adding a postprocessing step; this version has been successfully applied to instruction scheduling problems [19] and in the analysis of statistical data [11]. Clearly, both versions of the heuristic run in polynomial time but in both cases a more precise analysis of their runtime was not given. In this section, for an n-vertex and m-edge graph, we analyze the runtime of both heuristics as $O(nm^2)$. We show how to slightly modify Kellerman's heuristic such that we can improve the runtime of both heuristics to $O(nm)$ by a careful use of additional data structures.

The CLIQUE COVER heuristic by Kellerman [12], further-on referred to as CC-Heuristic 1, is described in pseudo-code in the left column of Fig. 1. To simplify notation, we use $V = \{1, \ldots, n\}$ as vertex set. The algorithm starts with an empty clique cover and successively, for $i = 1, \ldots, n$, updates the clique cover to account for edges $\{i, j\}$ with $j < i$. In case there are no edges between the currently processed vertex i and the set W of its already processed neighbors, a new clique is created, containing only i. Otherwise, we try to add i to existing cliques where possible. After this, there may still remain uncovered edges between i and W. To cover those edges, we create a new clique containing i and its neighbors from one of the existing cliques such that the number of edges covered by this new clique is maximized. We repeat this process until all edges between i and vertices from W are covered.

To improve both the analysis and in some cases the result of the heuristic, we assume that the algorithm is aborted as soon as m cliques are generated. In that case we can simply take the solution that covers each edge separately by a two-vertex clique.

Input: Clique C_l, vertex i.
Globals: S, I

```
1  for j ∈ N>(j'), with some arbitrary j' ∈ C_l:
2      if l ∈ S[j] ∧ i ∉ N<(j):
3          S[j] ← S[j] \ {l}
4  for j ∈ N>(i):
5      update position of l in sorted I[j]
6  C_l ← C_l ∪ {i}
```

Figure 2: The *add-to-clique* subroutine.

It is relatively straightforward to observe that CC-Heuristic 1 runs in $O(nm^2)$ time.

To improve the runtime, the idea is to identify the "hot spots" and to use caching data structures that will give the computation of these hot spots basically for free, and spread the work of keeping this structure up-to-date throughout the rest of the program.

We maintain the following two tables for vertices i, $1 \leq i \leq n$, where C_l, $1 \leq l \leq m$, are the cliques generated by the algorithm:

$$S[i] := \{l \mid C_l \subseteq N_<(i)\},$$
$$I[i] := \{l \mid C_l \cap N_<(i) \neq \emptyset\}, \text{ sorted descending according to } |C_l \cap N_<(i)|,$$

where the set $N_<(i)$ for a vertex i is defined as

$$N_<(i) := \{j \mid j < i \text{ and } \{i, j\} \in E \text{ is uncovered}\},$$

with $\{i, j\} \in E$ called *uncovered* if $\forall 1 \leq l \leq m : \{i, j\} \not\subseteq C_l$. The set $N_>(i)$ is defined analogously. The table $S[i]$ is used to keep track of the existing cliques to which a vertex i may be added. Using $S[i]$ will avoid to inspect every existing clique individually in order to test whether vertex i can be added to the clique (see lines 10 and 11 for CC-Heuristic 1 in Fig. 1). The table $I[i]$ is used to keep track of the existing cliques with which a vertex i has an "uncovered overlap", i.e., with which i shares yet uncovered edges. The cliques are kept sorted by the size of this uncovered overlap. Using table $I[i]$, we will avoid the costly computation of a clique with maximum uncovered overlap in line 18. Thus, tables S and I help to replace costly operations during the heuristic by constant-time look-ups in these tables. This comes at the price of having to keep these tables up-to-date throughout the heuristic.

We now first explain how the newly introduced tables are updated throughout the heuristic and, then,

```
Input:  Graph G = ({1,2,...,n}, E) without isolated vertices.
Output: A clique cover for G.
1   k ← 0   ▷ number of cliques created so far
2   for i = 1,2,...,n:
        ▷ loop invariant: C_1,...,C_k cover all edges incident to vertices v,w ≤ i
3       W ← {j | j < i and {i,j} ∈ E}
4       if W = ∅:
5           k ← k + 1
6           C_k ← {i}                          |  add-to-clique(k,i)
7       else:
8           ▷ try to add v_i to each of the existing cliques
9           U ← ∅   ▷ set of neighbors j of i where {i,j} is already covered
10          for l = 1,...,k:                   |  S' ← S[i]
11              if C_l ⊆ W:                    |  for l ∈ S':
                                               |      if C_l ⊈ U:
12                  C_l ← C_l ∪ {i}            |          add-to-clique(l,i)
13                  U ← U ∪ C_l                |          U ← U ∪ C_l
14                  if U = W: break            |          if U = W: break
15          W ← W \ U
16          while W ≠ ∅:
17              ▷ for the remaining edges, try to cover as many as possible at a time
18              l ← min{l | 1 ≤ l ≤ k,         |  l ← min{I[i]}
                          |C_l ∩ W| is maximal}
19              k ← k + 1                      |  k ← k + 1
20              C_k ← (C_l ∩ W) ∪ {i}           |  for q ∈ (C_l ∩ W) ∪ {i}:
                                               |      add-to-clique(k,q)
21              W ← W \ C_l
22  return {C_1, C_2, ..., C_k}
```

Figure 1: Comparison of CC-Heuristic 1 by Kellerman [12] and the improved CC-Heuristic 2 in pseudo-code. Code in the right column indicates that in CC-Heuristic 2 the code replaces the corresponding lines of CC-Heuristic 1 shown in the left column.

we show how they are used to modify CC-Heuristic 1. Initially, the entries $S[i]$ and $I[i]$ are empty for all $1 \leq i \leq n$. To handle all modifications to our partial clique cover and to update tables S and I, we introduce the function *add-to-clique*, presented in pseudo-code in Fig. 2. The function adds one vertex i to a clique C_l and updates these tables as follows.

- Table S: The grown C_l might not be subset of $N_<(j)$ anymore for some j with $l \in S[j]$. This is accounted for in lines 1–3. To this end, we need to find all vertices j such that clique C_l *is* subset of $N_<(j)$ *before* adding i to C_l but *is not* subset of $N_<(j)$ *after* adding i to C_l. All these j are certainly found when inspecting all neighbors of one arbitrarily selected element j' of the old C_l. If, for some j, $l \in S[j]$ but $i \notin N_<(j)$, then we remove l from $S[j]$.
- Table I: For each j in $N_>(i)$, we percolate l to its right place in $I[j]$ since $C_l \cap N_<(j)$ has grown by one. This is done in lines 4–5.

Clearly, after these updates, the addition of i to C_l has been correctly accounted for.

Fig. 1 shows how function *add-to-clique* is used to modify CC-Heuristic 1 in order to obtain our new heuristic to which we refer as CC-Heuristic 2. More precisely, we replace every addition of a vertex to a clique by a call to the new function *add-to-clique*; this explains the changes of lines 6, 12, and 20 and makes sure that in these modifications the tables I and S are updated.

Table S is then used to speed up lines 10 to 14 of the old heuristic where the currently processed vertex i is added to existing cliques where possible. Here, instead of testing for each existing clique individually whether this is possible (as done in CC-Heuristic 1), CC-Heuristic 2 directly accesses these cliques in table entry $S[i]$. Moreover, we also change the heuristic by adding vertex i to an existing clique C_l only if there are edges between i and C_l which are yet uncovered. In contrast, CC-Heuristic 1 adds vertex i to every possible existing clique C_l (unless all edges connecting i to pre-

viously processed vertices are already covered). Thus, in some cases CC-Heuristic 1 may add vertex i to an existing clique while not covering previously uncovered edges, and, in other cases, vertex i is not added to an existing clique although this would be possible. Therefore, our modification makes the algorithm more consistent and additionally this change will be essential in the runtime proof.

Table I is used to speed up lines 18 to 20 of the old heuristic where new cliques are generated for the still uncovered edges connecting the currently processed vertex i to its already processed neighbors in W. More precisely, we generate a clique containing i and $C_l \cap W$ where C_l is chosen such that it maximizes the overlap with W. While the choice of l was time-consuming in CC-Heuristic 1, it is now done by a constant-time look-up in table entry $I[i]$. Then, the new clique C_k is built by adding each element individually, using *add-to-clique*. This concludes the changes of CC-Heuristic 2 in comparison with CC-Heuristic 1.

To infer the runtime we observe the following essential invariant which applies to the solutions generated by CC-Heuristic 2:

LEMMA 2.1. *In a solution generated by CC-Heuristic 2, the sum of solution clique sizes is at most* $2m$.

It is not clear how to prove the invariant stated in Lemma 2.1 for CC-Heuristic 1. The reason lies in line 11 of the heuristic where CC-Heuristic 1 may add new vertices to a clique without covering previously uncovered edges. This is prevented by the modification introduced in CC-Heuristic 2 and we can infer the following runtime:

THEOREM 2.1. *CC-Heuristic 2 runs in* $O(nm)$ *time.*

Proof. We first analyze the runtime of *add-to-clique*. Clearly, each of the for loops iterates at most n times. The set operations in lines 2 and 3 can be implemented to run in constant time by using bitmaps. The update in line 5 can also be done in constant time by using n buckets for each $I[i]$ and an additional map that allows us to find the bucket where l resides. In summary, one *add-to-clique* call takes $O(n)$ time.

To analyze the runtime of the modified algorithm, we note that the total runtime is dominated by *add-to-clique* calls. We analyze the number and total runtime of *add-to-clique* calls amortized over the whole algorithm: With Lemma 2.1 the sum of clique sizes is at most $2m$ and therefore *add-to-clique* is called at most $2m$ times. Since lines 13 and 14 take at most $O(n)$ time and are called only on occasion of an *add-to-clique* call, their runtime can be also subsumed here. This shows a total runtime of $O(nm)$ for all *add-to-clique* calls (including lines 13 and 14).

Leaving aside the *add-to-clique* calls and lines 13/14, all other operations inside the main loop—iterating over all n vertices—can be done in $O(m)$ time. This is in particular true for line 11: Using bitmaps in combination with a doubly-linked structure, testing C_l for containment in a set U can be done in $|C_l|$ steps. With Lemma 2.1 we infer that testing *all* existing cliques for containment in a set U can be done in $O(m)$ time.

Finally, we turn our attention to the postprocessing proposed by Kou et al. [14] as an addition for CC-Heuristic 1. They proposed to test every clique found by CC-Heuristic 1 for redundancy: For every clique C in the solution of CC-Heuristic 1, it is tested whether C is "subsumed" by the remaining cliques of the solution, i.e., whether each of C's edges is also covered by another clique of the solution. If C is subsumed, then C is deleted from the solution. Kou et al. left it open to give a precise estimation of the time complexity of this postprocessing.

The following result is straightforward to obtain by using appropriate caching data structures.

PROPOSITION 2.1. *The postprocessing proposed by Kou et al. [14] on instances returned by CC-Heuristic 2 can be done in* $O(nm)$ *time.*

3 Data Reduction

A (*data*) *reduction rule* replaces, in polynomial time, a given CLIQUE COVER instance (G, k) consisting of a graph G and a nonnegative integer k by a "simpler" instance (G', k') such that (G, k) has a solution iff (G', k') has a solution. An instance to which none of a given set of reduction rules applies is called *reduced* with respect to these rules. A parameterized problem such as CLIQUE COVER (the parameter is k) is said to have a *problem kernel* if, after the application of the reduction rules, the reduced instance has size $f(k)$ for a function f depending only on k. It is a well-known result from parameterized complexity theory that the existence of a problem kernel implies fixed-parameter tractability for a parameterized problem [7, 17].

We formulate reduction rules for a generalized version of CLIQUE COVER in which already some edges may be marked as "covered." Then, the question is to find a clique cover of size k that covers all non-covered edges. Clearly, CLIQUE COVER is the special case of this annotated version where no edge is marked as covered.

We start by describing an initialization routine that sets up auxiliary data structures *once* at the beginning of the algorithm such that the *many* applications of the

subsequent Rule 2 become cheaper in terms of runtime. Moreover, the data structures initialized here are also used by the exact algorithm proposed in Sect. 4. From the reduction rules below, only Rule 1 updates these auxiliary data structures.

Initialization We inspect every edge $\{u, v\}$ of the original graph. We use two auxiliary variables: We compute a set $N_{\{u,v\}}$ of its common neighbors and we determine whether the vertices in $N_{\{u,v\}}$ induce a clique. More precisely, we compute a positive integer $c_{\{u,v\}}$ which stores the number of edges interconnecting the vertices of $N_{\{u,v\}}$.

The following is easy to see.

LEMMA 3.1. *The proposed initialization can be done in $O(m^2)$ time.*

We start the presentation of reduction rules with a trivial rule removing isolated elements.

RULE 1. *Remove isolated vertices and vertices that are only adjacent to covered edges.*

LEMMA 3.2. *Rule 1 is correct. Every application of Rule 1 including the update of auxiliary variables can be executed in $O(nm)$ time.*

The next reduction rule is concerned with maximal cliques. Note that we can safely assume that an optimal solution consists of maximal cliques only since a non-maximal clique in a solution can always be replaced by a maximal clique it is contained in. The following rule identifies maximal cliques which have to be part of *every* optimal solution.

RULE 2. *If an edge $\{u, v\}$ is contained only in exactly one maximal clique C, i.e., if the common neighbors of u and v induce a clique, then add C to the solution, mark its edges as covered, and decrease k by one.*

LEMMA 3.3. *Rule 2 is correct. Every application of Rule 2 can be executed in $O(m)$ time.*

Rules 1 and 2 imply that all degree-1 and degree-2 vertices are removed from the instance. Further, they imply that an isolated clique is deleted: Its vertices belong to exactly one maximal clique; the clique, if it contains more than one vertex, is added to the solution by Rule 2 and its vertices are "cleaned up" by Rule 1.

In the following we present two interrelated reduction rules Rules 3' and 3. Rule 3' is subsumed by Rule 3. Nevertheless we choose to present both rules separately since Rule 3' is easier to understand and more efficient to implement. Moreover, as will be shown in Theorem 3.1, already Rule 3' is sufficient to show a problem kernel for CLIQUE COVER.

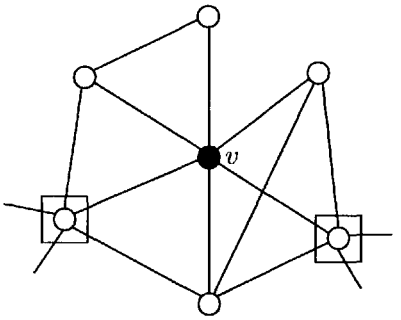


Figure 3: An illustration of the partition of the neighborhood of a vertex v. The two vertices with rectangles are exits, the others are prisoners.

RULE 3'. *If there is an edge $\{u, v\}$ whose endpoints have exactly the same closed neighborhood, i.e., for which $N[u] = N[v]$, then mark all edges incident to u as covered. To reconstruct a solution for the unreduced instance, add u to every clique containing v.*

Comparing $N[u]$ and $N[v]$ for each edge $\{u, v\}$, we can in $O(nm)$ time search an edge for which Rule 3' is applicable and invoke the rule.

For formulating a generalization of Rule 3' we introduce additional terminology. For a vertex v, we partition the set of vertices that are connected by an uncovered edge to v into *prisoners* p with $N(p) \subseteq N(v)$ and *exits* x with $N(x) \setminus N(v) \neq \emptyset$.[2] We say that the prisoners *dominate* the exits if for every exit x there is a prisoner connected to x. An illustration of the concept of prisoners and exits is given in Fig. 3.

RULE 3. *If there is a vertex v which has at least one prisoner and whose prisoners dominate its exits, then mark all edges incident to v as covered. To reconstruct a solution for the unreduced instance, add v to every clique containing a prisoner of v.*

Observe that a vertex v is always a prisoner of a vertex u with $u \neq v$ and $N[u] = N[v]$ (and vice versa). Thus, Rule 3' is subsumed by Rule 3.

LEMMA 3.4. *Rule 3 is correct. Every application of Rule 3 can be executed in $O(n^3)$ time.*

Proof. For the correctness note that, by definition, every neighbor of v's prisoners is also a neighbor of v itself. If a prisoner of v participates in a clique C, then $C \cup \{v\}$ is also a clique in the graph. Therefore,

[2] We remark that the concept of prisoners and exits (and, in addition, "gates") was introduced for data reduction rules designed for the DOMINATING SET problem [4]. The strength of these rules has been proven theoretically [4] as well as empirically [3].

it is correct to add v to every clique containing a prisoner in the reduced graph. Next, we show that all edges adjacent to v are covered by the cliques resulting by adding v to the cliques containing v's prisoners. W.l.o.g. we can assume that prisoners are not "isolated," i.e., they are connected to other prisoners or exits since, otherwise, Rules 1 and 2 would delete the isolated prisoner. Now, we consider separately the edges connecting v to prisoners and edges connecting v to exits. Regarding an edge $\{v, w\}$ to a non-isolated prisoner w, vertex w has to be part of a clique C of the solution for the instance after application of the rule. Therefore, the edge $\{v, w\}$ is covered by $C \cup \{v\}$ in the solution for the unreduced instance. Regarding an edge $\{v, x\}$ to an exit x, the exit x is dominated by a prisoner w and therefore x has to be part of a clique C with w. Therefore, the edge $\{v, x\}$ is covered by $C \cup \{v\}$ in the solution for the unreduced instance.

For executing the rule, we inspect every vertex v to test whether the rule is applicable. To this end, we inspect every neighbor u of v. In $O(n)$ time, we determine whether u is an exit or a prisoner. Having identified all prisoners, we can for every exit u determine in $O(n)$ time whether u is dominated by a prisoner.

LEMMA 3.5. *Using Rules 1 to 3, in $O(n^4)$ time one can generate a reduced instance where none of these rules applies any further.*

From a theoretical viewpoint, the main result of this section is a problem kernel with respect to parameter k for CLIQUE COVER:

THEOREM 3.1. *A* CLIQUE COVER *instance reduced with respect to Rules 1 and 3' contains at most 2^k vertices or, otherwise, has no solution.*

Proof. Consider a graph $G = (V, E)$ that is reduced with respect to Rules 1 and 3' and has a clique cover $C_1, \ldots, C_k$ of size k. We assign to each vertex $v \in V$ a binary vector b_v of length k where bit i, $1 \leq i \leq k$, is set iff v is contained in clique C_i. If we assume that G has more than 2^k vertices, then there must be $u \neq v \in V$ with $b_u = b_v$. Since Rule 1 does not apply, every vertex is contained in at least one clique, and since $b_u = b_v$, u and v are contained in the same cliques. Therefore, u and v are connected. As they also share the same neighborhood, Rule 3' applies, in contradiction to our assumption that G is reduced with respect to Rule 3'. Consequently, G cannot have more than 2^k vertices.

COROLLARY 3.1. CLIQUE COVER *is fixed-parameter tractable with respect to parameter k.*

Input: Graph $G = (\{1, 2, \ldots, n\}, E)$.
Output: A minimum clique cover for G.

```
1  k ← 0; X ← nil
2  while X = nil:
3      X ← branch(G, k, ∅)
4      k ← k + 1
5  return X
6  function branch(G, k, X):
7  if X covers G: return X
8  reduce (G, k)
9  if k < 0: return nil
10 choose {i, j} such that (|N_{i,j}| choose 2) − c_{i,j} is minimal
11 for each maximal clique C in N[i] ∩ N[j]:
12     X' ← branch(G, k − 1, X ∪ {C})
13     if X' ≠ nil: return X'
14 return nil
```

Figure 4: Exact algorithm for CLIQUE COVER.

The result of Corollary 3.1 might be surprising when noting that many graph problems that involve cliques turn out to be hard in the parameterized sense. For example, the NP-complete CLIQUE problem is known to be W[1]-complete with respect to the clique size [7, 17]. Another example even more closely related to CLIQUE COVER is given by the NP-complete CLIQUE PARTITION problem, which is also hard in the parameterized sense. Herein, we ask, given a graph, for a set of k cliques covering all *vertices* of the input graph (in contrast to covering all *edges* as in CLIQUE COVER). CLIQUE PARTITION is NP-hard already for $k = 3$ [9]. It follows that there is no hope for obtaining fixed-parameter tractability for CLIQUE PARTITION with respect to parameter k, unless P = NP.

4 Exact Search Tree Algorithm

Search trees are a popular means of exactly solving hard problems. The basic method is to identify for a given instance a small set of simplified instances such that the given instance has a solution if at least one of the simplified instances has one. The corresponding algorithm branches recursively into each of these instances until a stop criterion is met.

The search tree algorithm presented here for CLIQUE COVER works as follows. We choose an uncovered edge, enumerate all maximal cliques this edge is part of, and then branch according to which of these cliques we add to the clique cover. The recursion stops as soon as a solution is found or k cliques are generated without finding a solution. The algorithm is presented in pseudo-code in Fig. 4.

Regarding the choice of the edge to branch on, we would, ideally, like to branch on the edge that is contained in the least number of maximal cliques. However, this calculation would be costly. Therefore, we make use of the infrastructure set up for an efficient incremental application of Rule 2. The initialization described in Sect. 3 provides a set $N_{\{i,j\}}$ containing the common neighborhood of edge $\{i,j\}$ and a counter $c_{\{i,j\}}$ containing the number of edges in the common neighborhood of its endpoints. Therefore, $\binom{|N_{\{i,j\}}|}{2} - c_{\{i,j\}}$ is the number of edges missing in the common neighborhood of edge $\{i,j\}$ as compared to a clique (the *score*). For branching, we choose the edge with the lowest score. If the score is 0, then the edge is contained in only one maximal clique (and thus will be reduced). If the score is 1, the edge is contained in exactly two maximal cliques. Generalizing this, it is plausible to assume that an edge is contained in few maximal cliques if its score is low.

Having chosen the edge to branch on, we determine the set of maximal cliques the edge is contained in using a variant of the classical Bron–Kerbosch algorithm [6] by Koch [13].

We use the branching routine within an iterative deepening framework, that is, we impose a maximum search depth k and increase this limit by one when no solution is found.

Combining the data reduction rules described in Sect. 3—which yield a problem kernel for CLIQUE COVER—with the search tree algorithm described here, we obtain a competitive fixed-parameter algorithm for CLIQUE COVER that can solve problem instances of considerable size (a few hundred vertices) efficiently (see Sect. 5).

5 Experimental Results

In this extended abstract we focus on the newly developed exact algorithm with data reduction rules—eperimental investigations of CC-Heuristic 1 are kept to a minimum and of CC-Heuristic 2 are completely omitted. This is to become part of the full version of the paper.

We implemented the search tree algorithm from Sect. 4 and the data reduction rules from Sect. 3. The program is written in the Objective Caml programming language [15] and consists of about 1200 lines of code. The source code is free software and available from the authors on request. Graphs are implemented using a purely functional representation based on Patricia trees. This allows to (conceptually) modify the graph in the course of the algorithm without having to worry about how to restore it when returning from the recursion. Moreover, it allows for quick intersection operations on neighbor sets, as required for some reduction rules. The cache data structure c described in Sect. 3 is implemented using a priority search queue.

			Clique cover size	
	n	m	Heuristic	Optimal
A	13	55	4	4
B	17	86	6	5
C	124	4847	50	49
D	121	4706	48	48
E	97	3559	34	31

Table 1: Clique cover sizes for five real-world CLIQUE COVER instances, where "Heuristic" is CC-Heuristic 1 with the postprocessing by Kou et al. [14].

We tested our implementation on various inputs on an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory, running under the Debian GNU/Linux 3.1 operating system.

Real Data. We first tested the implementation on five "real-world" instances from an application in graphical statistics [18] (see Table 1). Currently, heuristics like that of Kou et al. [14] are used to solve the problem in practice [18, 11]. With our implementation of CC-Heuristic 1, the runtime is negligible for these instances (< 0.1 s). Our implementation based on the search tree with data reduction could solve all instances to optimality within less than two seconds. We observe that the heuristic produces reasonably good results for these cases; previously nothing was known about its solution quality. In summary, the application of our algorithm in this area seems quite attractive, since we can provide provably optimal results within acceptable runtime bounds.

Random Graphs. Next, we tested the implementation of the exact algorithm on random graphs, that is, graphs where every possible edge is present with a fixed probability. It is known that with high probability a random graph has a large clique cover of size $O(n^2/\log^2 n)$ [8]. Therefore, relying on branching and a not too large search tree is unlikely to succeed, and reduction rules are crucial. The results are presented in Fig. 5. In the following, the "size" of an instance means the number of vertices. We exhibit three trials: Sparse graphs with $m \approx n \log n$, graphs with edge probability 0.1, and graphs with edge probability 0.15. For the denser graphs outliers occur: for example for graphs of size 51 and edge probability 0.15, all instances could be solved within a second but one, which took 25 minutes. In contrast, sparse graphs can be solved uniformly very quickly, and the growth even seems to be subexponen-

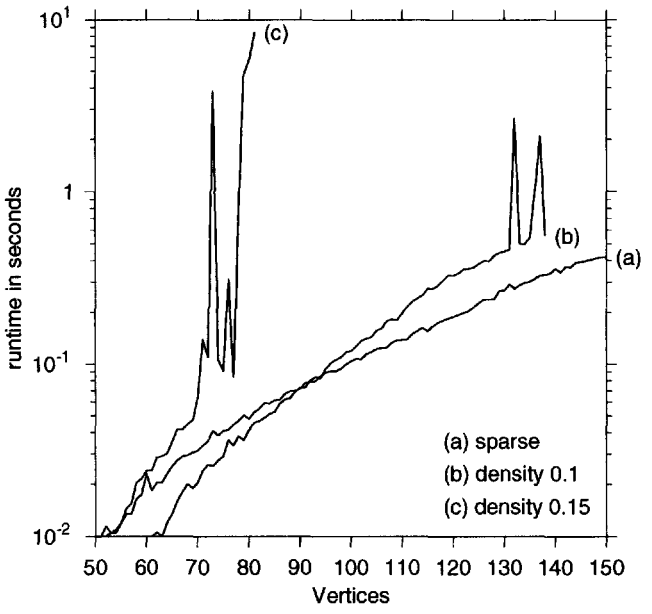


Figure 5: Runtime for random graphs. Each point is the average over 20 runs.

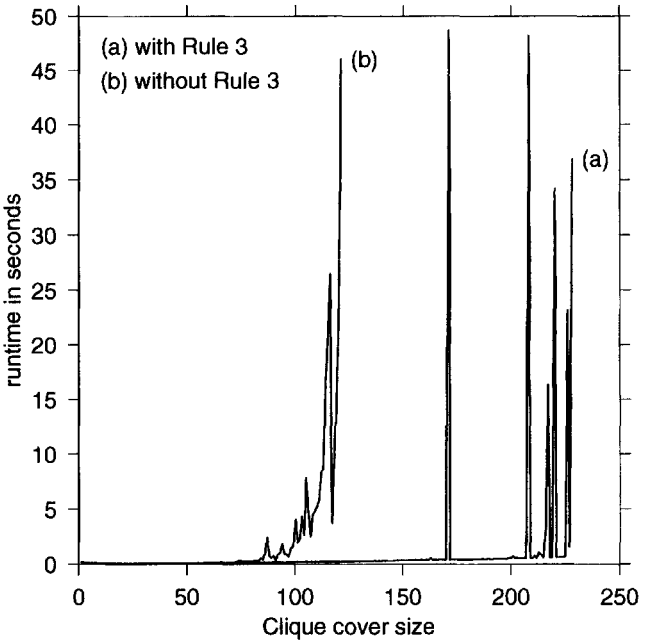


Figure 6: Average runtime for graphs with $n \approx 200$ and $m \approx 2000$ constructed by randomly placed cliques.

tial: Instances of size 1 000 can still be solved within 100 seconds and instances of size 2 000 within 11 minutes, with a standard deviation for the runtime of $< 2\,\%$. Our approach is very promising for sparse instances up to moderate size, while for denser instances probably a fallback to heuristic algorithms is required to compensate for the outliers.

The presence of extreme outliers for some combinations of parameters makes it difficult to get a clear picture based only on combining statistics such as averages. Therefore, we show measurements for several concrete instances in Table 2. For edge probability 0.1 and 0.15, respectively, we select an instance that takes very long, and additionally present two arbitrary instances with similar parameters. For sparse graphs, no such outliers occur, so we show three arbitrary instances of similar size.

Synthetic Data. Real instances are not completely random; in particular, in most sensible applications the clique cover is expected to be much smaller than that of a random graph. The fixed-parameter tractability of our algorithm also promises a better runtime for instances where the clique cover is small. To examine this, we generated random graph instances with approximately 200 vertices and 2000 edges by successively completing random sets of random size to form cliques until at least 2000 edges are present, but no more than 2020. By choosing the maximum size of the placed cliques, the number of placed cliques is (roughly) controlled. Figure 6 shows the resulting runtimes. In fact, these comparably large and dense instances can be solved very quickly when the size of the clique cover is small. Below a clique cover size of about 150, performance is also very smooth; no outliers occur. In contrast, the performance becomes erratic for more than about 170 cliques, with frequent occurrences of instances taking very long to solve. In summary, this makes our exact algorithm also attractive for the numerous applications where we can expect a small clique cover as solution.

Effectiveness of Rule 3. The prisoner-exit-rule (Rule 3) is comparably complicated and expensive and has been developed in context with searching for a problem kernel. Does it really gain any benefit in practice? To examine this question, we repeated the previous experiment with Rule 3 disabled (see Figure 6). While initially similar, around a cover size of 80 the performance drops sharply, and outliers taking very long time to solve occur. This means that Rule 3 nearly doubled the range of instances that can be solved smoothly, and is clearly worthwhile.

6 Outlook

As seen in Table 2, there are some outliers with exceedingly high runtimes when compared to "similar" instances. Without the application of data reduction Rule 3, there were even more such outliers. This clearly indicates that Rule 3, which also leads to a size-2^k prob-

	n	m	$\|C\|$	runtime	search tree	Rule 1	Rule 2	Rule 3
sparse	602	3 837	3 230	21.77	6 463	6 434	5 214 833	1 192
	602	3 786	3 239	21.55	6 479	3 472	5 243 941	0
	603	3 910	3 340	23.37	66 81	6 365	5 576 130	0
$p = 0.1$	152	1 202	628	4 860.24	76 856 966	103 117 567	126 934 019	166 594 479
	152	1 130	627	0.83	1 578	1 126	196 093	4 972
	151	1 207	644	1.01	1 603	1 715	206 732	6 260
$p = 0.15$	80	531	243	24 002.49	1 063 679 952	1 327 673 517	397 529 584	225 500 975
	80	492	244	0.11	1 248	898	29 450	1 753
	80	501	242	0.73	33 769	47 098	38 331	9 488

Table 2: Statistics for selected CLIQUE COVER instances. Here, p is the edge probability, runtime is in seconds, $|C|$ is the size of the clique cover, "search tree" is the number of nodes in the search tree, and "Rule r" is the number of applications of Rule r.

lem kernel, can cope with some of the outliers but not all. Hence it is an intriguing open question whether there are further data reduction rules that can cope with the remaining outliers. In parallel, this might also lead to a better upper bound on the problem kernel size and improved fixed-parameter tractability for CLIQUE COVER.

Acknowledgements. We thank Ramona Schmid (Universität Tübingen) for help with the implementation and Hans-Peter Piepho (Universität Hohenheim) for providing the test data.

References

[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proc. 6th ALENEX*, pages 62–69. SIAM, 2004.

[2] P. K. Agarwal, N. Alon, B. Aronov, and S. Suri. Can visibility graphs be represented compactly? *Discrete and Computational Geometry*, 12:347–365, 1994.

[3] J. Alber, N. Betzler, and R. Niedermeier. Experiments on data reduction for optimal domination in networks. In *Proc. 1st INOC*, pages 1–6, 2003. Long version to appear in *Annals of Operations Research*.

[4] J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for Dominating Set. *Journal of the ACM*, 51:363–384, 2004.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

[6] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[7] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[8] A. M. Frieze and B. A. Reed. Covering the edges of a random graph by cliques. *Combinatorica*, 15(4):489–497, 1995.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[10] J. F. Gimpel. A reduction technique for prime implicant tables. *IEEE Transactions on Electronic Computers*, EC-14:535–541, 1965.

[11] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H.-P. Piepho, and R. Schmid. Algorithms for compact letters displays—comparison and evaluation. Manuscript, FSU Jena, Dec. 2005.

[12] E. Kellerman. Determination of keyword conflict. *IBM Technical Disclosure Bulletin*, 16(2):544–546, 1973.

[13] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1–2):1–30, 2001.

[14] L. T. Kou, L. J. Stockmeyer, and C.-K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM*, 21(2):135–139, 1978.

[15] X. Leroy, J. Vouillon, D. Doligez, et al. The Objective Caml system. Available on the web, 1996. `http://caml.inria.fr/ocaml/`.

[16] S. Mecke and D. Wagner. Solving geometric covering problems by data reduction. In *Proc. 12th ESA*, volume 3221 of *LNCS*, pages 760–771. Springer, 2004.

[17] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[18] H.-P. Piepho. An algorithm for a letter-based representation of all-pairwise comparisons. *Journal of Computational and Graphical Statistics*, 13(2):456–466, 2004.

[19] S. Rajagopalan, M. Vachharajani, and S. Malik. Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints. In *Proc. CASES*, pages 157–164. ACM Press, 2000.

Fast Reconfiguration of Data Placement in Parallel Disks *

Srinivas Kashyap and Samir Khuller †
Department of Computer Science
University of Maryland
College Park, MD 20742
Email: {raaghav, samir}@cs.umd.edu

Yung-Chun (Justin) Wan
Google

Leana Golubchik
Department of Computer Science and IMSC
University of Southern California
Los Angeles, CA 90089
Email: leana@cs.usc.edu

1 Introduction

The "How much information?" study produced by the school of information management and systems at the University of California at Berkeley [10], estimates that about 5 exabytes of new information was produced in 2002. It estimates that the amount of stored information doubled in the period between 1999 and 2002. It is believed that more data will be created in the next five years than in the history of the world. Clearly we live in an era of data explosion. This data explosion necessitates the use of large storage systems. Storage Area Networks (or SANs) are the leading [13] infrastructure for enterprise storage.

A SAN essentially allows multiple processors to access several storage devices. They typically access the storage medium as though it were one large shared repository. One crucial function of such a storage system is that of deciding the placement of data within the system. This data placement is dependent on the demand pattern for the data. For instance, if a particular data item is very popular the storage system might want to host it on a disk with high bandwidth or make multiple copies of the item. The storage system needs to be capable of handling flash crowds [8]. During events triggered by such flash crowds, the demand distribution becomes highly skewed and different from the normal demand distribution.

It is known that the problem of computing an optimal data placement[1] for a given demand pattern is NP-Hard [5]. However, polynomial time approximation schemes as well as efficient combinatorial algorithms that compute almost optimal solutions are known for this problem [12, 11, 5]. So we can assume that a near-optimal placement can be computed once a demand pattern is specified.

As the demand pattern changes over time and the popularity of items changes, the storage system will have to modify its internal placement accordingly. Such a modification in placement will typically involve movement of data items from one set of disks to another or requires changing the number of copies of a data item in the system. For such a modification to be effective it should be computed and applied quickly. In this work we are concerned with the problem of finding such a modification i.e., modifying the existing placement to efficiently deal with a new demand pattern for the data. This problem is referred to as the data migration problem and was considered in [9, 6]. The authors used a data placement algorithm to compute a new "target" layout. The goal was to "convert" the existing layout to the target layout as quickly as possible. The communication model that was assumed was a half-duplex model where a matching on the disks can be fixed, and for each matched pair one can transfer a single object in a round. The goal was to minimize the number of rounds taken. The paper developed

*This research was supported by NSF grant CCF-0430650. The research was also funded in part by the NSF grant EIA-0091474, the Okawa Research Award, and the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

†Department of Computer Science and UMIACS

[1] An optimal placement will allow a maximum number of users to access information of their interest.

constant factor approximation algorithms for this NP-hard problem [9]. In practice these algorithms find solutions that are reasonably close to optimal. However, even when there is *no* drastic change in the demand distribution it can still take many rounds of migration to achieve the new target layout. This happens since the scheme completely disregards the existing placement in trying to compute the target placement.

In this paper we consider a new approach to dealing with the problem of changes in the demand pattern. We ask the following question:

In a given number of migration rounds, can we obtain a layout by making changes to the existing layout so that the resulting layout will be the best possible layout that we can obtain within the specified number of rounds?

Of course, such a layout is interesting only if it is significantly better than the existing layout for the new demand pattern.

We approach the problem of finding a good layout that can be obtained in a specified number of rounds by trying to find a sequence of layouts. Each layout in the sequence can be transformed to the next layout in the sequence by applying a small set of changes to the current layout. These changes are computed so that they can be applied within one round of migration (a disk may be involved in at most one transfer per round).

We show that by making these changes even for a small number of consecutive rounds, the existing placement that was computed for the old demand pattern can be transformed into one that is almost as good as the best layout for the new demand pattern.

Our method can therefore be used to *quickly* transform an existing placement to deal with changes in the demand pattern. We *do not* make any assumptions about the type of demand changes – hence the method can be used to quickly deal with any type of change in the demands. We also show that the problem of finding an optimal set of changes that can be applied in one round is NP-hard (see Appendix A.1 for the proof). The proof demonstrates that some unexpected data movement patterns can yield a high benefit.

In the remaining part of the introduction, we present the model and the assumptions made, and restate our result formally.

1.1 Model summary We consider the following model for our storage system. There are N parallel disks that form a *Storage Area Network*. Each disk has a storage capacity of K and has a load handling capacity (or bandwidth) of L.

The efficiency of the system depends crucially on the *data layout* pattern that is chosen for the disks. This data layout pattern or data placement specifies for each item, which set of disks it is stored on (note that the whole item is stored on each of the disks specified by the placement, so these are copies of the item). The next problem is that of mapping the demand for data to disks. Each disk has an upper bound on the total demand that can be mapped to that disk. A simple way to find an optimal assignment of demand to disks, is by running a single network flow computation in an appropriately defined graph (see Section 2.1).

Different communication models can be considered based on how the disks are connected. We use the same model as in [2, 7] where the disks may communicate on any matching; in other words, the underlying communication graph allows for communication between any pair of devices via a matching (e.g., as in a switched storage network with unbounded backplane bandwidth). *This model best captures an architecture of parallel storage devices that are connected on a switched network with sufficient bandwidth. This is most appropriate for our application.* This model is one of the most widely used in all the work related to gossiping and broadcasting. These algorithms can also be extended to models where the size of the matching in each round is constrained [9]. This can be done by a simple simulation, where we only choose a maximal subset of transfers to perform in each round.

Suppose we are given an initial demand pattern $\mathcal{I}$. We use this to create an initial layout $L_{\mathcal{I}}$. Over time, the demand pattern for the data may change. At some point of time the initial layout $L_{\mathcal{I}}$ may not be very efficient. At this point the storage manager may wish to re-compute a new layout pattern. Suppose the target demand pattern is determined to be $\mathcal{T}$ (this could be determined based on the recent demand for data, or based upon projections determined by previous historical trends). Our goal is to migrate data from the current layout to a new layout. We would like this migration to complete quickly since the system is running inefficiently in addition to using a part of its local bandwidth for migrating data. It is therefore desirable to complete the conversion of one layout to another layout quickly. However, note that previous methods completely ignored the current layout and fixed a target layout $L_{\mathcal{T}}$ based on the demand $\mathcal{T}$. *Is it possible that there are layouts $\mathcal{L}'$ with the property that they are almost as good as $L_{\mathcal{T}}$, however, at the same time we can "convert" the initial layout $L_{\mathcal{I}}$ to $\mathcal{L}'$ in very few rounds (say compared to the number of rounds required to convert $L_{\mathcal{I}}$ to $L_{\mathcal{T}}$)?* It is our objective to consider this central question in this paper. In fact, we answer the question in the affirmative by doing a large set of

experiments.

To do this, we define the following *one round problem*. Given a layout $L_{\mathcal{P}}$ and a demand distribution $\mathcal{T}$, our goal is to find a one round migration (a matching), such that if we transfer data along this matching, we will get the maximum increase in utilization. In other words, we will "convert" the layout $L_{\mathcal{P}}$ to a new layout $L_{\mathcal{P}+1}$, such that we get the maximum utilization, and the new layout is obtainable from the current layout in one round of migration.

Now we can simply use an algorithm for the *one round problem* repeatedly by starting with the initial layout $L_{\mathcal{I}}$, and running ℓ iterations of the one round algorithm. We will obtain a layout $L_{\mathcal{I}+\ell}$, which could be almost as good is the target layout $L_{\mathcal{T}}$.

Of course there is no reason to assume that repeatedly solving the one round problem will actually yield an optimal solution for the ℓ round version of this problem. However, as we will see, this approach is very effective.

2 The problem

2.1 Example Since the formal definition of the problem will involve a lot of notation, we will first informally illustrate the problem and our approach using an example. In this example, we will show an initial demand distribution $\mathcal{I}$; an initial placement for this distribution $L_{\mathcal{I}}$; we will then show the changed demand distribution $\mathcal{T}$. We will show why the initial placement $L_{\mathcal{I}}$ is inadequate to handle the changed demand distribution $\mathcal{T}$. We will then show how a small change (a one-round migration) to the initial placement $L_{\mathcal{I}}$ results in a placement that is optimal for the new demand distribution.

In this toy example, we consider a storage system that consists of 4 identical disks. Each disk has storage capacity of 3 units and load capacity (or bandwidth) of 100 units. There are 9 data items that need to be stored in the system. The initial demand distribution $\mathcal{I}$ and the new demand distribution $\mathcal{T}$ are as follows:

Item	Initial demand	New Demand
A	130	55
B	90	55
C	40	20
D	30	60
E	25	5
F	25	10
G	25	15
H	22	70
I	13	110

The placement $L_{\mathcal{I}}$ (which in this case is also an optimal placement) obtained using the sliding window algorithm[2] for the demand distribution above is as follows (the numbers next to the items on disks indicates the mapping of demand to that copy of the item):

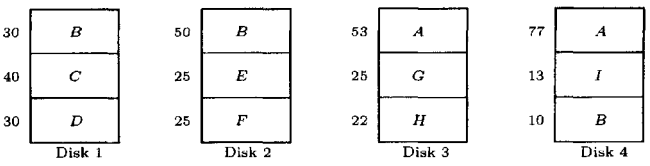


Figure 1: Optimal placement $L_{\mathcal{I}}$ for the initial demand distribution $\mathcal{I}$, satisfies all the demand. Storage capacity K=3, Bandwidth L=100. In addition to producing the layout the sliding window algorithm finds a mapping of demand to disks, which is optimal for the layout computed.

To determine the maximum amount of demand that the current placement $L_{\mathcal{I}}$ can satisfy for the new demand distribution $\mathcal{T}$, we compute the max-flow in a network constructed as follows. In this network we have a node corresponding to each item and a node corresponding to each disk. We also have a source and a sink vertex. We have edges from item vertices to disk vertices if in the placement $L_{\mathcal{I}}$, that item was put on the corresponding disk. Capacities of edges from the source to every item is equal to the demand for that item in the new distribution. The rest of the edges have capacity equal to the disk bandwidth. Using the flow network above, we can re-assign the demand $\mathcal{T}$ using the same placement $L_{\mathcal{I}}$ as given in Figure 3. Figure 2 shows the flow network obtained by applying the construction described above, corresponding to the initial placement $L_{\mathcal{I}}$ and new demand $\mathcal{T}$.

A small change can convert $L_{\mathcal{I}}$ to an optimal placement. In general, we would like to find changes that can be applied to the existing placement in a single round and get a placement that is close to an optimal placement for the new demand distribution. In a round a disk can either be the source or the target of a data transfer but not both. In fact, in this example a single change that involves copying an item from one disk to another is sufficient (and does not involve the other two disks in data transfers). This is illustrated in Figure 4.

We stress that we are not trying to minimize the total number of data transfers, but simply find the *best* set of changes that can be applied in parallel to modify the existing placement for the new demand distribution.

We compare this approach to that of previous works [9, 6] which completely disregard the existing placement

[2]The sliding window algorithm proposed by Shachnai and Tamir [12] is currently the best practical algorithm for this problem. For more on the sliding window algorithm and its performance, see [5].

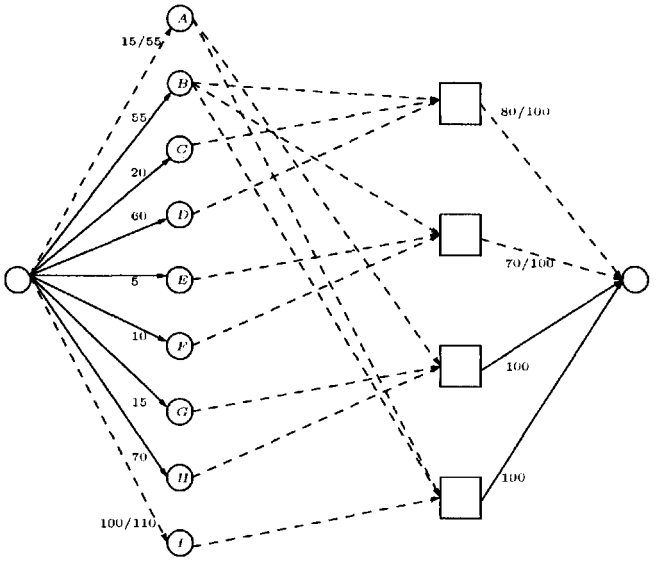


Figure 2: Flow network to determine maximum benefit of using placement $L_{\mathcal{I}}$ with demand distribution $\mathcal{T}$. $L_{\mathcal{I}}$ is sub-optimal for $\mathcal{T}$ and can only satisfy 350 out of a maximum of 400 units of demand. Saturated edges are show using solid lines.

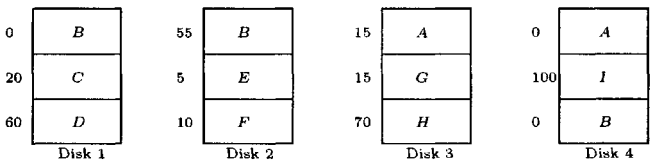


Figure 3: Maximum demand that placement $L_{\mathcal{I}}$ can satisfy for the new demand distribution $\mathcal{T}$. $L_{\mathcal{I}}$ is sub-optimal for $\mathcal{T}$ and can only satisfy 350 out of a maximum of 400 units of demand.

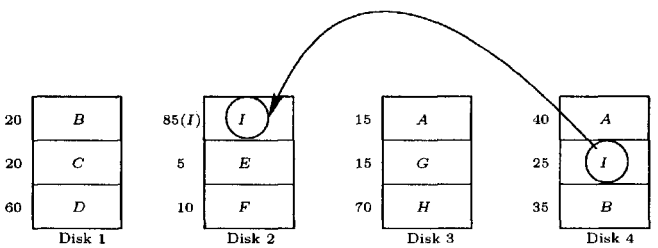


Figure 4: Removing item B from disk 2 and replacing it with a copy of item I from disk 4 converts $L_{\mathcal{I}}$ to an optimal placement $\mathcal{L}'$ for the new demand distribution $\mathcal{T}$. The placement shown above is optimal for $\mathcal{T}$ and satisfies all demand.

and simply try to minimize the number of parallel rounds needed to convert the existing placement to an optimal placement for the new demand distribution. In Fig. 6, we show that using the old approach, it takes 4 rounds of transfers to achieve what our approach did in a single round (and using just one transfer). In Figure 5 an optimal placement $L_{\mathcal{T}}$ is recomputed [3] for the new demand distribution $\mathcal{T}$. We show in Figure 6 the smallest set of transfers required to convert $L_{\mathcal{I}}$ to $L_{\mathcal{T}}$. *Note that both placement $\mathcal{L}'$ (obtained after the transfer shown in Figure 4 is applied) and placement $L_{\mathcal{T}}$ shown in Figure 5 are optimal placements for the new demand distribution $\mathcal{T}$.* Note that this is an optimal solution that also addresses the space constraint on the disk (this property is not actually maintained by the data migration algorithms developed earlier [9]).

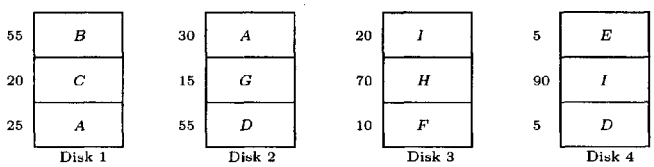


Figure 5: Placement $L_{\mathcal{T}}$. Output of the Sliding window algorithm for the new demand distribution $\mathcal{T}$.

2.2 Formal definition The storage system consists of N disks. Each disk has load-capacity of L and a storage-capacity of K. We have m items, each item j has size 1 and demand ℓ_j. This constitutes an m-dimensional demand distribution $\ell = (\ell_1, \ldots, \ell_m)$. An m-dimensional *placement* vector p_i for a disk i is $(p_{i1}, \ldots, p_{im})$ where p_{ij} are $0-1$ entries indicating that item j is on disk i. An m-dimensional *demand* vector d_i for a disk i is $(d_{i1}, \ldots, d_{im})$ where d_{ij} is the demand for item j assigned to disk i. Define $\mathcal{V}(\{d_i\}) = \sum_i \sum_j d_{ij}$ as the benefit of the set of demand vectors $\{d_i\}$. A set of placement and demand vectors that satisfy the following constraints is said to constitute a feasible placement and demand assignment:

1. $\sum_j p_{ij} \leq K$ for all disks i. This ensures that the storage-capacity is not violated.
2. $\sum_j d_{ij} \leq L$ for all disks i. This ensures that the load-capacity is not violated.
3. $d_{ij} \leq p_{ij}\ell_j$. This ensures that the demand for an item j is routed to disk i only if that item is present on disk i.
4. $\sum_i d_{ij} \leq \ell_j$ for all items j. This ensures that no more than the total demand for an item is packed.

A one-round-migration is essentially a matching on the set of disks. More formally, a one-round-migration is a 0-1 function $\Delta(s_d, s_i, t_d, t_i)$ where $s_d, t_d \in \{1, \ldots, N\}$

[3] Using the sliding window algorithm for computing a placement for a given demand.

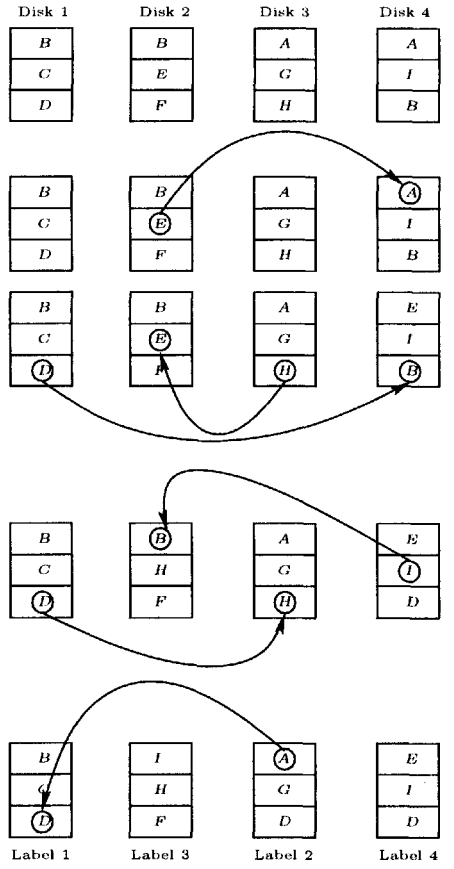


Figure 6: Transforming $L_{\mathcal{I}}$ to $L_{\mathcal{T}}$ takes 4 rounds. Note that the disks here will need to be renumbered to match the sliding window output. Final disk 2 corresponds to disk 3 in the sliding window output, final disk 3 corresponds to disk 2 in the sliding window output.

and $s_i, t_i \in \{1, \ldots, m\}$. Here s_d is the source disk, s_i is the source item, t_d is the target disk, t_i is the target item. Further, $\Delta(.)$ has to satisfy the following conditions:

1. $\sum_{t_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i) \leq 1$ for all disks s_d. This ensures that a disk can be the source for at most one transfer.

2. $\sum_{s_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i) \leq 1$ for all disks t_d. This ensures that a disk can be the target for at most one transfer.

3. $\left(\sum_{s_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i)\right) + \left(\sum_{t_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i)\right) \leq 1$ for all disk pairs $s_d = t_d$. This ensures that a disk can simultaneously not be both a source and a target.

4. $(s_d = t_d) \Rightarrow \Delta(s_d, *, t_d, *) = 0$. This ensures that there are no self loops in the transfer graph.

5. $\sum_{t_d} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i) \leq p_{s_d s_i}$ for all disks s_d and items s_i. This ensures that a disk s_d can be source of an item s_i only if that item is on that disk (i.e. $p_{s_d s_i} = 1$).

We can apply this function to an existing placement to obtain a new placement as follows. If $\Delta(s_d, s_i, t_d, t_i) = 1$, then set $p_{t_d t_i} = 0$ and $p_{t_d s_i} = 1$. We compute the optimal demand assignment for the new placement using max-flow.

ONE-ROUND-MIGRATION: When given an initial demand distribution $\ell_{initial}$, a corresponding set of feasible placement vectors $\{p_i\}$, and demand vectors $\{d_i\}$ and a final demand distribution ℓ_{final}, the problem asks for a one-round-migration $\Delta(.)$ that when applied to the initial placement yields placement vectors $\{p_i^*\}$ and demand vectors $\{d_i^*\}$ such that $\mathcal{V}(\{d_i^*\})$ is maximized.

We show that this problem is NP-Hard (See Appendix A.1 for a proof).

3 Algorithm for one round migration

For any disk d, let $I(d)$ denote the items on that disk.

Corresponding to any placement $\{\mathbf{p_i}\}$ (a placement specifies for each item, which set of disks it is stored on), we define the corresponding flow graph $G_p(V, E)$ as follows. We add one node a_i to the graph for each item $i \in \{1 \ldots m\}$. We add one node d_j for each disk $j \in \{1 \ldots N\}$. We add one source vertex s and one sink vertex t. We add edges (s, a_i) for each item i. Each of these edges have capacity $\texttt{demand}(i)$ (where $\texttt{demand}(i)$ is the demand for item i). We also add edges (d_j, t) for each disk j. These edges have capacity L (where L is the load capacity of disk j). For every disk j and for every item $i \in I(j)$, we add an edge (a_i, d_j) with capacity L.

The algorithm starts with the initial placement and works in phases. At the end of each phase, it outputs a pair of disks and a transfer corresponding to that disk pair.

We determine the disk and transfer pair as follows. Consider a phase r. Let $\{\mathbf{p_i}\}_r$ be the current placement. For every pair of disks d_i and d_j, for every pair of items (a_i, a_j) in $I(d_i) \times I(d_j)$, modify the placement $\{\mathbf{p_i}\}_r$ to obtain $\{\mathbf{p_i}'\}_r$ by overwriting a_j on d_j with a_i. Compute the max-flow in the flow graph for the placement $\{\mathbf{p_i}'\}_r$. Note down the max-flow value and revert the placement back to $\{\mathbf{p_i}\}_r$. After we go through all pairs, pick the (a_i, a_j) transfer pair and the corresponding (d_i, d_j) disk pair that resulted in the flow-graph with the largest max-flow value. Apply the transfer (a_i, a_j) modifying placement $\{\mathbf{p_i}\}_r$ to obtain $\{\mathbf{p_i}\}_{r+1}$ - which will be the starting placement for the next phase. We can no longer use disks d_i and d_j in the next phase. Repeat until there is no pair that can increase the max-flow or till we run

out of disks.

4 Speeding up the algorithm

The algorithm described in Section 3 recomputes max-flow in the flow graph from scratch when evaluating each move. Recall that the algorithm proceeds in phases and at the end of each phase, it identifies a pair of disks (d_i, d_j) and a $(a_i \in I(d_i), a_j \in I(d_j))$ transfer for that pair of disks.

We can speed up the algorithm by observing that the max-flow value increases monotonically from one phase to the next and therefore we need not recompute max-flow from scratch for each phase. Rather, we compute the residual network for the flow graph once and then make incremental changes to this residual network for each max-flow computation. All max-flow computations in this version of the algorithm are computed using the Edmonds-Karp algorithm (see [1]). Let G_i denote the residual graph at the end of phase i. Let G_0 be the residual graph corresponding to the initial graph. All max-flow computations in phase $i+1$, we begin with the residual graph G_i and find augmenting paths (using BFS on the residual graph) to evaluate the max-flow. After each transfer pair in phase $i+1$ is considered, we undo the changes to the residual graph and revert back to G_i. At the end of phase $i+1$, we apply the best transfer found in that phase, recompute max-flow and use the corresponding residual graph as G_{i+1}.

Even with the speedup, the algorithm needs to perform around 415,000 max-flow computations even for one of the smallest instances (N=60, K=15) that we consider in our experiments. Since we want to quickly compute the one-round migration, too many flow computations are not acceptable. We therefore consider the following variants of our algorithm. In our experiments, we found these variants to yield solutions that are as good as the algorithm described above.

Variant 1: For every pair of disks d_i and d_j, let $I_+(d_i)$ be the set of items on disk d_i that have **unsatisfied demand**. For every pair of items (a_i, a_j) in $I_+(d_i) \times I(d_j)$, overwrite a_j on d_j with a_i, compute the max-flow. Pick the (a_i, a_j) pair that gives the largest increase in the max-flow value. Repeat till there is no pair that can increase the max-flow or until we run out of disks.

Variant 2: For every pair of disks d_i and d_j, let $I_+(d_i)$ be the set of items on disk d_i that have **unsatisfied demand** and $I_-(d_j)$ be the items with **lowest demand** on disk d_j. For every pair of items (a_i, a_j) in $I_+(d_i) \times I_-(d_j)$, overwrite a_j on d_j with a_i, compute the max-flow. Pick the (a_i, a_j) pair that gave the largest increase in the max-flow value. Repeat till there is no pair that can increase the max-flow or until we run out of disks.

All the experimental results that we present in Section 5 are obtained using the second variant (described above). To solve[4] even the largest instances in our experiments, a C (gcc 3.3) implementation of the second variant took only a couple of seconds while the brute force algorithm took on the order of several hours.

5 Experiments

In this section, we describe the experiments used to evaluate the performance of our heuristic and compare it to the old approach to data migration. The framework of our experiments is as follows:

1. (*Create an initial layout*) Run the sliding window algorithm [5], given the number of user requests for each data object.
2. (*Create a target layout*) To obtain a target layout, we take one of the following approaches.
   (a) Shuffle method 1: Initial demand distribution is chosen with Zipf (will be defined later in this section) parameter 0.0 (high-skew). To generate the target distribution, pick 20% of the items and promote them to become more popular items.
   (b) Shuffle method 2: Initial demand distribution is chosen with Zipf parameter 0.0 (high-skew). To generate the target distribution, the lowest popularity item is promoted to become the most popular item.
   (c) Shuffle method 3: The initial demand distribution is chosen with Zipf parameter 1.0 (uniform-distribution). The target distribution is chosen with Zipf parameter 0.0 (high-skew).
   (d) Shuffle method 4: The initial demand distribution is chosen with Zipf parameter 0.0 (high-skew). The target distribution is chosen with Zipf parameter 1.0 (uniform-distribution).
3. Record the number of rounds required by the old data migration scheme to migrate the initial layout to the target layout.
4. Record the layout obtained in each round of our heuristic. Run 10 successive rounds of our one round migration starting from the initial layout.

[4] Experiments were run on a 2.8Ghz Pentium 4C processor with 1GB RAM running Ubuntu Linux 5.04.

The layout output after running these 10 successive rounds of our heuristic will be considered as the final layout output by our heuristic.

We note that few large-scale measurement studies exist for the applications of interest here (e.g., video-on-demand systems), and hence below we are considering several potentially interesting distributions. Some of these correspond to existing measurement studies (as noted below) and others we consider in order to explore the performance characteristics of our algorithms and to further improve the understanding of such algorithms. For instance, a Zipf distribution is often used for characterizing people's preferences.

Zipf Distribution The Zipf distribution is defined as follows:

$$\text{Prob(request for item } i) = \frac{c}{i^{1-\theta}} \quad \begin{matrix} \forall i = 1, \ldots, M \\ \text{and} \\ 0 \leq \theta \leq 1 \end{matrix}$$

$$\text{where} \quad c = \frac{1}{H_M^{1-\theta}} \quad \text{and} \quad H_M^{1-\theta} = \sum_{j=1}^{M} \frac{1}{j^{1-\theta}}$$

and θ determines the degree of skewness. For instance, $\theta = 1.0$ corresponds to the uniform distribution, whereas $\theta = 0.0$ corresponds to the skewness in access patterns often attributed to movies-on-demand type applications. See for instance the measurements performed in [3]. Flash crowds are also known to skew access patterns according to Zipf distribution [8]. In the experiments below, Zipf parameters are chosen according to the shuffle methods described earlier in the section.

We now describe the storage system parameters used in the experiments, namely the number of disks, space capacity, and load capacity (the maximum number of simultaneous user requests that a disk may serve).

In the first set of experiments, we used a value of 60 disks. We tried three different pairs of settings for space and load capacities, namely: (A) 15 and 40, (B) 30 and 35, and (C) 60 and 150.

In the second set of experiments, we varied the number of disks from 10 to 100 in steps of 10. We used a value of K=60, L=150 (this is the 3rd pair of L,K values used in the first set of experiments).

We obtained these numbers from the specifications of modern SCSI hard drives. For example, a 72GB 15,000 rpm disk can support a sustained transfer rate of 75MB/s with an average seek time of around 3.5ms. Considering MPEG-2 movies of 2 hours each with encoding rates of 6Mbps, and assuming the transfer rate under parallel load is 40% of the sustained rate, the disk can store 15 movies and support 40 streams. The space capacity 30 and the load capacity 35 are obtained from using a 150GB 10,000 rpm disk with a 72MB/s sustained transfer rate. The space capacity 60 and the load capacity 150 are obtained by assuming that movies are encoded using MPEG-4 format (instead of MPEG-2). So a disk is capable of storing more movies and supporting more streams. For each tuple of N,L,K and shuffle method we generated 10 instances. These instances were then solved using both our heuristic as well as the old data migration heuristic. The results for each N,L,K and shuffle method tuple were averaged out over these 10 runs.

5.1 Results and Discussion Figures 7, 8, 9, 10 and Tables 1, 2, 3 correspond to the first set of experiments. Figures 12, 13, 14, 15, 11 and Tables 4, 5, 6, 7 correspond to the second set of experiments.

Figures 8, 9, 10, 12, 13, 14, 15 compare the solution quality of our heuristic with that of the old approach. Tables 1, 2, 3, 4, 5, 6, 7 and Figure 7 compares the number of rounds taken by our approach with the number of rounds taken by the old approach to achieve similar solution quality.

We highlight the following observations supported by our experimental results:

- In all our experiments, our heuristic was able to get within 8% of the optimal solution using 10 rounds. This can be seen in all the figures and tables. For instance, see Figure 7.
- In comparison (see Figure 7 and Tables 1, 2, 3, 4, 5, 6, 7), the old scheme took a significantly larger number of rounds. For example, in the case of K=60, L=150 (corresponding to storing video as mpeg-4) the old scheme took over 100 rounds for every shuffle method and for every value of N we used, while our scheme was able to achieve similar solution quality within 10 rounds.
- Response to change in demand distribution: The experiments reveal an interesting behavior of the heuristic. When the target demand distribution is highly skewed, the heuristic's response or the amount of improvement made in successive rounds is linear. In contrast, when the demand is less skewed (i.e. the demand distribution is significantly different from the initial distribution but still the target distribution is not very skewed), the response is much sharper. For example in Figure 11, consider the response curve for shuffle methods 4 and 2 (low-skew) and contrast it with the flat response curves for shuffle methods 1 and 3 (high-skew).
  - Sharp response or diminishing returns: For a concrete example; in Figure 12 the improve-

ment obtained by our heuristic in the first round is almost as high as 10%, but successive improvements taper off quickly. This probably happens because we use a greedy algorithm and most of the gains are made in the first round and since this type of behavior is observed mainly when the demand is less skewed, there are presumably several items that need to replicated.

- Flat response: For a concrete example; in Figure 14 the improvement obtained by our heuristic for N=100 in the first two rounds (1 and 2) is just about twice the benefit obtained in the last two rounds (9 and 10). This is probably because most of the load is concentrated on a few items and there is a large amount of unsatisfied demand. In each round we make more copies of these high popularity items and see almost the same benefit in each round.

- The case for this type of approach (that of making small changes to existing placement in consecutive rounds) is best supported by results from Table 7. This is an example of a case where the existing placement is already very good for the target distribution. The storage manager may wish to do a few rounds of migration to recover the amount of lost load. Our scheme lets the storage manager do such a quick adaptation. In contrast the old scheme takes over 150 rounds on average to achieve comparable results. This is especially unacceptable given that we already start off with a pretty good placement. In fact, shuffle method 4 seemed to consistently trigger expensive migrations in the old scheme while our scheme was able to get close to optimal within a couple of rounds. This is not surprising since the old scheme completely disregards the existing placement.

- Shuffle method 3 seemed to produce "harder" instances for our heuristic compared to the other shuffle methods we tried. This is not surprising since shuffle method 3 makes a drastic change to the demand distribution (moving it from uniform to highly skewed Zipf).

- It is very promising that our scheme performs particularly well for shuffle methods 1 and 2 (which is the type of demand change we expect to see in practice).

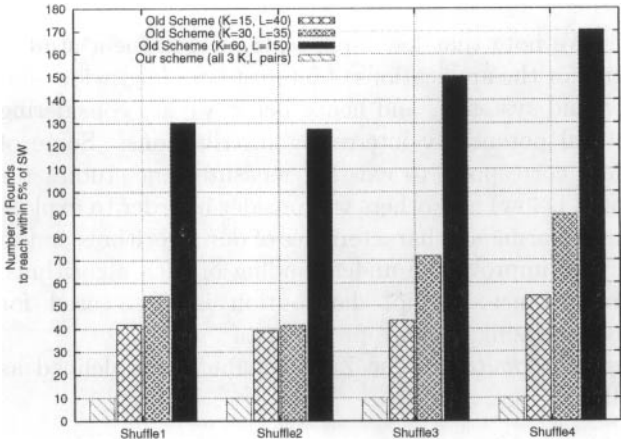


Figure 7: Plot compares the number of rounds that the old migration scheme took to reach within 5% of the optimal solution. We used N=60 and tried each of the shuffle methods for every pair of K and L shown in the plot. Every data point was obtained by averaging over 10 runs. In each of the experiments shown above, our scheme was set to run for 10 consecutive rounds.

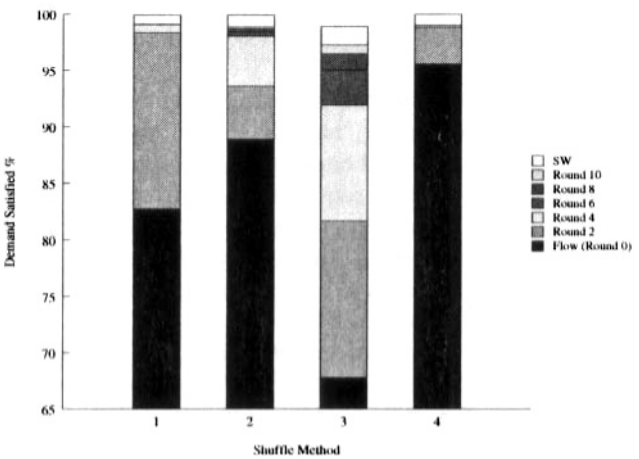


Figure 8: Plot shows improvement obtained by our scheme when presented with instances generated using the different shuffle methods. We used N=60, K=15, L=40 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

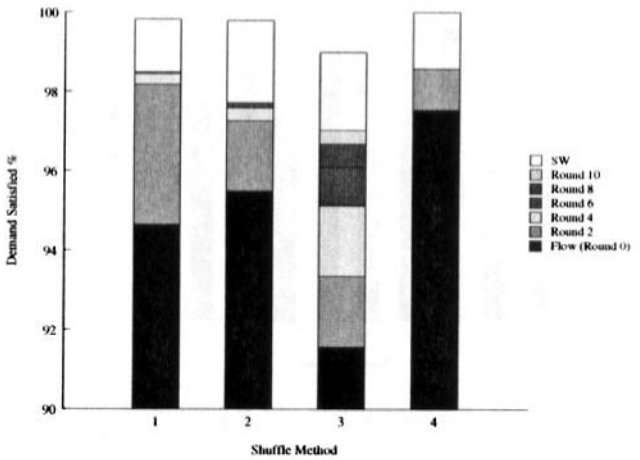


Figure 9: Plot shows improvement obtained by our scheme when presented with instances generated using the different shuffle methods. We used N=60, K=30, L=35 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

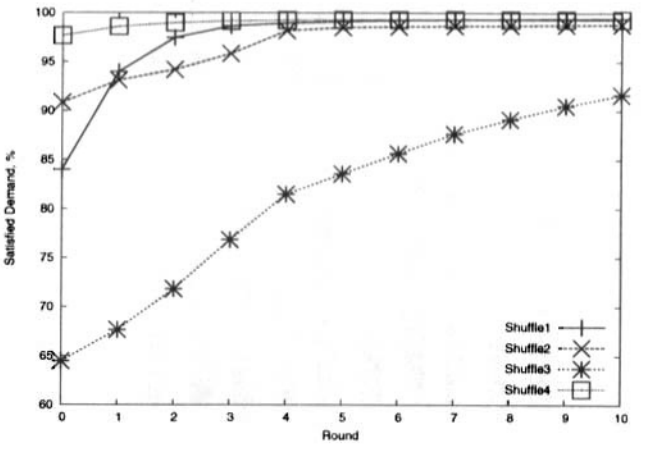


Figure 11: Plot comparing the response of our heuristic to the various shuffle methods. The response to shuffle 3 and shuffle 2 is much flatter than the diminishing returns type of response for shuffle 4 and shuffle 1. We used N=100, K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs.

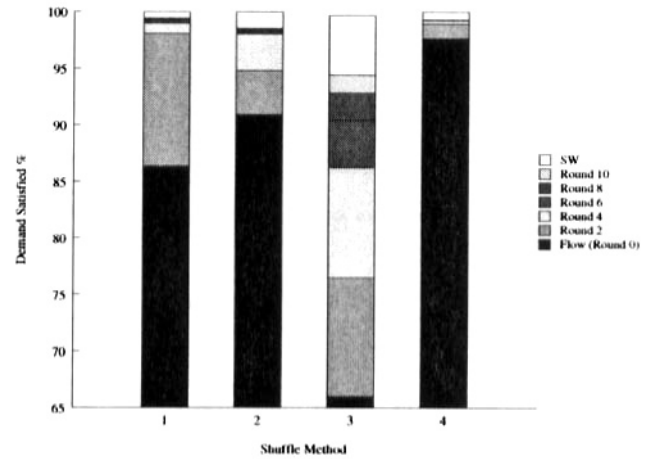


Figure 10: Plot shows improvement obtained by our scheme when presented with instances generated using the different shuffle methods. We used N=60, K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

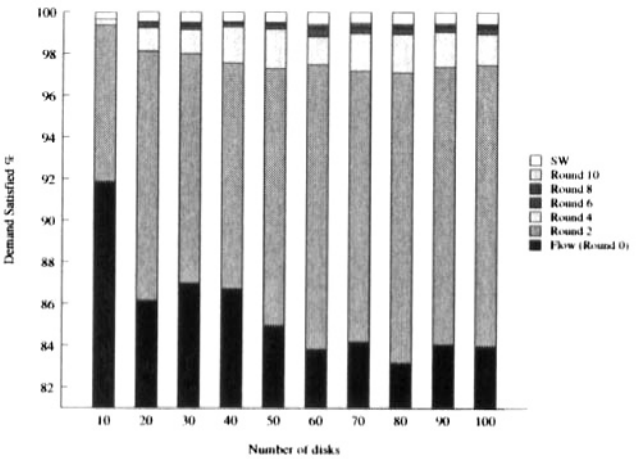


Figure 12: Performance of our scheme with varying number of disks for shuffle method 1. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

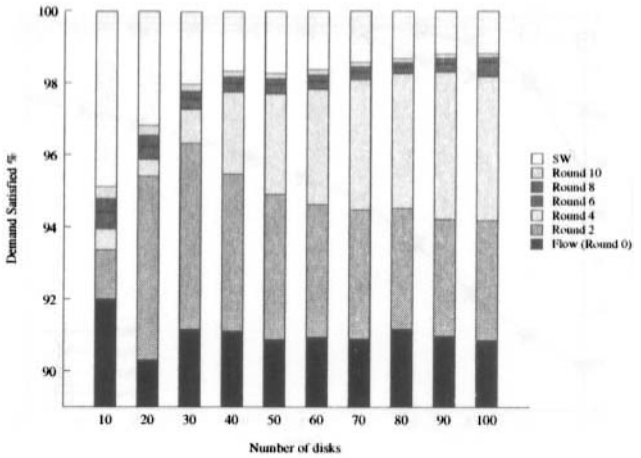


Figure 13: Performance of our scheme with varying number of disks for shuffle method 2. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

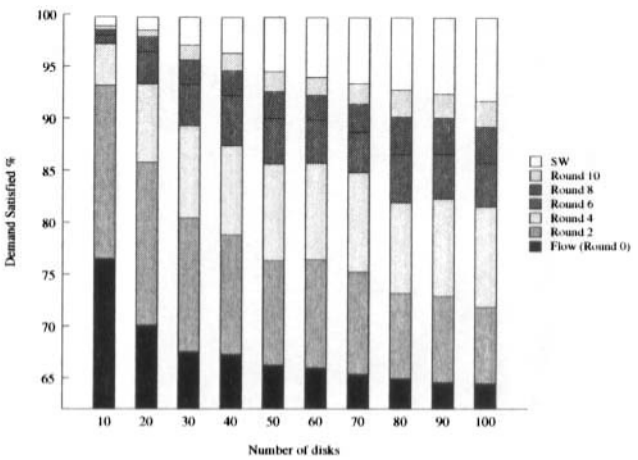


Figure 14: Performance of our scheme with varying number of disks for shuffle method 3. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

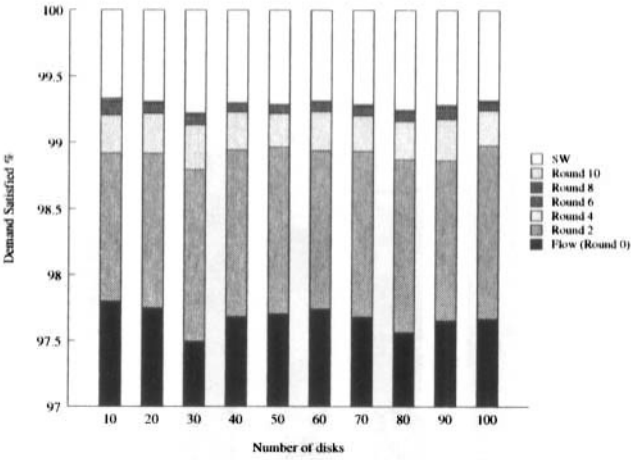


Figure 15: Performance of our scheme with varying number of disks for shuffle method 4. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	**10**	99.10	**41.8**	99.92
2	**10**	98.86	**39.1**	99.92
3	**10**	97.26	**43.7**	98.91
4	**10**	99.04	**54.2**	100

Table 1: Comparison of old scheme with our scheme for N=60, K=15, L=40

Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	**10**	98.50	**54.2**	99.83
2	**10**	97.72	**41.6**	99.79
3	**10**	97.02	**71.8**	98.99
4	**10**	98.57	**89.9**	100

Table 2: Comparison of old scheme with our scheme for N=60, K=30, L=35

Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	**10**	99.41	**130.3**	99.98
2	**10**	98.54	**127.3**	99.99
3	**10**	94.41	**150.4**	99.68
4	**10**	99.30	**170.5**	100

Table 3: Comparison of old scheme with our scheme for N=60, K=60, L=150

6 Conclusion

We proposed a new approach to deal with the problem of changing demand. We defined the one-round-migration problem to aid us in our effort. We showed that the one-round-migration problem is NP-Hard and

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	99.64	**104.8**	99.98
20	**10**	99.52	**111.8**	99.99
30	**10**	99.50	**121**	99.99
40	**10**	99.53	**121.9**	99.98
50	**10**	99.51	**125.9**	99.99
60	**10**	99.41	**128.2**	99.98
70	**10**	99.46	**128.5**	99.99
80	**10**	99.42	**135.4**	100
90	**10**	99.45	**138.6**	99.99
100	**10**	99.43	**136.2**	99.99

Table 4: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 1. K=60, L=150.

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	95.12	**103.3**	99.98
20	**10**	96.82	**109.4**	99.98
30	**10**	97.96	**119.4**	99.97
40	**10**	98.31	**119.7**	99.98
50	**10**	98.26	**124.7**	99.99
60	**10**	98.37	**127.4**	100
70	**10**	98.57	**129.3**	100
80	**10**	98.67	**135.3**	100
90	**10**	98.81	**134.4**	100
100	**10**	98.82	**137.5**	100

Table 5: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 2. K=60, L=150.

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	98.89	**126.2**	99.75
20	**10**	98.46	**133**	99.71
30	**10**	97.03	**138.8**	99.7
40	**10**	96.22	**144.1**	99.68
50	**10**	94.45	**146.5**	99.69
60	**10**	93.89	**149.4**	99.67
70	**10**	93.29	**149.8**	99.68
80	**10**	92.69	**154.6**	99.66
90	**10**	92.29	**152.8**	99.65
100	**10**	91.63	**155.9**	99.66

Table 6: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 3. K=60, L=150.

that unexpected data movement patterns can yield high benefit. We gave heuristics for the problem. We gave experimental evidence to suggest that our approach of doing a few rounds of one-round-migration consecutively performs very well in practice. In particular, in all our experiments they were able to quickly adapt the existing

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	99.33	**128.3**	100
20	**10**	99.31	**139.6**	100
30	**10**	99.22	**148.9**	100
40	**10**	99.30	**159.2**	100
50	**10**	99.29	**166.6**	100
60	**10**	99.32	**170.8**	100
70	**10**	99.29	**178.7**	100
80	**10**	99.25	**183.6**	100
90	**10**	99.29	**190.3**	100
100	**10**	99.32	**196.1**	100

Table 7: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 4. K=60, L=150.

placement to one that is close to the optimal solution for the changed demand pattern. We showed that, in contrast, previous approaches took many more rounds to achieve similar solution quality.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] Eric Anderson, Joseph Hall, Jason D. Hartline, Michael Hobbs, Anna R. Karlin, Jared Saia, Ram Swaminathan, and John Wilkes. An experimental study of data migration algorithms. In *WAE '01: Proceedings of the 5th International Workshop on Algorithm Engineering*, pages 145–158, London, UK, 2001. Springer-Verlag.

[3] Ann Louise Chervenak. *Tertiary storage: an evaluation of new applications.* PhD thesis, Berkeley, CA, USA, 1994.

[4] Shahram Ghandeharizadeh and Richard Muntz. Design and implementation of scalable continuous media servers. *Parallel Comput.*, 24(1):91–122, 1998.

[5] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 223–232, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[6] L. Golubchik, S. Khuller, Y. Kim, S. Shargorodskaya, and Y-C. Wan. Data migration on parallel disks. In *Proc. of European Symp. on Algorithms (2004). LNCS 3221*, pages 689–701. Springer, 2004.

[7] Joseph Hall, Jason Hartline, Anna R. Karlin, Jared Saia, and John Wilkes. On algorithms for efficient data migration. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 620–629, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[8] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites, 2002.

[9] Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for data migration with cloning. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 27–36, New York, NY, USA, 2003. ACM Press.

[10] How much information? School of Information Management and Systems. University of California at Berkeley. http://www.sims.berkeley.edu/research/projects/how-much-info 2003/.

[11] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. In *Proceedings of Workshop on Approximation Algorithms (APPROX). LNCS 1913*, pages 238–249. Springer-Verlag, 2000.

[12] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29(3):442–467, 2001.

[13] Introduction to Storage Area Networks. IBM Redbook. http://www.redbooks.ibm.com/.

A Appendix

A.1 Hardness proof Recall that the *Subset-Sum* Problem is known to be NP-complete [4]. The *Subset-Sum* problem is defined as follows: Given a set $S = \{a_1, \ldots, a_n\}$ and a number b, where $a_i, b \in \mathcal{Z}^+$. Does there exist a subset $S' \subset S$ such that $\sum_{a_j \in S'} a_j = b$? Let $\text{sum}(S) = \sum_{a_i \in S} a_i$.

The *One-Round Migration* problem is defined as follows. We are given a collection of identical disks $D_1, \ldots D_N$. Each disk has a storage capacity of K, and a load capacity of L. We are also given a collection of data objects $M_1, \ldots M_M$, and a layout of the data objects on the disks. The layout specifies the subset of K data objects stored on each disk. Each data object M_i has demand u_i. The demand for any data object may be assigned to the set of disks containing that object (demand is splittable), without violating the load capacity of the disks. For a given layout, there may be no solution that satisfies all the demand. Is there a one-round migration to compute a new layout in which all the demand can be satisfied?

A one-round migration is a matching among the disks, such that for each edge in the matching, one source disk may send an item to a disk that it is matched to (half-duplex model).

We show that the One-Round Migration problem is NP-hard by reducing Subset-Sum to it. We will create a set of $N = 3n + 4$ disks, each having capacity two ($K = 2$). There are $4n + 6$ items in all. We will assume that L is very large. The current layout is shown in Figure 16.

The demand for various items is as follows: Demand for G_i is $L - a_i$. Demand for $C_i = \frac{L}{2} + a_i$. Demand for $E_i = \frac{L}{2}$. Demand for $F_i = L - a_i$.

Demand for $A = \text{sum}(A) + \frac{L}{2}$. Demand for $H = \text{sum}(A) + \frac{L}{2}$. Demand for $X = \frac{L}{2}$. Demand for $Y = L - b$. Demand for $Z = L - (\text{sum}(A) - b)$. Demand for $W = \frac{L}{2}$.

If we assume that the demand for C_i is $\frac{L}{2}$ then the assignment shown can satisfy all the demand. We will assume that all but two of the disks are load saturated (total assigned demand is exactly L). *If the demand for C_i increases by a_i*, then we have to re-assign some of this demand. The claim is that all of the demand can be handled after one round of migration if and only if there is a solution to the subset-sum instance. It is clear that a given solution (a matching) can be verified in polynomial time.

($\Rightarrow$) Suppose there is a subset $S' \subset S$ that adds

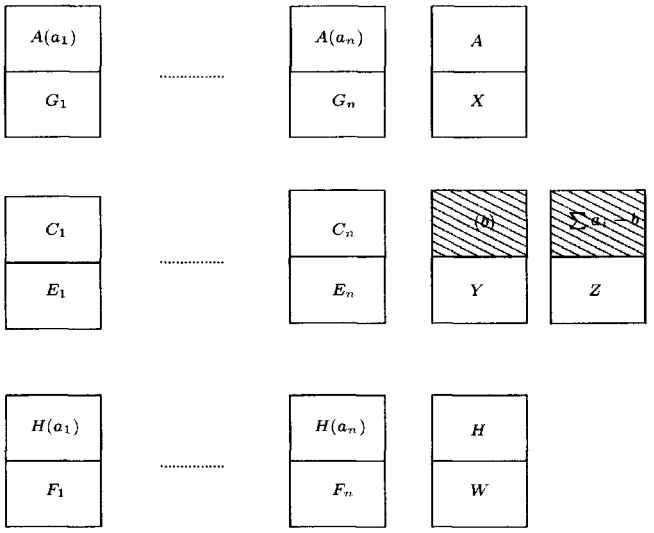


Figure 16: Reduction from SUBSET-SUM to ONE-ROUND-MIGRATION. Shaded portion indicates empty space. Number within brackets following item name indicates the amount of load assigned to the item.

exactly to b. We copy H (from the disk containing H and W) to the disk containing Z, and A (from the disk containing X and A) to the disk containing Y. If $a_i \in S'$ then we copy C_i to the disk containing G_i, and over-write the copy of A on that disk. All clients for A from this set of disks can be moved to the disk containing A and Y. If $a_i \notin S'$ then we copy C_i to the disk containing F_i and over-write the copy of H on that disk. All clients for H from this set of disks can be moved to the disk containing H and Z.

($\Leftarrow$) First note that the total demand is $3nL + 4L$. Since there are $3n + 4$ disks, all disks must be load saturated for a solution to exist. We leave it for the reader to verify that with the current layout there is no solution that meets all the demand. Suppose there exists a one-round migration that enables a solution where all of the demand can be assigned. A new copy has to be created for each C_i, or E_i since the total load for C_i and E_i is $L + a_i$, and exceeds L. Assume w.lo.g that a copy of C_i will be made to handle the excess demand of a_i on this disk. We also assume without loss of generality that $a_i < b$ so moving C_i to one of the disks containing Y or Z would not be of much use in load saturating those disks. The only choice is to decide whether this new copy is made at the expense of a copy of H or at the expense of a copy of A. Note that C_i cannot overwrite any of the other items since only a single copy of these items exists in the system. Since this is a one-round migration, we cannot move a single copy of an item to another disk, and then re-write it subsequently. Note that C_i has to overwrite the corresponding A disk or H disk, otherwise we will be unable to recover all the demand. Since the disks containing Y and Z are also load saturated, we will copy an item onto those disks. Moreover we have to move one item (either A or H) to the disk containing Y. Suppose that A is copied to the disk containing Y and H is copied to the disk containing Z. (The reverse case is similar.) When we shift b amount of demand of A to the disk containing Y, we have to completely remove the demand from a disk containing A, otherwise we will lose some demand. If C_i is moved to a disk containing A then $a_i \in S'$. If C_i is moved to a disk containing H then $a_i \notin S'$. Since C_i over-writes A (H), all of the demand of A (H) is moved out of the disk. Clearly, the total size of S' must be exactly b.

Force-Directed Approaches to Sensor Localization*

Alon Efrat, David Forrester, Anand Iyer, and Stephen G. Kobourov
Department of Computer Science
University of Arizona
{alon,forrestd,anand,kobourov}@cs.arizona.edu

Cesim Erten
Department of Computer Science and Engineering
Isik University, Turkey
cesim@isikun.edu.tr

Abstract

We consider the centralized, anchor-free sensor localization problem. We consider the case where the sensor network reports range information and the case where in addition to the range, we also have angular information about the relative order of each sensor's neighbors. We experimented with classic and new force-directed techniques. The classic techniques work well for small networks with nodes distributed in simple regions. However, these techniques do not scale well with network size and yield poor results with noisy data. We describe a new force-directed technique, based on a multi-scale dead-reckoning, that scales well for large networks, is resilient under range errors, and can reconstruct complex underlying regions.

1 Introduction

Wireless sensor networks are used in many applications, from natural habitat monitoring to earthquake detection; see [1] for a survey. Often, the actual location of the sensors is not known but is necessary for the underlying application, e.g., determining the epicenter of a quake. Further, the location of the sensors can be used to design efficient network routing algorithms [13].

Abstractly, the sensor localization problem can be thought of as a graph layout problem. The true state of the underlying sensor network is captured by a layout D of the source graph G. Given partial information about G (adjacency information, possibly information about edge lengths, or angles between adjacent neighbors), we would like to construct a layout $\hat{D}$ of G that matches D as well as possible. There are many variations of the problem, depending on the quality of the edge length data (obtained using signal strength), or whether some of the vertices know their exact positions (GPS-equipped sensors), or whether the vertices can detect the relative order of their neighbors (obtained by using multiple antennas per sensor). Centralized and distributed algorithms have both been proposed for these problems.

Sensors typically have a *range* that allows them to detect other sensors that fall in that range, thus providing adjacency information for the underlying graph. The strength of the signal, or the time of arrival of the signal are typically used to estimate the actual distance between two sensors. However, sensing neighbors is far from perfect, especially close to the limits. Sensors equipped with GPS are often called *anchors* and while they make the localization problem easier, they are bulky and expensive. Anchor-free sensor networks are more practical but pose greater challenges in localization.

Sensors equipped with multiple antennas can provide *angular information* by reporting the relative order of their neighbors or an estimate on the angle between adjacent neighbors. Multiple antennas add to the cost and size of the sensor, but not nearly as much as in the case of GPS. Once again, the angular information is not perfect, but even allowing for some errors, angular information can be used to find good localizations.

In this paper we focus on the centralized sensor localization problem for anchor-free networks. We consider the cases with or without angular information. We also consider different types of underlying regions for the sensor network: simple convex polygons, simple non-convex polygons, and non-simple polygons. Classic force-directed methods can be augmented to take into account the edge length information. This approach works well for small graphs of up to fifty or so vertices, provided that the graphs are well-connected. For

*This work is supported in part by NSF grant ACR-0222920.

larger graphs, the simple force-directed algorithms fail to reconstruct the vertex locations. We show that multi-scale versions of the force-directed algorithms can extend the utility of these algorithms to graphs with hundreds of vertices, provided that the graphs are defined inside simple convex polygons. Finally, we describe a new multi-scale force-directed approach that incorporates the angular information in a dead-reckoning fashion. This approach can extend the utility of multi-scale force-directed algorithms to graphs with thousands of vertices, defined inside non-convex and even non-simple polygons.

1.1 Related Work

In the last decade the sensor localization problem has received a great deal of attention in the networks and wireless communities, due to the lowering of the production cost of miniature sensors and due to the numerous practical applications, such as environmental and natural habitat monitoring, smart rooms and robot control [1]. Several recent approaches have exploited the natural connections with graph layout algorithms. Priyantha *et al.* [15] propose a new distributed anchor-free layout technique, based on force-directed methods. Gotsman and Koren [9] utilize a stress majorization technique in their distributed method. Neither of these approaches assumes that angular information is available and as a consequence these algorithms need additional assumptions to achieve good results (both approaches assume that sensors are distributed in a simple convex polygon, and Priyantha *et al.* assume that the graph is rigid).

Most of the algorithms that do utilize angular information, assume that a fraction of the sensors is GPS-equipped. Doherty *et al.* [3] formulate the sensor localization problem as a linear or semidefinite program based on both adjacency and angular information. Savvides *et al.* [17] describe an ad-hoc localization system (AHLoS) which employs an anchor-based algorithms for sensor localization using both edge length and angular information. Savarese *et al.* [16] and Niculescu and Nath [14] describe anchor-based algorithms for sensor localization utilizing edge lengths information. Fekete *et al.* [4] use a combination of stochastic, topological, and geometric ideas for determining the structure of boundary nodes of the region, and the topology of the region.

1.2 Our Contributions

We focus on centralized force-directed sensor localization algorithms for anchor-free networks. We consider two variations of the problem: one in which the input contains (noisy) edge lengths information and the other in which the input also contains (noisy) angular information. We perform experiments by varying the sizes of the graphs, in terms of number of vertices and edge density (average vertex degree). We also consider different types of shapes for the region in which the sensors are distributed: simple convex polygons, simple non-convex polygons, and non-simple polygons. Finally, we measure two types of performance metrics: the global quality of the layout and the structure of the boundary of the region.

We describe one new force-directed technique and adapt several standard force-directed technique to the centralized sensor localization problem. Two standard force-directed techniques are those of Fruchterman-Reingold [6] and Kamada-Kawai [11]. If we are only given adjacency information about the underlying graph, these algorithms fail to solve the sensor localization problem even for small graphs. Incorporating the (noisy) edge lengths information works surprisingly well for graphs defined inside simple convex regions.

For larger graphs, the multi-scale graph layout algorithms [7] perform better. However, even these techniques fail to reconstruct graphs defined in non-simple, or non-convex regions.

With the aid of (noisy) angular information, we can extend the utility of multi-scale graph layout algorithms to large graphs with complicated underlying regions. In particular, we show that the new *multi-scale dead-reckoning* algorithm performs well and is tolerant to non-trivial noise for large networks defined in non-simple and non-convex regions.

2 Algorithms, Metrics, and Experiments

In this section we briefly describe the algorithms we implemented, the metrics used to evaluate performance, and our experimental setup.

2.1 Algorithms

We implemented and tested six force-directed algorithms: Fruchterman-Reingold Algorithm (FR), Kamada-Kawai Algorithm (KK), Fruchterman-Reingold Range Algorithm (FRR), Kamada-Kawai Range Algorithm (KKR), Multi-Scale Kamada-Kawai Range Algorithm (MSKKR) and Multi-Scale Dead-Reckoning Algorithm (MSDR). The first two utilize only the graph adjacency information. The next three utilize the graph adjacency information and the edge lengths (range) information. The last algorithm utilizes the graph adjacency information, the edge lengths (range) information and the angular information. Details about these algorithms are provided in the next section.

2.2 Metrics

We compare the performance of various algorithms on different underlying graphs, varying the number of vertices, edge density, as well as the types of regions in which the graphs are defined. We also vary the amount of error in both the edge length and angular information. We implemented six different metrics to capture the performance of the algorithms, some intended to measure the global quality of the layout and the others measuring the quality of the boundary. In this paper, we report the results using the *Frobenius* metrics for comparing the layouts globally and the *BAR* metric for comparing the quality of the boundary reconstruction.

The global quality metrics attempt to measure how the layout $\hat{D}$ created by a given algorithm matches the source layout D. In particular, the Frobenius metric [8] is equivalent to the Frobenius norm of a matrix M whose entries are:

$$M_{ij} = \frac{\hat{d}_{ij} - d_{ij}}{n},$$

where n is the number of sensors, d_{ij} is the actual distance between sensors i and j in D, and $\hat{d}_{ij}$ is the distance between those sensors in the layout $\hat{D}$. Thus, we can measure the global quality of the layout[1] in terms of the Frobenius error:

$$FROB1 = \sqrt{\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\hat{d}_{ij} - d_{ij})^2}.$$

The *boundary alignment ratio* (BAR) is the sum-of-squares normalized error value of a boundary matching. Given the true layout D, we compute its boundary and then compute an approximation by taking a sample of the boundary points B. We compute the same size sample $\hat{B}$ of the boundary of the layout $\hat{D}$ produced by our algorithm. We then apply the iterative closest point algorithm (ICP) [2] to align the two boundaries using rotation and translation. The boundary alignment ratio is defined as:

$$BAR = \frac{\sum_{\hat{p}\in\hat{B}}(\hat{p} - p)^2}{|B|}.$$

[1] The *global energy ratio* (GER) defined by Priyantha *et al.* [15] is similar to the Frobenius metric:

$$GER = \frac{1}{n(n-1)/2}\sqrt{\sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\frac{\hat{d}_{ij} - d_{ij}}{d_{ij}}\right)^2}.$$

While appropriate for comparing the layouts obtained by different algorithms for graphs of the same size, the GER metric is not well-suited to compare the quality of the layout across different graph sizes.

The ICP algorithm first computes a match $\hat{p} \to p$ for each point $\hat{p} \in \hat{B}$, based on nearest neighbors. Next, the ICP algorithm aligns the two layouts D and $\hat{D}$ as well as possible using the BAR metric. This process of nearest-neighbor computation and alignment is repeated until the improvement in the BAR score becomes negligible.

2.3 Experiments

Since we did not have actual sensors to work with, we wrote a plugin for our graph visualization system, Graphael [5], that simulates the placement of the sensors and the reported information from each. Our sensor data generator takes the following parameters as input: number of sensors, average connectivity (density), region to place the sensors in (square-shape, star-shape, etc.), range error, and angle error. All of our regions have the same area so that the size of the region does not affect the performance metric results.

Our data generator fills the region with the given number of sensors placed at random inside it. Then the distances between all pairs of sensors are computed so that we can determine the sensor range that will give us the desired average connectivity. Finally, we connect the sensors that are within the determined sensor range and report the distance between them after incorporating the range error into the actual distances. The range error specifies standard deviation (in percentage) about 100% of the true edge length using Gaussian distribution.

Next we compute the angular information. Each sensor chooses a random direction to be called "north." Then, the sensor detects the clockwise angle from north that each of its neighbors are located at, and angle error is factored in. We then sort these edges by reported angle and generate a mapping from each edge to its next clockwise edge about the node, and store with it the angle to that edge. This procedure guarantees that although error may be present in the reported data, the sum of the reported angles between edges is equal to 360°. Angle error specifies standard deviation (in degrees) about the actual angle from a sensor's declared "north" to an edge using Gaussian distribution.

3 Force-Directed Algorithms for Localization

Some of the most flexible algorithms for calculating layouts of simple undirected graphs belong to a class known as force-directed algorithms. Also known as spring embedders, such algorithms calculate the layout of a graph using only information contained within the structure of the graph itself. In general, force-directed methods define an objective function which maps each graph layout into a number in $\mathcal{R}^+$ representing the energy of the layout. This function is defined in such a

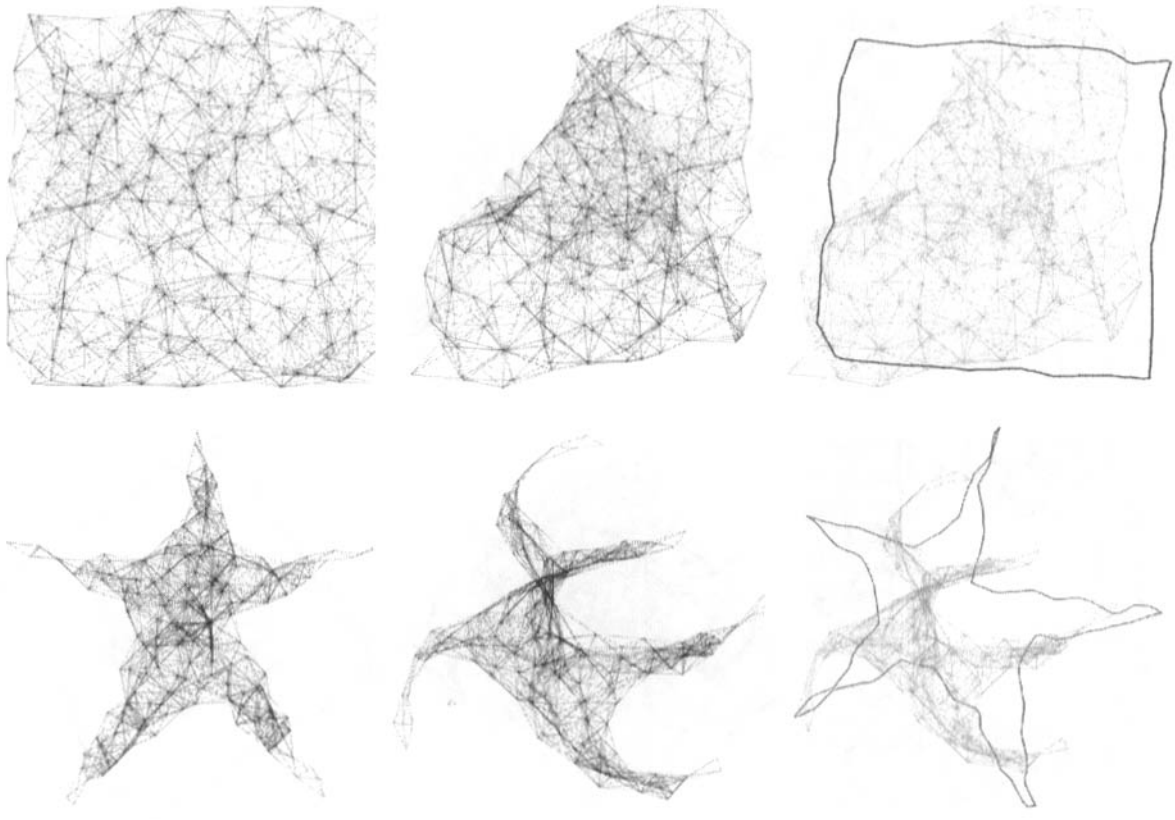


Figure 1: Typical results illustrating input/output/boundary-allignment for KK (top) and FR (bottom) for graphs with 200 vertices inside square and star-shape regions, respectively.

way that low energies correspond to layouts in which adjacent nodes are near some pre-specified distance from each other, and in which non-adjacent nodes are well-spaced. A layout for a graph is then calculated by finding a (often local) minimum of this objective function.

The Fruchterman-Reingold (FR) algorithm [6] defines an attractive force function for adjacent vertices and a repulsive force function for non-adjacent vertices. The vertices in the layout are repeatedly moved according to this function until a low energy state is reached. FR, relies on *edgeLength*: the unweighted "ideal" distance between two adjacent vertices. The displacement of a vertex v of G is calculated by $F_{FR}(v) = F_{a,FR} + F_{r,FR}$, where:

$$F_{a,FR} = \sum_{u \in Adj(v)} \frac{\text{dist}_{R^n}(u,v)^2}{\text{edgeLength}^2}(\text{pos}[u] - \text{pos}[v]),$$

$$F_{r,FR} = \sum_{u \in Adj(v)} s \cdot \frac{\text{edgeLength}^2}{\text{dist}_{R^n}(u,v)^2} \cdot (\text{pos}[u] - \text{pos}[v]).$$

Alternatively, forces between the nodes can be computed based on their graph theoretic distances, determined by the lengths of shortest paths between them. The Kamada-Kawai (KK) algorithm [11] uses spring forces proportional to the graph theoretic distances. The displacement of a vertex v of G is calculated by $F_{KK}(v)$:

$$\sum_{\forall u \neq v} \left(\frac{\text{dist}_{R^n}(u,v)^2}{\text{dist}_G(u,v)^2 \cdot \text{edgeLength}^2} - 1 \right) (\text{pos}[u] - \text{pos}[v]).$$

Since neither FR, nor KK use the range information, the resulting layouts $\hat{D}$ are not of the same scale[2] as the original graph layout D. Still, for small graphs (50-100 vertices) in simple underlying regions these algorithms often manage to reconstruct the underlying

[2]The notions of "scale" and "scalability" can be confusing. In this context, "scale" refers to the edge lengths of the graph. In general, when we refer to "scalable algorithms" we mean algorithms whose performance does not degrade with larger input sizes as measured by the number of vertices and edges in the input graphs. Finally, when we refer to "multi-scale" algorithms we mean multi-level, multi-stage type algorithms.

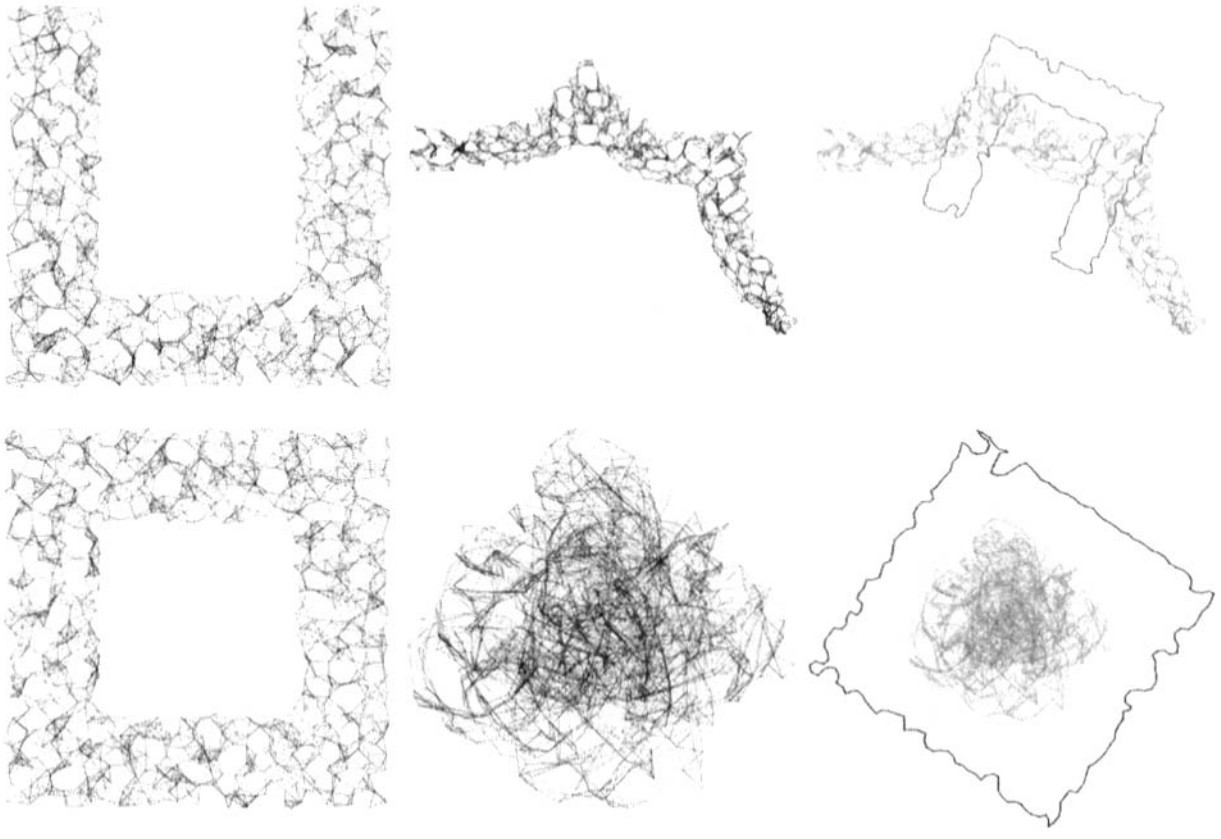


Figure 2: Typical results illustrating input/output/boundary-allignment for KKR (top) and FRR (bottom) for graphs with 1000 vertices, density 8, range error 10%, angle error 10°, inside U-shape and donut-shape regions, respectively.

structure, as well as the boundaries. For larger graphs these algorithms exhibit the typical problems of fold-over and global distortion; see Fig. 1. To address the scale issue, we extend these algorithms to take into account the range information.

3.1 Range Extensions

In range version of the Fruchterman-Reingold algorithm, FRR, the forces are defined by $F_{FRR}(v) = F_{a,FRR} + F_{r,FRR}$. The difference between the FR and FRR algorithms is in the definition of *edgeLength*. While in FR the ideal *edgeLength* was the same for all edges, in FRR *edgeLength* is different for different edges and is defined by the reported distance between the corresponding pair of vertices. In a sensor network setup, this information comes from the range of the sensors and strength-of-signal or time-of-arrival data.

In the range version of Kamada-Kawai, KKR, we incorporate the range data and use the weighted graph distance instead of the unweighted graph distance, $dist_G(u, v)$. Similar to KKR, the weight of the edges comes from the range of the sensors and strength-of-signal or time-of-arrival data.

FRR and KKR perform well on some graphs and not so well on others; see Fig. 2. FRR works well for small graphs of fifty to one hundred vertices, defined in simple convex shapes. However, larger graphs pose serious problems as FRR often settles in a local minimum. KKR, performs well on many large graphs, given enough iterations. Yet, KKR performs poorly on graphs defined in non-convex shapes. As we show in Section 4 the poor performance on non-convex shapes of algorithms based on the Kamada-Kawai approach can be addressed with the help of angular information.

3.2 Multi-Scale Extensions

One of the problems with the classic force-directed algorithms, such as Fruchterman-Reingold and Kamada-Kawai, is that they typically do not scale to larger graphs. One way to avoid this problem is to use multi-scale variants of these algorithms. In particular, multi-scale variants of the Kamada-Kawai algo-

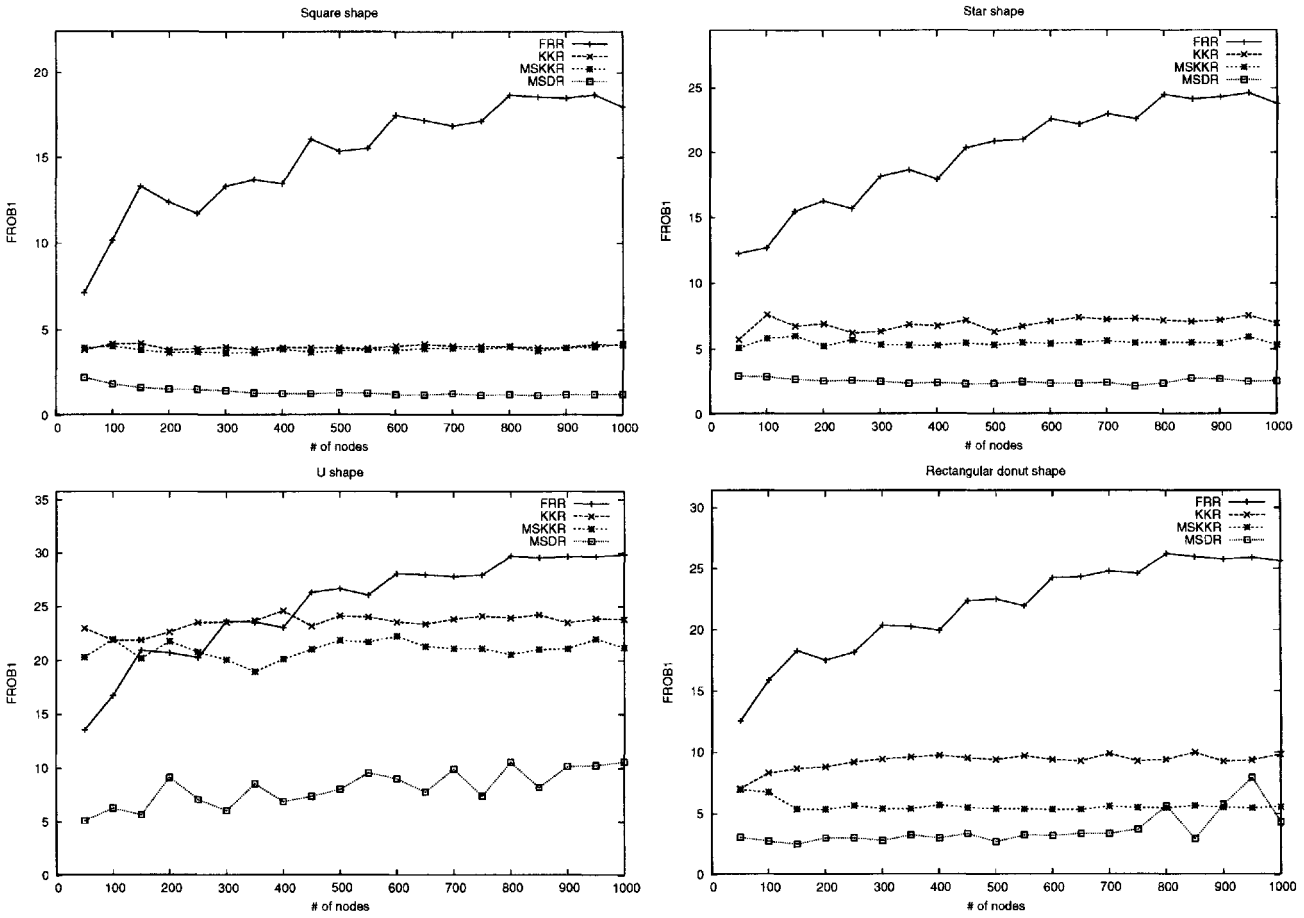


Figure 3: Comparison between FRR, KKR, MSKKR, and MSDR algorithms measured by the Frobenius metric across square-shape, star-shape, U-shape, and donut-shape graphs with 50 to 1000 vertices. There were twenty trials per shape, using graphs with density 8, range error of 20% and angle error of 10°.

rithm have already been shown to produce good results in traditional graph drawing setting [7, 10]. Our multi-scale algorithm, MSKKR, uses these ideas to extend the utility of KKR to larger graphs.

The MSKKR algorithm relies on a *filtration of the vertices, intelligent placement, and multi-scale refinement.* Given $G = (V, E)$, we use a maximal independent set filtration $F : V = V_0 \supset V_1 \supset \ldots, \supset V_k \supset \emptyset$, such that each V_i is a maximal subset of V_{i-1} for which the graph distance between any pair of vertices is at least $2^{i-1}+1$. It is easy to see that given this definition $k = O(\log n)$.

The vertices in V_k are placed first, based on an estimate of their graph distances. Then the vertices in each successive set in the filtration are placed based on their graph distances from the vertices that have already been placed, followed by a refinement of the current layout. Details of this approach are discussed in [7].

While the quality of the layouts obtained by KKR are comparable to those obtained by MSKKR, the multi-scale approach is much faster and offers a better chance of getting right some of the global details of the placement. As the charts in Fig. 3 indicate, MSKKR performs especially well for star-shapes and donut-shapes. The same figure indicates that just as KKR, MSKKR has problems with U-shape graphs that the next algorithm can address.

4 Multi-Scale Dead-Reckoning Algorithm

The KK, KKR, and MSKKR algorithms use either the graph theoretical distance or a weighted version of this distance when the range data is taken into account. This approach provides layouts that typically match the underlying graphs. Non-convex underlying shapes, however, yield poor results even for MSKKR. This is a problem exhibited by all of the algorithms considered so far.

Consider the sensor network obtained by distributing sensors in a U-shape region. Both the Kamada-

Figure 4: A typical problem with graphs defined in non-convex shapes. Input/output/boundary-allignment for MSKKR for a graph with 1000 vertices, density 8, range error of 10% and angle error of 10°.

Kawai and Fruchterman-Reingold style algorithm would typically produce layouts in which the bends have been straightened; see Fig. 4. This is not a flaw of the algorithms but a byproduct of the way they compute the layouts as both of these algorithms attempt to place vertices whose graph distances are large, as far away from each other as possible. Pairs of vertices at the tips of the U-shape are at maximum graph distance from each other, but their Euclidean distance is small. Thus, to be able to reconstruct layouts of graphs defined in non-convex or non-simple regions, we need additional information. Most previous approaches rely on anchors (vertices with GPS) but these are too costly and bulky. Instead, angular information (if available) can be used with great effect to improve the quality of the layouts. With this in mind, we propose the multi-scale dead-reckoning (MSDR) algorithm.

4.1 Dead-Reckoning

Dead-reckoning has been used for centuries as a method of estimating the current position of a moving object by applying the direction and distance traveled to a previously determined position [12]. It is a common method for calculating the position of a mobile robot, using the robot's measurements of traveled distance and turns made. Although the problem we are considering is a static problem, we can use this technique to obtain better estimates for the relative positions of two distant sensor nodes. Given range and angular information, we can compute the distance between two vertices x and y in the graph using this idea. We call that distance $dr(x, y)$.

Suppose we want to calculate the dead-reckoning distance from vertex A to a vertex D. Let node C be D's predecessor in the shortest path from A to D, and let B be C's predecessor; see Fig. 5. Assume that $dr(A, B)$ and $dr(A, C)$ have already been calculated and that we also know the orientation of $\triangle BCA$. The $\angle BCD$ is also known since the angle between edges on node C is part of the source data, and the lengths of the edges from B to C and from C to D are known as well. To reduce the number of special cases, we convert this angle to a clockwise angle by negating it if it's counter-clockwise.

Ultimately, we want to calculate $\angle ACD$ so that we can determine $dr(A, D)$ via the law of cosines. To do this, we first compute $\angle BCA$ using the law of cosines: $dr(A,B)^2 = edge(B,C)^2 + dr(A,C)^2 - 2 * edge(B,C) * dr(A,C) * cos(\angle BCA)$:

$$\angle BCA = cos^{-1}\left(\frac{edge(B,C)^2 + dr(A,C)^2 - dr(A,B)^2}{2 * edge(B,C) * dr(A,C)}\right)$$

To determine the clockwise angle $\angle ACD$, we must either add or subtract $\angle BCA$ to/from $\angle BCD$, depending on the orientation of $\triangle BCA$. If $\triangle BCA$ is clockwise, we simply add the two. If $\triangle BCA$ is counter-clockwise, then the angles overlap and we must therefore take their difference. Put another way, we can just convert $\angle BCA$

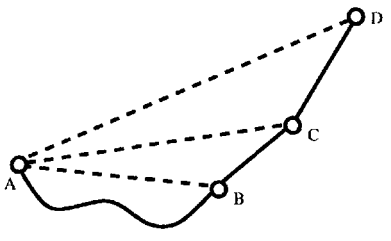


Figure 5: In the BFS path from vertex A to D, the predecessor of D is C and the predecessor of C is B.

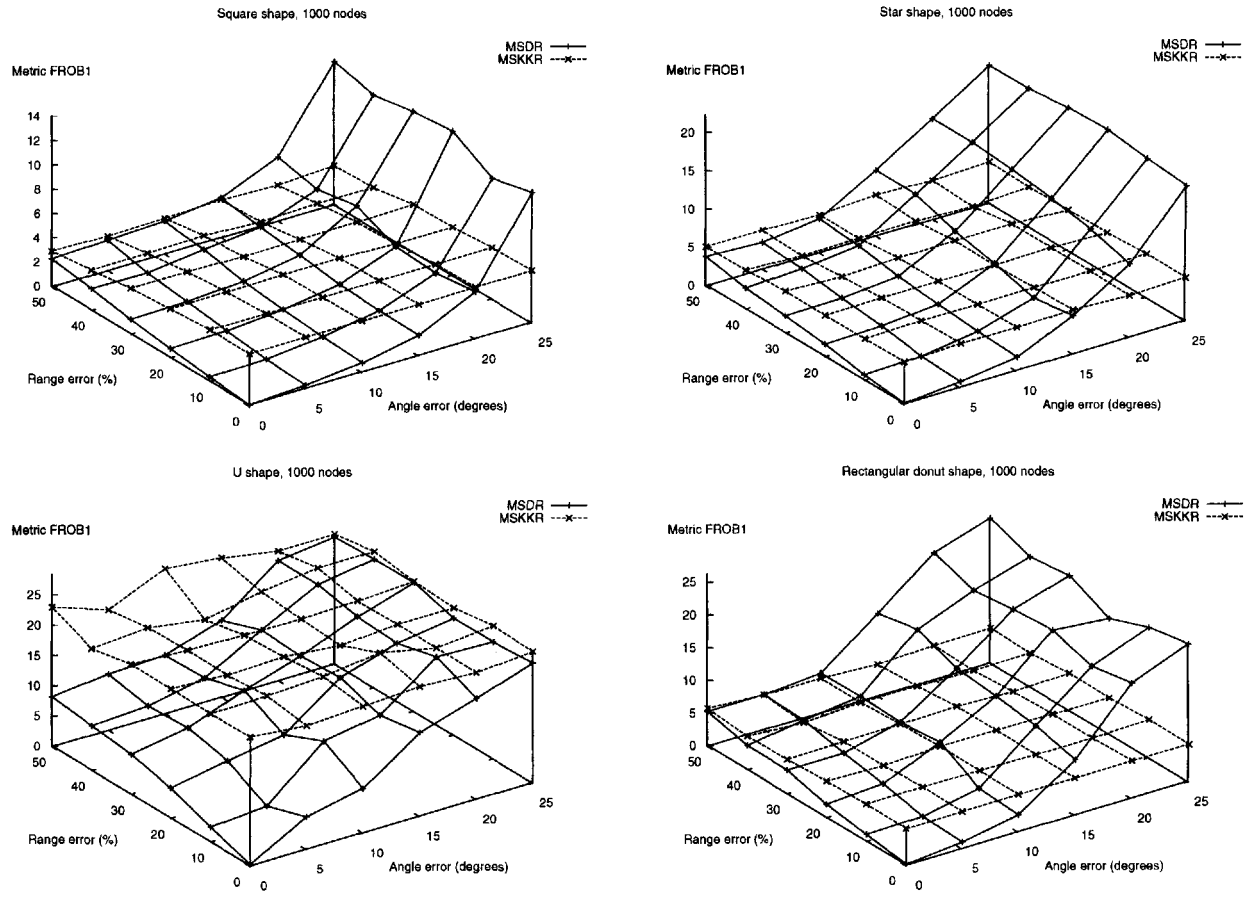


Figure 6: Comparison between MSKKR and MSDR measured by the Frobenius metric across square-shapes, star-shapes, U-shapes, and donut-shapes with 50 to 1000 vertices. There were twenty trials per shape, using graphs with density 8 and range errors 0-50% and angle error $0° - 25°$.

to a clockwise angle and add it to $\angle BCD$, then wrap it so that it is in the range $[0^o, 360^o)$.

Now we know the following useful information: $dr(A, C)$, $\angle ACD$, and $edge(C, D)$. Using the law of cosines again, we can compute the distance from A to D: $dr(A, D)^2 = dr(A, C)^2 + edge(C, D)^2 - 2 * dr(A, C) * edge(C, D) * cos(\angle ACD)$. Although $\angle ACD$ may be over 180^o, the law of cosines still yields the proper DR distance (the law of cosines yields the same result for the clockwise angle which is greater than 180^o and the counter-clockwise angle which is less than 180^o). After the DR distance has been computed, we save the orientation of $\triangle ACD$ (determined by whether or not $\angle ACD$ is greater than 180^o) so that we can reference it when calculating the DR distance to further nodes.

There are two base cases that must be considered separately. For nodes adjacent to the starting node, the edge length is the DR distance and no further computation is necessary. For nodes that are 2 edges away from the starting node, $\angle ACD$ is already known and does not need to be calculated. Therefore, only the final law of cosines used in our algorithm needs to be applied to find $dr(A, D)$.

4.2 MSDR Performance

Putting together the dead-reckoning idea with the multi-scale range-based Kamada-Kawai algorithm results in our MSDR algorithm. Not surprisingly, it outperforms all of the algorithms discussed earlier in the paper, given small angle errors; see Fig. 3.

Comparing MSKKR to MSDR shows that MSDR with angle errors of less than $10°$ consistently performs better; see Fig 6. Since MSKKR does not depend on angle errors and is resilient to range-errors it produces stable results for in most of the experiments, with the exception of the U-shape. MSDR's performance depends heavily on the angle errors and less on the range errors. For non-convex shapes such as the U-shape, MSDR offers significant advantages even with

50% range error and 25° angle error.

Layouts obtained with the MSDR algorithm using small angle and range errors often match near-perfectly the given source graphs; see Fig. 7.

The quality of the layouts under varying range and angular errors is captured in Figs. 8-9. Under the Frobenius metric, the algorithm seems stable for range errors of less than 30% and angular errors of less than 10°. As expected, the effect of angular errors is more pronounced; see Fig. 8. MSDR also captures the boundary of the underlying region very well. Experiments with the BAR metric also confirm that the MSDR is stable under range errors of up to 30%; see Fig. 9.

5 Conclusions and Future Work

We presented several adaptations of force-directed graph layout algorithms for the centralized, anchor-free sensor localization problem. We also presented a new approach that takes advantage of angular information, based on dead-reckoning and multi-scale techniques. Our results indicate that incorporating angular information can significantly improve the performance of force-directed sensor localization approaches. All of these algorithms as well as the simulation that generates the data have been implemented as a part of the Graphael [5] system.

The results presented in this paper are for centralized algorithms, whereas distributed algorithms for the sensor localization problem are more desirable. We plan to explore the possibility of developing practical distributed variants of the two multi-scale algorithms, MSKKR and MSDR.

References

[1] I. F. Akyildiz, S. Weilian, Y. Sankarasubramaniam, and E. E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[2] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–258, Feb. 1992.

[3] L. Doherty, K. Pister, and L. E. Ghaoui. Convex optimization methods for sensor node position estimation. In *Proceedings of the 20th IEEE Computer and Communications Societies (INFOCOM-01)*, pages 1655–1663, 2001.

[4] S. P. Fekete, A. Kröller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. In *ALGOSENSORS*, volume 3121 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2004.

[5] D. Forrester, S. G. Kobourov, A. Navabi, K. Wampler, and G. Yee. graphael: A system for generalized force-directed layouts. In *12th Symposium on Graph Drawing (GD)*, pages 454–466, 2004.

[6] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991.

[7] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. *Computational Geometry: Theory and Applications*, 29(1):3–18, 2004.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Press, Baltimore, MD, 1996.

[9] C. Gotsman and Y. Koren. Distributed graph layout for sensor networks. In *12th Symposium on Graph Drawing (GD)*, pages 273–284, 2004.

[10] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of Graph Algorithms and Applications*, 6:179–202, 2002.

[11] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[12] E. Krotkov, M. Hebert, and R. Simmons. Stereo perception and dead reckoning for a prototype lunar rover. *Autonomous Robots*, 2(4):313–331, 1995.

[13] M. Mauve, J. Widmer, and H. Hartenstein. A Survey on Position-Based Routing in Mobile Ad-Hoc Networks. pages 30–39, November 2001.

[14] D. Niculescu and B. Nath. Ad hoc positioning system (APS) using AOA. In *Proceedings of the 22 Conference of the IEEE Computer and Communications Societies (INFOCOM-03)*, pages 1734–1743, 2003.

[15] N. B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks. In *1st International Conference on Embedded Networked Sensor Systems (SenSys-03)*, pages 340–341, 2003. Also *TR #892, MIT LCS, 2003.*

[16] C. Savarese, J. Beutel, and J. Rabaey. Locationing in distributed ad-hoc wireless sensor networks. In *Proceedings of the 2001 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2037–2040, 2001.

[17] A. Savvides, C. Han, and M. Srivastava. Dynamic Fine-Grained localization in Ad-Hoc networks of sensors. In *Proceedings of the 7th Conference on Mobile Computing and Networking (MOBICOM-01)*, pages 166–179, 2001.

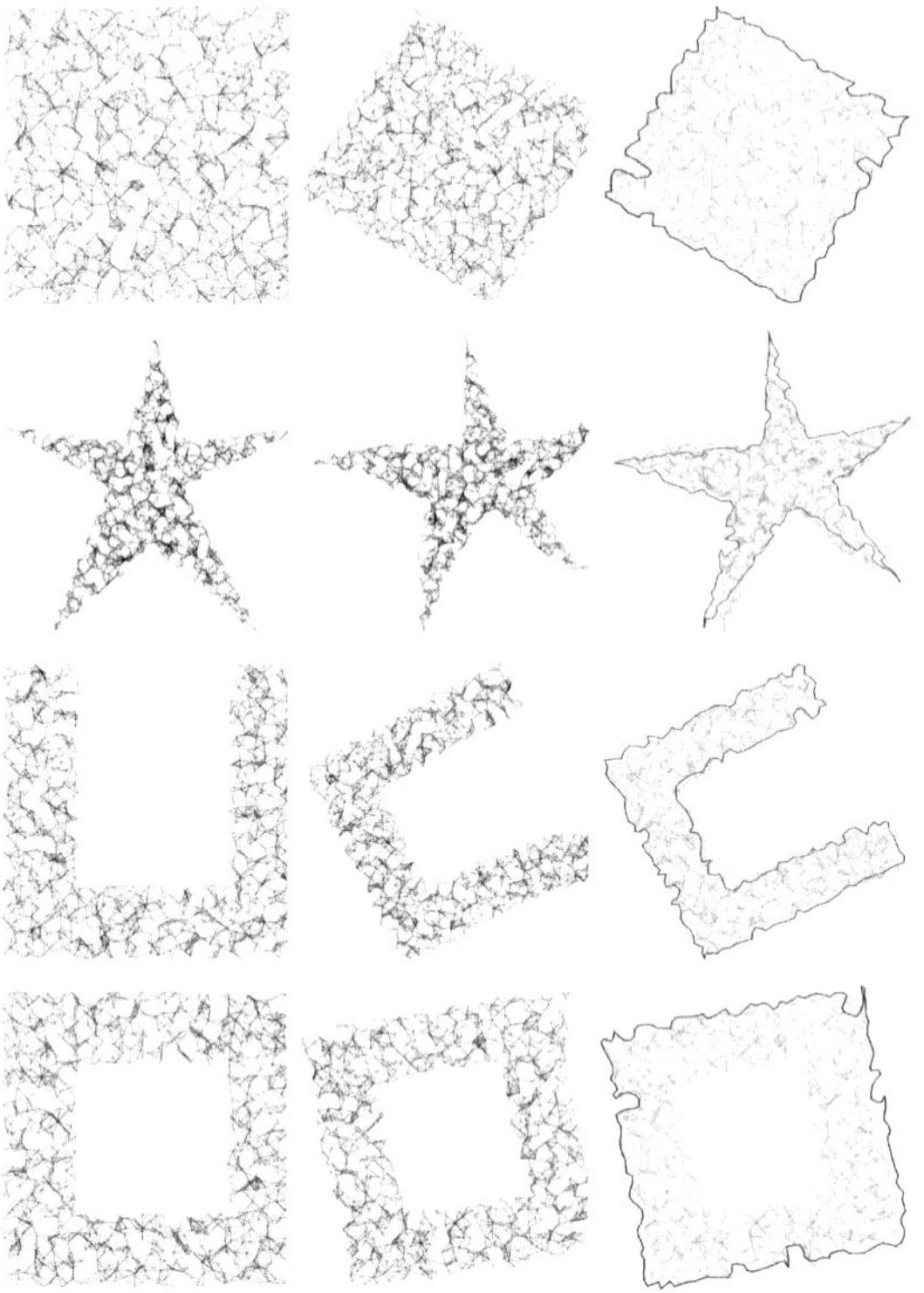


Figure 7: Typical results illustrating input/output/boundary-allignment for MSDR on square-shape, star-shape, U-shape, and donut-shape graphs. The underlying graphs have 1000 vertices, density 8, range error of 10% and angle error of 10°.

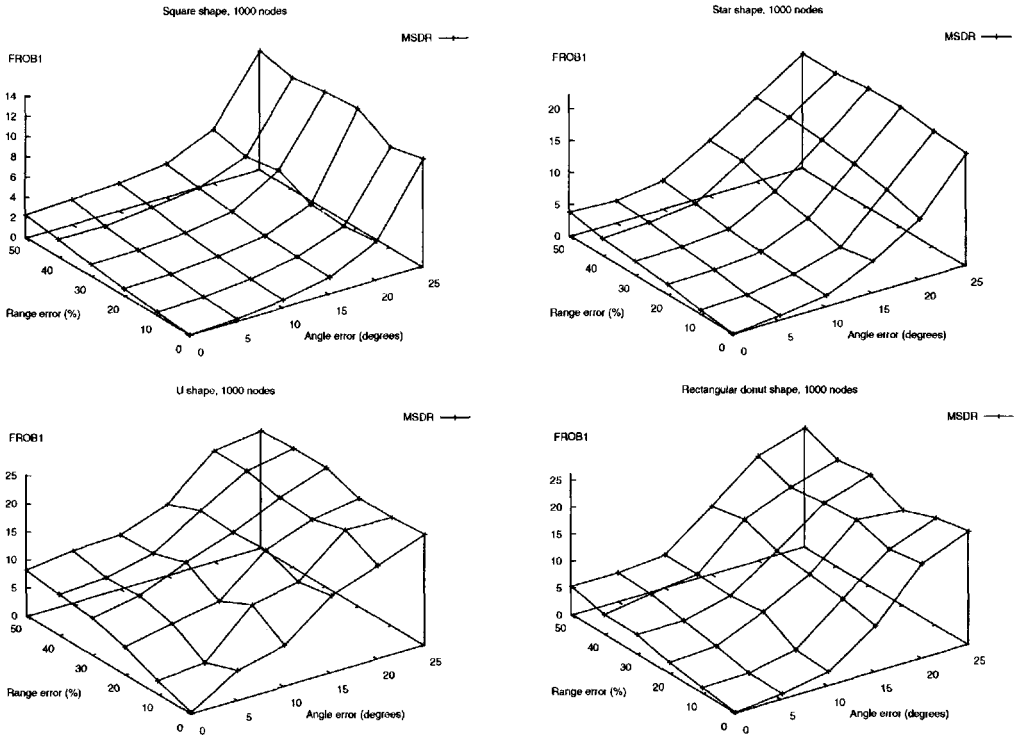


Figure 8: Frobenius metric error tolerance for MSDR across square-shape, star-shape, U-shape, and donut-shape graphs. There were twenty trials for each experiment using graphs with 1000 vertices, density 8 and varying the range and angle errors.

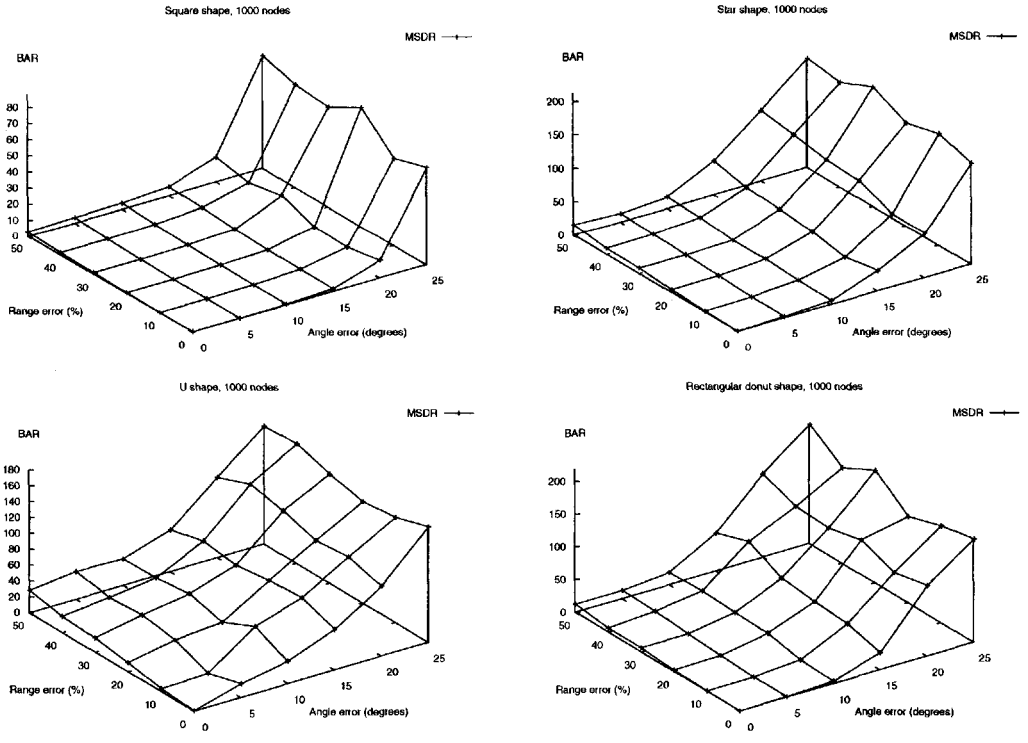


Figure 9: BAR metric error tolerance for MSDR across square-shape, star-shape, U-shape, and donut-shape graphs. There were twenty trials for each experiments using graphs with 1000 vertices, density 8 and varying the range and angle errors.

Compact Routing on Power Law Graphs with Additive Stretch*

Arthur Brady† Lenore Cowen†

Abstract

We present a universal routing scheme for unweighted, undirected networks that always routes a packet along a path whose length is at most an additive factor of d more than OPT (where OPT is the length of an optimal path), using $O(e \log^2 n)$-bit local routing tables and packet addresses, with d and e parameters of the network topology. For power-law random graphs, we demonstrate experimentally that d and e take on small values. The Thorup-Zwick universal multiplicative stretch 3 scheme has recently been suggested for routing on the Internet inter-AS graph; we argue, based on the results in this paper, that it is possible to improve worst-case performance on this graph by directly exploiting its power-law topology.

1 Introduction

Compact routing refers to the design of routing algorithms which store a small amount of information in a routing table at each node in a network, and provide a bound on the *stretch* of messaging routes. Following the terminology of Elkin for graph spanners [12], we say that a compact routing scheme has *stretch* (α, β) when the length of the route taken by a message from a source node u to a destination v is always at most $\alpha d(u,v) + \beta$, where $d(u,v)$ is the minimum length of a path from u to v in the network. A scheme with stretch $(\alpha, 0)$ is said to have *multiplicative stretch* α; one with stretch $(1, \beta)$ is said to have *additive stretch* β. If the worst-case size of a local routing table in a given scheme is $o(n)$, the scheme is said to be *compact*, and there is typically a tradeoff between minimizing multiplicative stretch and minimizing routing table size. Compact routing (and the closely-related problems of spanner construction and distance labeling) has been well-studied on special-case networks, such as trees [16, 27], graphs of bounded genus [19], and, recently, graphs whose doubling dimension [7] is bounded. There has also been much recent work on *universal* compact routing schemes [1, 10, 15, 27], which provide space bounds and multiplicative stretch guarantees for any undirected network. (As far as we know, the only work on compact routing in directed networks is work on roundtrip routing, appearing in [9, 26]; though additive stretch has been studied for graph spanners [3, 6], we are unaware of any prior compact routing results, even for special-case networks.)

A great deal of recent research has focused on discovering and modeling the topological properties of various large-scale real-world networks, including social networks and the Internet graph. In 1999, several teams [5, 20] reported independently that the degree distribution of the web graph appeared to follow a power-law. In a seminal paper of the same year, Faloutsos et al. [13] reported that the degrees of the Internet inter-AS graph also appeared to follow a power law. Since then, there has been extensive interest in random-graph models which capture this property. Two popular families of models have been proposed by Barabási and Albert [5] and Aiello, Chung and Lu [2]. Barabási and Albert proposed a model in which new vertices are added iteratively to a graph and linked to existing vertices with probability proportional to their degree; they showed that this construction induces a power-law degree distribution in the resulting graph. Aiello et al. proposed a power-law random graph model in which the degree sequence of an n-node graph is constructed according to the desired distribution, and demonstrate an elegant method of building a graph using the given degree sequence. In what follows, we will refer to graphs generated using the Barabási and Albert model as PC random graphs, and we will call graphs generated by the Aiello, Chung and Lu model PLRG graphs.

The minimum multiplicative stretch achieved by any known universal compact routing scheme is 3 [10, 27]. In the *name-independent* routing model (cf. [1, 4]), as well as in the case of *name-dependent* models where packet headers are restricted to be precisely $\log n$ bits long, a result of Gavoille and Gengler [18] shows that this is also a lower bound. Universal routing schemes only make their guarantees based on worst-case graph topologies; the question naturally arises as to the stretch *in practice* of these schemes on models of real-world networks, in particular on power-law networks. This question motivated the recent work of Krioukov et al. [22], who showed that on power-law random

*Supported in part by NSF grant CCR-0208629.

†Department of Computer Science, Tufts University, Medford, MA 02155. Email: {abrady,cowen}@cs.tufts.edu

graphs generated by the PLRG model, the average experimental performance of Thorup and Zwick's universal multiplicative stretch 3 routing algorithm [27] is in fact much better than three, and closer to 1.1 on certain power-law graphs. We consider the same problem they do (name-dependent compact routing on PLRG models) but show that designing schemes that exploit the expected structure of these networks can lead to better stretch on this class of networks.

While the observation that the degrees of the actual Internet inter-AS routing graph obey a power-law distribution is grounded in a lot of research (both theoretical and based on empirical measurement) [14, 24], work has also appeared recently that has convincingly cast doubt on the full sufficiency of this type of Internet model. Chen et al. [8] observe that the inferred power-law distribution may be (at least in part) due to artifacts of measurement, and Willinger et al. [29] conclude that descriptive models in their current form fail to address the underlying causes of the observed emergent properties of the network. Despite this, the idea of a power-law network remains an object of broad interest, making appearances in economic, social and biological models as well as in computer science.

The main theoretical result of this paper is a universal routing scheme on undirected, unweighted graphs with *additive* stretch $(1, d)$, using $O(e \log^2 n)$-bit local routing tables and message headers, where d and e are parameters of the network. We also describe a hybrid compact routing scheme which overlays our stretch $(1, d)$ scheme with the Thorup-Zwick universal stretch $(3, 0)$ scheme, with table size the same as in the latter (namely $O(\sqrt{n} \log^2 n)$). The hybrid scheme always routes along the best path provided by either scheme; its stretch is thus $\min\{(1, d), (3, 0)\}$.

Using the PLRG model, we verify experimentally that for power-law random graphs generated over a significant range of power-law parameters (which includes all values which have been estimated for the Internet graph), d and e take on small constant values, suggesting that our new routing scheme may work very well in practice.

Part of our contribution is the development of DIGG (DynamIc Graph Generator), a free C++-based software suite for the efficient generation and representation (in XML) of large graphs according to user-supplied parameters and generation algorithms. As the name suggests, the software also provides a way to create, store and analyze the life-cycles of *dynamic* graphs; this functionality was not used in the current paper, but we intend to demonstrate its utility in future work. The current (beta) version of the DIGG source code, the library of graphs which we generated for the experimental component of this paper, and the analytic code we created for our experiments are all freely available at **http://digg.cs.tufts.edu**.

2 Definitions

Consider a communications network modeled as a connected, undirected, unweighted graph $G = (V, E)$, $|V| = n$, with network nodes represented as vertices (each of which is assigned a unique label $v \in \{1, 2, \ldots, n\}$), and direct communications links represented as edges $uv \in E$.

A *routing scheme* R is a distributed algorithm defined on G which guarantees that any vertex u can send a message M to any specified destination v (along some (u, v)-path P in G), using metadata stored in M along with information stored locally at each vertex in P.

We refer to the metadata stored (by R) in a message M as M's *header*, and to the local information stored at a vertex v as v's *routing table*. Given an input graph $G = (V, E)$, a routing scheme R must specify:

1. the construction of the routing table at each vertex $v \in V$,
2. the construction of the header of any message M originating at a given source u and intended for a given destination v, and
3. a forwarding function $F(table(x), header(M))$ computed locally at each vertex $x \in V$ which, given the information in x's routing table and the information in M's header, selects an edge adjacent to x along which to forward M.

F is known as R's *routing function*. Given a source vertex u, a destination vertex v and a message M, the sequence of vertices $\langle u = v_0, v_1, \ldots, v_k \rangle$ defined by successive applications of $F(table(v_i), header(M))$ must be such that k is finite, and $v_k = v$. We refer to this (u, v)-path as the *route* P_{uv} from u to v with respect to R. Note the distinction between the *label* of a vertex v of G and the *header* of a message destined for v. Because the header of a message is forced to be succinct but is otherwise unconstrained, we are studying compact routing in the *name-dependent* routing model (also known as the *labeled* routing model). In contrast, routing models in which a message header destined for v is constrained to contain only the $(\log n)$-bit label of v are known as *name-independent* models (cf. [1, 4] for formal definitions and discussion). Our compact routing scheme requires neither a port-relabeling nor rewritable headers, so it is said to be a *1-phase* [11] routing scheme in the *fixed-port* [16] model.

As in Elkin [12], we say that a routing scheme R has *stretch* (α, β) with respect to a family of graphs G_f if the length of any route P_{uv} is always at most $\alpha d(u,v) + \beta$, where $d(u,v)$ is the minimum length of a path from u to v in the network.

A routing scheme is called *compact* if headers have size bounded by $O(\log^c n)$ bits for some constant c, and local routing tables each have size bounded by $o(n)$ bits. There is a natural tradeoff between header size and routing table size. There is also a natural tradeoff between the total size of all data structures employed by a routing scheme and its stretch; intuitively, if we are willing to take longer paths, we can use less information overall to get where we're going.

Recent experimental work of Krioukov et al. [22] looked at the performance of the best known universal multiplicative stretch-3 compact routing algorithm (due to Thorup and Zwick) on graphs generated by the PLRG model, and showed that the *average* stretch of routes on these graphs was much better than 3, and was closer to a multiplicative stretch of 1.1. The question we asked that led to this paper was: can you do better still on PLRGs?

2.1 Some facts about power-law graphs

DEFINITION 2.1. *A* power-law graph $G = (V, E)$ *is an undirected, unweighted graph whose degree distribution approximates a power law, i.e. the number* $y = |\{v \in V \mid deg(v) = x\}|$ *of vertices whose degree is* x *satisfies*

1. $\begin{cases} y = \lfloor c \rfloor - r & \text{when } x = 1 \\ y = \left\lfloor \frac{c}{x^\gamma} \right\rfloor & \text{when } x = 2,\ 3,\ \ldots,\ \left\lfloor c^{\frac{1}{\gamma}} \right\rfloor, \end{cases}$
2. $r = n - \sum_{x=1}^{\left\lfloor c^{\frac{1}{\gamma}} \right\rfloor} \left\lfloor \frac{c}{x^\gamma} \right\rfloor$, *and*
3. c *is a value minimizing* $|n - r|$

for some constant $\gamma \in \mathcal{R}^+$, *called the* power-law parameter *of* G.

DEFINITION 2.2. *A* γ-RPLG *(for "random power-law graph") is a graph* $G_\gamma = (V, E), |V| = n$, *which has been uniformly randomly selected from the set of all* n*-vertex power-law graphs with power-law parameter* γ.

Using a model very close to the PLRG model (but not exactly identical, see [2, 23] for discussion); Lincoln Lu [23], in his probabilistic analysis of power-law graph topology, observed that for sufficiently large values of n, with high probability[1] the following hold:

1. For all ranges of $\gamma > 0$, a random power-law graph G_γ has a unique giant component, and all components other than the giant component have size at most $O(\log n)$.

   (Throughout the following – since we consider a routing problem on connected networks, and since all non-giant components are small – we ignore all non-giant components, and abuse notation slightly by identifying G_γ with its giant component.)

2. For $0 < \gamma < 2$, G_γ contains an edge-dense "core" of diameter at most 3, which is connected to a set of "tree-like tails" of constant length.

3. For $2 < \gamma < 4$, G_γ contains

   - an inner layer consisting of a small edge-dense "core,"
   - an outer layer of "tree-like tails,"
   - and a "middle layer" in between the core and the outer layer.

   Furthermore, each of these three layers is of diameter $\Theta(\log n)$.

4. For all ranges of γ, the highest-degree vertices in G_γ are contained in the "core."

These facts suggest considering routing algorithms that employ different strategies in the core and in the tails, which is what we do in the next section.

3 Our new routing scheme

DEFINITION 3.1. *Let* G *be an undirected, unweighted graph, and let* h *be the node of* G *of highest degree (breaking ties lexographically by node names, so that* h *is always uniquely defined). For each positive even integer* d, *we define the* d-core *of* G *to be the subgraph of* G *induced by the set of vertices of distance at most* $d/2$ *from* h *(thus the* d*-core is of diameter* d*). We also define the* d-fringe *of* G *to be the subgraph of* G *induced by the set of vertices which are not in the* d*-core of* G.

DEFINITION 3.2. *Given a graph* H, *we define the* extra-edge count e_H *of* H *to be the minimum number of edges that must be removed from* H *in order to make* H *acyclic.*

With the parameters d and e defined, we are now ready to state our principal theorem:

THEOREM 3.1. *For any (unweighted, undirected) graph* $G = (V, E), |V| = n$, *there is a routing scheme* CFROUTE *that uses* $O(e \log^2 n)$*-bit headers and routing tables, and has a stretch of* $(1, d)$, *where* e *is the extra-edge count of the* d*-fringe of* G.

[1] When we say "X is true with high probability," we mean that the probability that X is false is $o(n^{-1})$.

Notice that it is easy to construct graph families for which either d or e must always be large, meaning that there exist graph families where this routing scheme isn't even compact. In this section, we prove this theorem. In the following section, we describe an experimental study of the tradeoffs between d and e for the PLRG model, and demonstrate ranges of γ for which we will expect to achieve small *additive* stretch using this routing scheme.

In order to present CFROUTE, we first need to slightly modify a known compact routing algorithm on trees, which we do in subsection 3.1. In subsection 3.2, we present and analyze CFROUTE as a proof of Theorem 3.1.

3.1 A compact routing algorithm for trees

There exist compact routing schemes for trees due both to Fraignaud and Gavoille [16] and to Thorup and Zwick [28] which use $O(\log^2 n)$-bit headers, $O(\log n)$-bit routing tables, and guarantee stretch $(1,0)$ (that is, they always route along optimal paths). Given a tree $T = (V, E)$, we denote by $TZ_T table[u]$ the local routing table assigned to vertex u by the Thorup-Zwick tree-routing scheme on T, and by $TZ_T header[w]$ the header assigned to vertex w by the Thorup-Zwick tree-routing scheme on T. Let $d_T(u,v)$ represent the length of the unique (u,v)-path in T.

Peleg [25] demonstrated that given any tree T with uniform edge weights, each vertex v of T can be assigned an $O(\log^2 n)$-bit label $l(v)$, such that given the labels of any two vertices v, w in T, the distance $d(v,w)$ between them can be computed exactly. Such a labeling scheme is referred to as an *(exact) distance labeling scheme;* we will refer to the scheme in [25] as the Peleg scheme. We denote by $l_T(v)$ the label generated by the Peleg scheme for a vertex v in a tree T.

We augment the Thorup-Zwick tree-routing scheme slightly to create a new tree-routing scheme TZ' as follows.

Let $T = (V, E)$ be any tree with root r.

1. Initially, store a routing table $TZ_T table[u]$ at each node u, and let $TZ_T header[w]$ be the header of node w, exactly as in the Thorup-Zwick tree-routing scheme.
2. For each vertex u, add the label $l_T(u)$ to $TZ_T table[u]$ to form $TZ'_T table[u]$.
3. For each vertex v, add the label $l_T(v)$ to $TZ_T header[v]$ to form $TZ'_T header[v]$.
4. Routing decisions are made exactly as prescribed by the Thorup-Zwick tree-routing scheme.

LEMMA 3.1. *TZ' uses $O(\log^2 n)$-bit headers, $O(\log^2 n)$-bit routing tables, and allows any vertex u, given the header of any destination v, to compute $d_T(u,v)$.*

Proof. The header and routing table size bounds are immediate from the fact that TZ' only requires $O(\log^2 n)$ more bits than the corresponding headers and tables in the original scheme. $d_T(u,v)$ can be computed from $l_T(u)$ and $l_T(v)$ exactly as described in [25]. □

3.2 Proof of Theorem 3.1 Given an unweighted, undirected graph $G = (V, E)$, $|V| = n$, and an even integer d, denote the d-core of G by I, and denote the d-fringe of G by F. Let h be the vertex of G of highest degree, with ties broken lexicographically. Let T be a single-source shortest path tree spanning G, with source h. Consider $T \cap F$. Extend $T \cap F$ by adding edges between vertices in F, until we have a spanning tree on each connected component of F. Call the resulting forest T_F. Let $E' = \{uv \mid uv \in E,\ uv \notin T_F,\ u, v \in F\}$ be set of all edges of G between vertices in F which are not contained in T_F. Let e_F denote the extra-edge count of F; note that because T_F spans each connected component of F, $|E'| \leq e_F$.

For any $u, v \in V$, let $d(u,v)$ denote the distance from u to v in G, let $d_T(u,v)$ represent the distance from u to v in T, and let $d_{T_F}(u,v)$ represent the distance from u to v in T_F, or ∞ if there is no (u,v)-path in T_F.

DEFINITION 3.3. *We define a set $\mathcal{T}$ of trees T_i to be the union of the following two sets:*

1. *a set of spanning trees $\{T_0 = T,\ T_1,\ \ldots,\ T_{|E'|}\}$ on G, constructed as follows:*
   - $i \longleftarrow 1$.
   - *For each edge $uv \in E'$,*
     - *Grow a single-source shortest path tree T_i rooted at u which includes uv.*
     - *Increment i.*
2. *the connected components of T_F, with each assigned an arbitrary root vertex.*

Note that each vertex of G is in at most one component of T_F. Throughout the following we refer to an element of $\mathcal{T}$ as T_i.

CFROUTE consists of four parts: a preprocessing step, in which we construct temporary data structures using TZ', a labeling step, in which nodes are assigned headers, a storage step, in which a local routing table is constructed at each node, and the routing procedure itself.

header[v] $\longleftarrow \emptyset$
For each $T_i \in \mathcal{T} \mid v \in T_i,\ 0 \leq i < |\mathcal{T}|$
header[v] $\longleftarrow$ header[v] $\circ(i,\ TZ'_{T_i}header[v])$

Figure 1: **The labeling step**

table[u] $\longleftarrow \emptyset$
For each $T_i \in \mathcal{T} \mid u \in T_i,\ 0 \leq i < |\mathcal{T}|$
table[u] $\longleftarrow$ table[u] $\circ(i,\ TZ'_{T_i}table[u])$

Figure 2: **The storage step**

3.2.1 Preprocessing Process each tree $T_i \in \mathcal{T}$ using TZ'. Let $TZ'_{T_i}table[u])$ be the routing table for node u in tree T_i, and let $TZ'_{T_i}header[v])$ be the header assigned to node u in tree T_i.

3.2.2 Labeling Assign to each vertex v a list of pairs $(i,\ TZ'_{T_i}header[v])$ (ordered by increasing i), one for every tree T_i which contains v.

3.2.3 Storage The routing table stored at each vertex u consists of a list of pairs $(i,\ TZ'_{T_i}table[u])$ (ordered by increasing i), one for every tree T_i which contains u.

LEMMA 3.2. CFROUTE *uses* $O(e_F \log^2 n)$*-bit headers and routing tables.*

Proof. Given any node $v \in G$, the number of trees $T_i \in \mathcal{T}$ containing v is at most $e_F + 2$:

- v is contained in T (because T spans G),
- v is contained in $|E'| \leq e_F$ spanning trees T_i in $\mathcal{T}$ by construction, and
- v is contained in at most 1 component of T_F.

So since the routing table at each node u contains at most $e_F + 2$ entries $(i,\ TZ'_{T_i}table[u])$, and since the header assigned to each node v contains at most $e_F + 2$ entries $(i,\ TZ'_{T_i}header[v])$, the result follows immediately from Lemma 3.1. □

3.2.4 Routing procedure Routing from a source vertex u to a destination v using CFROUTE proceeds as follows:

1. For each tree T_i containing both u and v, extract $l_{T_i}(u)$ from $TZ'_{T_i}table[u]$, extract $l_{T_i}(v)$ from $TZ'_{T_i}header[v]$, and compute $d_{T_i}(u,v)$ according to the Peleg scheme ([25]).
2. Choose some tree T_j such that $d_{T_j}(u,v)$ is minimized.
3. Route from u to v in T_j according to TZ'.

3.2.5 Analysis of the routing procedure

LEMMA 3.3. *Given any node u and any destination v,* CFROUTE *routes from u to v along a path of length at most $d(u,v) + d$.*

Proof. Let u and v be any two vertices which are both in I. Since $d_T(u,v) \leq d_T(u,h) + d_T(h,v) = d(u,h) + d(h,v) \leq \frac{d}{2} + \frac{d}{2} = d$, and since $1 \leq d(u,v)$, we have that $d_T(u,v) \leq d(u,v) + (d-1) < d(u,v) + d$.

Now let u and v be such that $u \in I$ and $v \notin I$. Since $u \in I$, $d(u,h) \leq \frac{d}{2}$. Since $d(h,v) \leq d(h,u) + d(u,v) \leq \frac{d}{2} + d(u,v)$, we have that $d_T(u,v) \leq d_T(u,h) + d_T(h,v) = d(u,h) + d(h,v) \leq \frac{d}{2} + [\frac{d}{2} + d(u,v)] = d(u,v) + d$.

Finally, let u and v be any two vertices both in F, and let P be any shortest (u,v)-path in G.

Either $d_{T_F}(u,v) = d(u,v)$, or $d_{T_F}(u,v) > d(u,v)$. If the latter is the case, then there exists some edge $u'v' \in P$ which is not in T_F. We consider two cases.

1. If there exists some such edge where $u', v' \in F$, then because $u'v' \notin T_F$, $u'v' \in E'$, so the preprocessing routine of CFROUTE constructed a single-source shortest path tree T_i spanning G with source u' (or v'; wlog assume u'). Since we have $d_{T_i}(u,v) \leq d_{T_i}(u,u') + d_{T_i}(u',v) = d(u,u') + d(u',v)$ and since u' is on some shortest (u,v)-path P in G, we conclude that $d_{T_i}(u,v) = d(u,v)$ for some T_i.
2. Now assume that $u' \in I$ or $v' \in I$ (or both) for all edges $u'v' \in P$ which are not in T_F.

   Notice that because P contains at least one vertex in I, we have that $d(u,I) + d(I,v) \leq |P| = d(u,v)$. So we conclude that $d_T(u,v) \leq d_T(u,h) + d_T(h,v) \leq [d(u,I) + \frac{d}{2}] + [\frac{d}{2} + d(I,v)] \leq d(u,v) + d$.

TZ' routes with stretch $(1,0)$ on each tree T_i. We have shown that for any two vertices $u, v \in V$, there is always some tree $T_i \in \mathcal{T}$ such that $d_{T_i}(u,v) \leq d(u,v) + d$. Given a source u and a destination v, since we always choose to route within a tree T_i minimizing $d_{T_i}(u,v)$, we have that any (u,v)-route in CFROUTE has length at most $d(u,v) + d$, giving a stretch of $(1,d)$ as desired. □

3.3 A hybrid scheme In practice, when faced with a network that may or may not be a PLRG, we remark that the right thing to do is to superimpose our scheme and Thorup and Zwick (TZ)'s *universal* stretch $(3,0)$ routing scheme, resulting in a hybrid scheme. (Note that the latter is a different scheme from the Thorup-Zwick tree-routing scheme discussed in Section 3.1). The key observation is that the universal TZ scheme can be modified so that a packet that arrives at any node u destined for some node v can, with help from the local routing table stored at u, compute exactly the length of the path from u to v that would be traversed using that scheme. (Our scheme already uses an analogous function.) The details are straightforward and are omitted from this extended abstract.

Thus we can simply concatenate headers for both schemes, and store tables for both schemes at every node. A packet then decides which scheme to follow by computing which one would result in a shorter path to its destination, then routing according to that scheme. The hybrid scheme is guaranteed to have better stretch than either the universal TZ scheme or our new scheme implemented separately: its stretch is $\min\{(1,d),(3,0)\}$. It also uses tables of the same asymptotic size as those used by the universal TZ scheme, namely $O(\sqrt{n}\log^2 n)$ bits.

4 Experiments

We have shown that for a given graph G, a given value of d, and I and F being the d-core and d-fringe of G respectively, CFROUTE guarantees an additive stretch of $(1,d)$ and uses $O(e_F \log^2 n)$-bit headers and routing tables. It remains to determine what values of d and e_F are typical for power-law graphs.

We conducted two sets of experiments: the first was an exploration of RPLG topology, and the second compared the performance of the Thorup-Zwick universal scheme, our scheme, and the hybrid scheme outlined in Section 3.3 on RPLGs.

Our topology experiments were designed to provide answers to the following:

- For a given γ-RPLG G_γ, can we find a small value of d such that the extra-edge count e_F of the d-fringe F of G_γ is also small?
- Lu [23]'s results on the properties of power-law graphs hold for sufficiently large n. We were interested in whether or not the desirable properties (such as a low-diameter core) were observed for the graph sizes we were interested in.

For the topology experiments, we used DIGG to implement the PLRG generator in [2], with which we generated random power-law graph instances on n vertices (for n = 2500, 5000, 10000, 20000 and 40000) for power-law parameter γ (for $1.2 \le \gamma \le 3.0$). We constructed 30 graph instances for each value of the parameter pair (n,γ). DIGG supports the fast saving and loading of graph instances to and from XML files; we were thus able both to create a permanent library of all the graph instances we generated, and to provide a basis for exact or parallel replication of our experiments by others.

For each generated graph instance G_γ, we calculated[2] the following:

1. d_{min}, the minimum value of d such that the d-fringe of G_γ was exactly a forest, and
2. for each $1 \le d \le d_{min}$,
   - the extra-edge count e of the d-fringe of G_γ.

We divided our topology experiments into two phases. In phase 1, we looked at graphs G_γ where $1.2 \le \gamma \le 1.9$; in phase 2, we examined G_γ for $2.0 \le \gamma \le 3.0$. According to the predictions in [23], we expected to find that graphs in the first range would actually have cores of *constant* diameter, and that graphs in the second range would have core of diameter $O(\log n)$. The extremely slow growth rates of the observed diameters for both types of graph this seem to confirm this prediction.

The table size of our algorithm is acceptable when e is not too large, and its stretch is smallest when $d = 2r$ is as small as possible. Define $\overline{e}$ to be the average value of the extra-edge count e of the d-fringe of G_γ (for given values of n, γ and d) across all graphs G_γ in the corresponding sample set; we denote by σ_e the estimated standard deviation of this statistic.

In phase 1 ($1.2 \le \gamma \le 1.9$), we found that for *all* observed values of n and γ, there was a sharp threshold: $\overline{e}$ was large when d was set to be 2 or 4, but setting $d = 6$ produced values of $\overline{e}$ and σ_e which were both less than 1. (In other words, for all values studied, the 6-fringe of G_γ differed on average from a forest by less than one edge.)

In phase 2 ($2.0 \le \gamma \le 3.0$), we found that for all observed values of n and for $2.0 \le \gamma \le 2.5$ (this being the range containing the vast majority of power-law parameter estimates for the various models of the

[2] Exceptions: we generated graphs for (n = 20000, γ = 1.2), but could not analyze them due to memory constraints. Also because of memory issues, for n = 40000, we only generated graphs for $2.0 \le \gamma \le 3.0$. As our data will show, however, the properties under consideration began to conform very closely to their predicted values in [23] well before n became this large.

Internet and web graphs), setting $d = 10$ produced values of $\overline{e}$ and σ_e which were both less than 5. (In other words, for these values, the 10-fringe of G_γ differed on average from a forest by less than 5 edges.)

Thus we conclude that for graph sizes up to 40,000, our scheme displays a worst-case additive stretch of $(1, 6)$ for phase 1 graphs, and $(1, 10)$ for phase 2 graphs, while maintaining $O(\log^2 n)$-bit tables. Compare to the Thorup-Zwick multiplicative-stretch scheme which uses $O(\sqrt{n} \log^2 n)$-bit tables and has worst-case stretch $(3, 0)$. Note that the theory of [23] implies that for phase 1 graphs, the additive stretch and table size of our scheme is unlikely to increase much as n grows beyond 40,000, whereas for phase 2 graphs, additive stretch and table size should increase logarithmically.

As was noted by Krioukov et al. in [22], the average stretch of the Thorup-Zwick scheme is considerably better than worst-case stretch on power-law graphs. In our routing experiments, simulations of the Thorup-Zwick scheme on our synthetic RLPGs yielded average-case (multiplicative) stretch between 1.25 and 1.18 (for γ between 2.0 and 2.2).[3] Simulations of the additive scheme presented in this paper produced an average multiplicative stretch between 1.22 and 1.11 (for γ in the same range), and the hybrid scheme outlined in Section 3.3 resulted in an average multiplicative stretch between 1.13 and 1.07. See Figures 9 - 11 for details.

The observed average stretch of our scheme was consistently better than the average stretch of the TZ scheme; also, the margin of improvement increased significantly as γ increased, which is intuitively due to the fact that the fringe becomes more sparse as the exponent of the power-law increases.

The hybrid scheme significantly outperformed both schemes (indicating that the sets of optimal routes discovered by each scheme were different from one another, so that when taken together, they provided a strong improvement over either scheme on its own).

We present two sets of figures summarizing two different perspectives on our topology results.

In Figures 3 and 4, n is fixed at a particular value in {10000, 20000}. Within each chart, $\overline{e}$ is plotted as a function of γ and d.

In each of figures 5 - 8, γ is fixed at a particular value in {2.0, 2.1, 2.2, 2.3}. (We emphasize the middle range here because it is in this range that power-law parameters for the Internet inter-AS topology have been estimated.) Within each chart, $\overline{e}$ is plotted as a function of n and d.

For our routing experiments, we simulated the Thorup-Zwick scheme, our scheme, and the hybrid scheme outlined in Section 3.3 on 15 of the 10,000-node graphs which we generated for the topology experiments: 5 graphs each of $\gamma \in \{2.0, 2.1, 2.2\}$.

The essential power of our algorithm lies in exploiting the sparsity of each RPLG, outside the core, using a small number of spanning trees. We discovered in our routing simulations that while a single spanning tree sourced at the highest-degree node provided an average stretch close to the worst-case bound, adding a very small number of spanning trees sourced at fringe edges caused the stretch to drop dramatically. We therefore added a heuristic to the simulations of our scheme and the hybrid scheme: if the extra-edge count of the fringe was less than 5, we added up to 5 trees spanning G_γ, sourced at random edges in the fringe.

Figures 9 - 11 describe our results: we measured the mean stretch over all possible paths (framed both multiplicatively and additively), as well as the percentage of optimal paths used by each algorithm. All data has been averaged over the 5 graphs in each set. More detailed data, including the full distribution of stretch over all paths for each algorithm on each graph, is available on our website.

The most commonly studied model of Internet routing is the *inter-AS graph*, which represents ASs (autonomous systems, which are roughly equivalent to ISPs) as vertices, and communications links (BGP peering links; cf. [17] for discussion) as edges. Recent empirical studies of the inter-AS graph [24] suggest that it has less than 20,000 nodes; its power-law parameter γ has been estimated to be between 2.0 and 2.5. Our choices of n and γ are therefore realistic for this type of model, and our algorithm remains quite compact (table size $O(\log^2 n)$) across observed values of n and γ, guaranteeing an additive stretch of $(1, 10)$.

We remark that a more granular representation of the Internet as a graph would model individual *routers* as vertices. There are approximately 250,000 Internet routers at present [21]. A model of this size would be more than 6 times larger than the graphs we studied in our experiments, but based on extrapolations of the observed growth curves of $\overline{e}$ for phase 2 graphs with γ between 2.0 and 2.5, we predict that when $d = 10$, both $\overline{e}$ and σ_e will remain less than 5 for graphs of this size. Thus for RPLGs of this size, the stretch and table size of our algorithm would remain $(1, 10)$ and $O(\log^2 n)$, respectively. Unfortunately, less is known about the topology of this type of Internet model; it may not even exhibit a power-law degree distribution.

While additive-stretch and multiplicative-stretch

[3] Note that [22] reports an average TZ stretch of 1.1 on RPLGs in this range, whereas we observed stretch closer to 1.2. The disparity is attributable to slight differences in the respective methods used to generate degree distributions for synthetic RPLGs.

schemes are hard to rank against each other, certainly we achieve a dramatic reduction in table size. If we have space resources which can accomodate $O(\sqrt{n}\log^2 n)$-bit tables, we can instead implement the hybrid scheme of Section 3.3, which is always guaranteed to do as well in stretch as the better of our additive and Thorup and Zwick's multiplicative stretch schemes. According to our simulations, the hybrid scheme appears to significantly outperform both schemes in practice.

We have made copies of our code for PLRG generation, topology analysis, routing simulations, and routing-scheme stretch analysis, as well as complete tabulations of all raw and aggregate analysis, and all generated graphs (encoded in XML) available at **http://digg.cs.tufts.edu**.

5 Discussion and future work

Taking into account the unique topological properties of power-law graphs has allowed us to design better compact routing schemes with superior performance on these graphs. We intend to investigate applications of this approach to other graph topologies.

Xenofontas Dimitropoulos and Dmitri Krioukov at CAIDA (**http://www.caida.org**) have recently provided us with graph data which incorporates estimates of inferred AS relationships in the real Internet inter-AS graph, and we intend to study new schemes which take advantage of their particular topological properties.

6 Acknowledgements

Thanks to Daniel Wolchonok and Patrick Schmid for major contributions to the development of DIGG. Kofi Laing and Alva Couch contributed helpful preliminary discussions concerning its high-level design. Dmitri Krioukov and k claffy of CAIDA (**http://www.caida.org**) continue to give critical insight about routing on "real" Internet graphs. Thanks to the anonymous referees for pointing out two errors in an early version of this manuscript. The authors gratefully acknowledge support from NSF grant CCR-0208629.

References

[1] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2004)*, pages 20–24. ACM Press, June 2004.

[2] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *STOC '00: Proceedings of the thirty-second annual ACM*

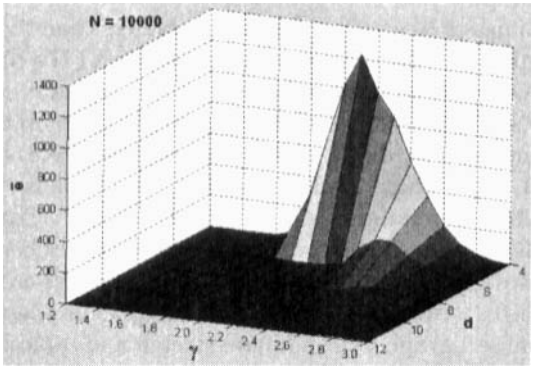


Figure 3: $N = 10000$. In this and the next figure, the mean extra-edge count $\overline{e}$ (over all graphs in each set) of the d-fringe of each graph is plotted as a function of the core diameter d and the power-law parameter γ.

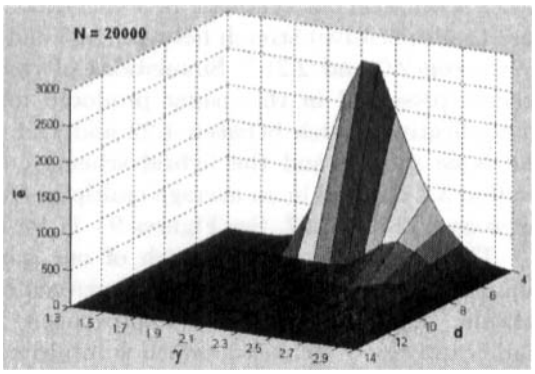


Figure 4: $N = 20000$.

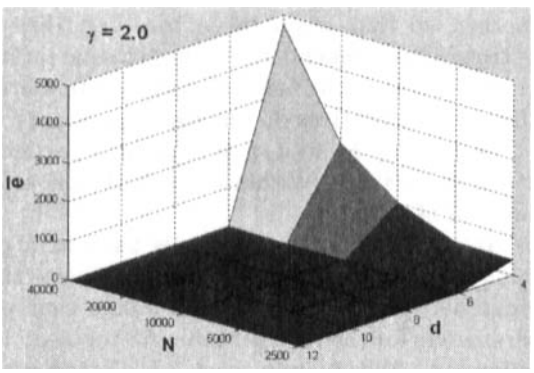


Figure 5: $\gamma = 2.0$. In this and the following three figures, the power-law parameter γ is held fixed, and the mean extra-edge count $\overline{e}$ of each set of graphs is plotted as a function of the graph size N and the core diameter d.

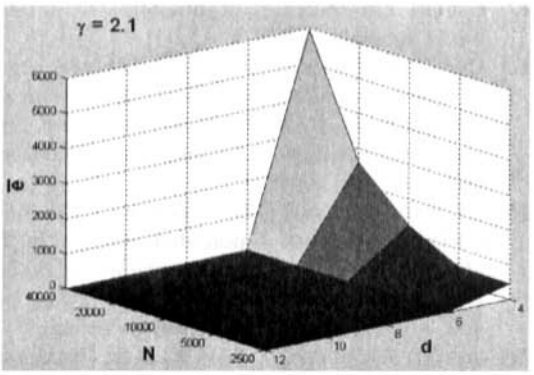


Figure 6: $\gamma = 2.1$.

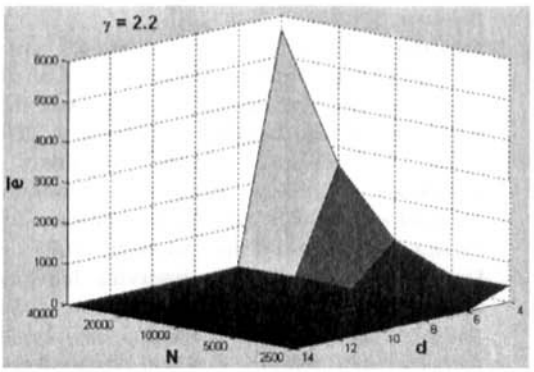


Figure 7: $\gamma = 2.2$.

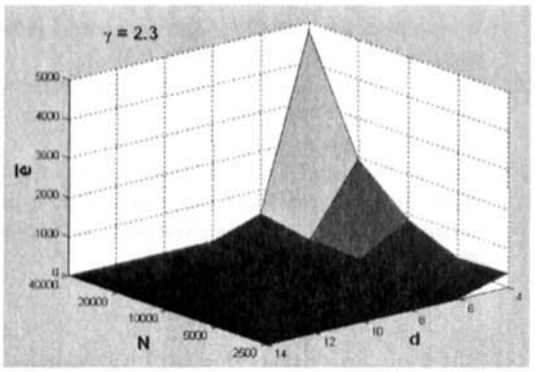


Figure 8: $\gamma = 2.3$.

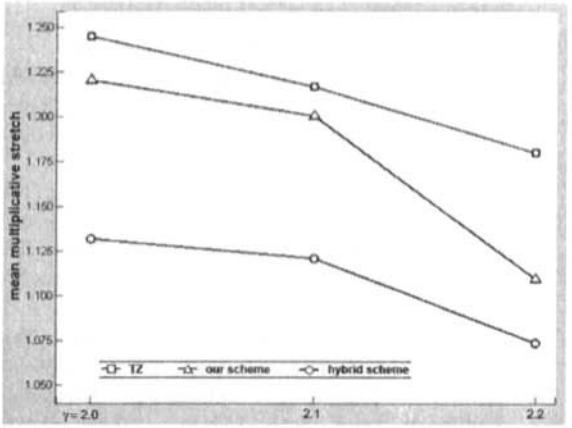


Figure 9: Observed average-case multiplicative stretch for the TZ universal scheme, the scheme presented in this paper, and the hybrid scheme mentioned in Section 3.3.

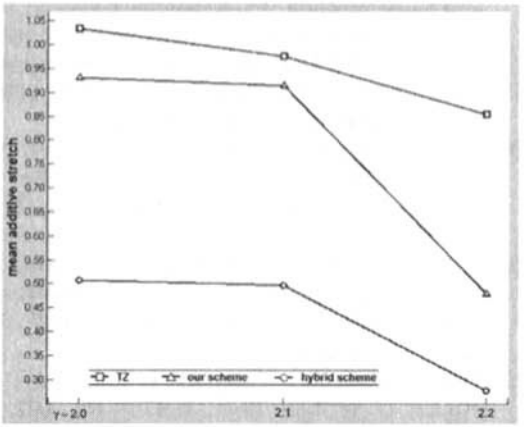


Figure 10: A comparison of average-case additive stretch for each of the three schemes.

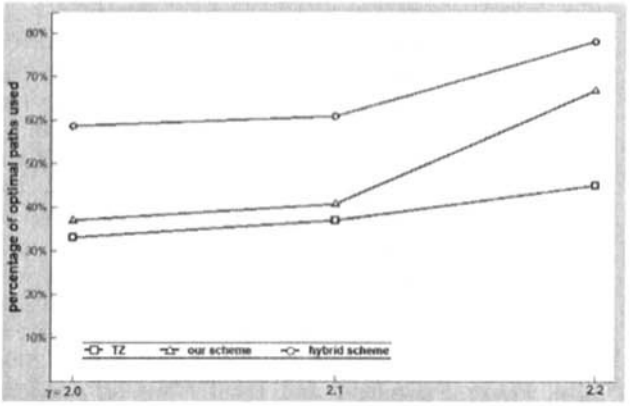


Figure 11: Percentage of optimal (i.e. shortest) paths detected by each of the three schemes in our routing simulations.

symposium on Theory of computing, pages 171–180, New York, NY, USA, 2000. ACM Press.

[3] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

[4] Marta Arias, Lenore Cowen, Kofi Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. In *Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 184–192, 2003.

[5] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[6] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β)-spanners and purely additive spanners. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 672–681, January 2005.

[7] Hubert T-H. Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[8] Qian Chen, Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Willinger. The origin of power laws in internet topologies revisited. In *Proc. of the IEEE INFOCOM*, 2002.

[9] Lenore J. Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. In *Proc. 19th ACM Symp. on Principles of Distrib. Computing*, pages 51–59, 2000.

[10] Lenore Cowen. Compact routing with minimum stretch. *J. of Algorithms*, 38:170–183, 2001.

[11] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. In *17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 11–20, 1998.

[12] Michael Elkin and David Peleg. $(1+\epsilon, \beta)$-spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004.

[13] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[14] Georgos Siganos, Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. Power laws and the as-level internet topology. *IEEE/ACM Trans. Netw.*, 11(4):514–524, 2003.

[15] Pierre Fraigniaud and Cyril Gavoille. Local memory requirement of universal routing schemes. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, August 1996.

[16] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, 2001.

[17] L. Gao. On inferring autonomous system relationships in the internet, 2000.

[18] Cyril Gavoille and Marc Gengler. Space-efficiency of routing schemes of stretch factor three. In *4th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 162–175, July 1997.

[19] Cyril Gavoille and Nicolas Hanusse. Compact routing tables for graphs of bounded genus. In *ICAL '99: Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, pages 351–360, London, UK, 1999. Springer-Verlag.

[20] Jon M. Kleinberg, S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: Measurements, models and methods. In *Proceedings of the International Conference on Combinatorics and Computing*, July 1999.

[21] Dmitri Krioukov. Private correspondence, 2005.

[22] Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. Compact routing on internet-like graphs. In *Proceedings of Infocom*, 2004.

[23] Linyuan Lu. The diameter of random massive graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 912–921. Society for Industrial and Applied Mathematics, 2001.

[24] Priya Mahadevan, Dmitri Krioukov, Marina Fomenkov, Bradley Huffaker, Xenofontas Dimitropoulos, kc claffy, and Amin Vahdat. Lessons from three views of the internet topology, 2005.

[25] David Peleg. Proximity-preserving labeling schemes and their applications. In *WG '99: Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 30–41, London, UK, 1999. Springer-Verlag.

[26] Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pages 844–851, January 2002.

[27] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10. ACM, July 2001.

[28] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 183–192, May 2001.

[29] Walter Willinger, Ramesh Govindan, Sugih Jamin, Vern Paxson, and Scott Shenker. Scaling phenomena in the internet: Critically examining criticality. *PNAS*, 99, 2002.

Reach for A^*: Efficient Point-to-Point Shortest Path Algorithms

Andrew V. Goldberg* Haim Kaplan† Renato F. Werneck‡

Abstract

We study the point-to-point shortest path problem in a setting where preprocessing is allowed. We improve the reach-based approach of Gutman [17] in several ways. In particular, we introduce a bidirectional version of the algorithm that uses implicit lower bounds and we add shortcut arcs to reduce vertex reaches. Our modifications greatly improve both preprocessing and query times. The resulting algorithm is as fast as the best previous method, due to Sanders and Schultes [28]. However, our algorithm is simpler and combines in a natural way with A^* search, which yields significantly better query times.

1 Introduction

We study the following *point-to-point shortest path problem* (P2P): given a directed graph $G = (V, A)$ with nonnegative arc lengths and two vertices, the source s and the destination t, find a shortest path from s to t. We are interested in exact shortest paths only. We allow preprocessing, but limit the size of the precomputed data to a (moderate) constant times the input graph size. Preprocessing time is limited by practical considerations. For example, in our motivating application, driving directions on large road networks, quadratic-time algorithms are impractical.

Finding shortest paths is a fundamental problem. The single-source problem with nonnegative arc lengths has been studied most extensively [1, 3, 4, 5, 9, 10, 11, 12, 15, 20, 25, 33, 37]. For this problem, near-optimal algorithms are known both in theory, with near-linear time bounds, and in practice, where running times are within a small constant factor of the breadth-first search time.

The P2P problem with no preprocessing has been addressed, for example, in [19, 27, 31, 38]. While no nontrivial theoretical results are known for the general P2P problem, there has been work on the special case of undirected planar graphs with slightly super-linear preprocessing space. The best bound in this context appears in [8]. Algorithms for approximate shortest paths that use preprocessing have been studied; see e.g. [2, 21, 34]. Previous work on exact algorithms with preprocessing includes those using geometric information [24, 36], hierarchical decomposition [28, 29, 30], the notion of reach [17], and A^* search combined with landmark distances [13, 16].

In this paper we focus on road networks. However, our algorithms do not use any domain-specific information, such as geographical coordinates, and therefore can be applied to any network. Their efficiency, however, needs to be verified experimentally for each particular application. In addition to road networks, we briefly discuss their performance on grid graphs.

We now discuss the most relevant recent developments in preprocessing-based algorithms for road networks. Such methods have two components: a *preprocessing algorithm* that computes auxiliary data and a *query algorithm* that computes an answer for a given s–t pair.

Gutman [17] defines the notion of vertex *reach*. Informally, the reach of a vertex is a number that is big if the vertex is in the middle of a long shortest path and small otherwise. Gutman shows how to prune an s-t search based on (upper bounds on) vertex reaches and (lower bounds on) vertex distances from s and to t. He uses Euclidean distances for lower bounds, and observes that the idea of reach can be combined with Euclidean-based A^* search to improve efficiency.

Goldberg and Harrelson [13] (see also [16]) have shown that the performance of A^* search (without reaches) can be significantly improved if landmark-based lower bounds are used instead of Euclidean bounds. This leads to the ALT (A^* search, landmarks, and triangle inequality) algorithm for the prob-

*Microsoft Research, 1065 La Avenida, Mountain View, CA 94062. E-mail: goldberg@microsoft.com; URL: http://www.research.microsoft.com/~goldberg/.

†School of Mathematical Sciences, Tel Aviv University, Israel. Part of this work was done while the author was visiting Microsoft Research. E-mail: haimk@math.tau.ac.il.

‡Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544. Supported by the Aladdin Project, NSF Grant no. CCR-9626862. Part of this work was done while the author was visiting Microsoft Research. E-mail: rwerneck@cs.princeton.edu.

lem. In [13], it was noted that the ALT method could be combined with reach pruning in a natural way. Not only would the improved lower bounds direct the search better, but they would also make reach pruning more effective.

Sanders and Schultes [28] (see also [29]) have recently introduced an interesting algorithm based on highway hierarchy; we call it the HH algorithm. They describe it for undirected graphs, and briefly discuss how to extend it to directed graphs. However, at the time our experiments have been completed and our technical report [14] published, there was no implementation of the directed version of the highway hierarchy algorithm. Assuming that the directed version of HH is not much slower than the undirected version, HH is the most practical of the previously published P2P algorithms for road networks. It has fast queries, relatively small memory overhead, and reasonable preprocessing complexity. Since the directed case is more general, if an algorithm for directed graphs performs well compared to HH then it follows that this algorithm performs well compared to the current state of the art. We compare our new algorithms to HH in Section 8.3.

The notions of reach and highway hierarchies have different motivations: The former is aimed at pruning the shortest path search, while the latter takes advantage of inherent road hierarchy to restrict the search to a smaller subgraph. However, as we shall see, the two approaches are related. Vertices pruned by reach have low reach values and as a result belong to a low level of the highway hierarchy.

In this paper we study the reach method and its relationship to the HH algorithm. We develop a shortest path algorithm based on improved reach pruning that is competitive with HH. Then we combine it with ALT to make queries even faster.

The first contribution of our work is the introduction of several variants of the reach algorithm, including bidirectional variants that do not need explicit lower bounds. We also introduce the idea of adding *shortcut arcs* to reduce vertex reaches. A small number of shortcuts (less than n, the number of vertices) drastically speeds up the preprocessing and the query of the reach-based method. The performance of the algorithm that implements these improvements (which we call RE) is similar to that of HH. We then show that the techniques behind RE and ALT can be combined in a natural way, leading to a new algorithm, REAL. On road networks, the time it takes for REAL to answer a query and the number of vertices it scans are much lower than those for RE and HH.

Furthermore, we suggest an interpretation of HH in terms of reach, which explains the similarities between the preprocessing algorithms of Gutman, HH, and RE. It also shows why HH cannot be combined with ALT as naturally as RE can.

In short, our results lead to a better understanding of several recent P2P algorithms, leading to simplification and improvement of the underlying techniques. This, in turn, leads to practical algorithms. For the graph of the road network of North America (which has almost 30 million vertices), finding the fastest route between two random points takes less than 4 milliseconds on a standard workstation, while scanning fewer than 2 000 vertices on average. Local queries are even faster.

Due to the page limit, we omit some details, proofs, and experimental results. A full version of the paper is available as a technical report [14].

2 Preliminaries

The input to the preprocessing stage of a P2P algorithm is a directed graph $G = (V, A)$ with n vertices and m arcs, and nonnegative lengths $\ell(a)$ for every arc a. The query stage also has as inputs a source s and a sink t. The goal is to find a shortest path from s to t. We denote by $\text{dist}(v, w)$ the shortest-path distance from vertex v to vertex w with respect to ℓ. In general, $\text{dist}(v, w) \neq \text{dist}(w, v)$.

The labeling method for the shortest path problem [22, 23] finds shortest paths from the source to all vertices in the graph. The method works as follows (see e.g. [32]). It maintains for every vertex v its distance label $d(v)$, parent $p(v)$, and status $S(v) \in \{\texttt{unreached}, \texttt{labeled}, \texttt{scanned}\}$. Initially $d(v) = \infty$, $p(v) = nil$, and $S(v) = \texttt{unreached}$ for every vertex v. The method starts by setting $d(s) = 0$ and $S(s) = \texttt{labeled}$. While there are labeled vertices, the method picks a labeled vertex v, *relaxes* all arcs out of v, and sets $S(v) = \texttt{scanned}$. To relax an arc (v, w), one checks if $d(w) > d(v) + \ell(v, w)$ and, if true, sets $d(w) = d(v) + \ell(v, w)$, $p(w) = v$, and $S(w) = \texttt{labeled}$.

If the length function is nonnegative, the labeling method terminates with correct shortest path distances and a shortest path tree. Its efficiency depends on the rule to choose a vertex to scan next. We say that $d(v)$ is *exact* if it is equal to the distance from s to v. Dijkstra [5] (and independently Dantzig [3]) observed that if ℓ is nonnegative and v is a labeled vertex with the smallest distance label, then $d(v)$ is exact and each vertex is scanned once. We refer to the labeling method with the minimum label selection rule as *Dijkstra's algorithm*. If ℓ is nonnegative then Dijkstra's algorithm scans vertices in nondecreasing order of distance from s and scans each vertex at most once.

For the P2P case, note that when the algorithm is about to scan the sink t, we know that $d(t)$ is exact and

the s-t path defined by the parent pointers is a shortest path. We can terminate the algorithm at this point. Intuitively, Dijkstra's algorithm searches a ball with s in the center and t on the boundary.

One can also run Dijkstra's algorithm on the *reverse graph* (the graph with every arc reversed) from the sink. The reverse of the t-s path found is a shortest s-t path in the original graph.

The *bidirectional algorithm* [3, 7, 26] alternates between running the forward and reverse versions of Dijkstra's algorithm, each maintaining its own set of distance labels. We denote by $d_f(v)$ the distance label of a vertex v maintained by the forward version of Dijkstra's algorithm, and by $d_r(v)$ the distance label of a vertex v maintained by the reverse version. (We will still use $d(v)$ when the direction would not matter or is clear from the context.) During initialization, the forward search scans s and the reverse search scans t. The algorithm also maintains the length of the shortest path seen so far, μ, and the corresponding path. Initially, $\mu = \infty$. When an arc (v, w) is relaxed by the forward search and w has already been scanned by the reverse search, we know the shortest s-v and w-t paths have lengths $d_f(v)$ and $d_r(w)$, respectively. If $\mu > d_f(v) + \ell(v, w) + d_r(w)$, we have found a path shorter than those seen before, so we update μ and its path accordingly. We perform similar updates during the reverse search. The algorithm terminates when the search in one direction selects a vertex already scanned in the other. A better criterion (see [16]) is to stop the algorithm when the sum of the minimum labels of labeled vertices for the forward and reverse searches is at least μ, the length of the shortest path seen so far. Intuitively, the bidirectional algorithm searches two touching balls centered at s and t.

Alternating between scanning a vertex by the forward search and scanning a vertex by the reverse search balances the number of scanned vertices between these searches. One can, however, coordinate the progress of the two searches in any other way and, as long as we stop according to one of the rules mentioned above, correctness is preserved. Balancing the work of the forward and reverse searches is a strategy guaranteed to be within a factor of two of the optimal strategy, which is the one that splits the work between the searches to minimize the total number of scanned vertices. Also note that remembering μ is necessary, since there is no guarantee that the shortest path will go through the vertex at which the algorithm stops.

3 Reach-Based Pruning

The following definition of reach is due to Gutman [17]. Given a path P from s to t and a vertex v on P, the *reach of v with respect to P* is the minimum of the length of the prefix of P (the subpath from s to v) and the length of the suffix of P (the subpath from v to t). The *reach* of v, $r(v)$, is the maximum, over all **shortest** paths P that contain v, of the reach of v with respect to P. (For now, assume that the shortest path between any two vertices is unique; Section 5 discusses this issue in more detail.)

Let $\overline{r}(v)$ be an upper bound on $r(v)$, and let $\underline{\text{dist}}(v, w)$ be a lower bound on $\text{dist}(v, w)$. The following fact allows the use of reaches to prune Dijkstra's search:

> Suppose $\overline{r}(v) < \underline{\text{dist}}(s, v)$ and $\overline{r}(v) < \underline{\text{dist}}(v, t)$. Then v is not on the shortest path from s to t, and therefore Dijkstra's algorithm does not need to label or scan v.

Note that this also holds for the bidirectional algorithm.

To compute reaches, it suffices to look at all shortest paths in the graph and apply the definition of reach to each vertex on each path. A more efficient algorithm is as follows. Initialize $r(v) = 0$ for all vertices v. For each vertex x, grow a complete shortest path tree T_x rooted at x. For every vertex v, determine its reach $r_x(v)$ within the tree, given by the minimum between its *depth* (the distance from the root) and its *height* (the distance to its farthest descendant). If $r_x(v) > r(v)$, update $r(v)$. This algorithm runs in $\tilde{O}(nm)$ time, which is still impractical for large graphs. On the largest one we tested, which has around 30 million vertices, this computation would take years on existing workstations.

Note that, if one runs this algorithm from only a few roots, one will obtain valid lower bounds for reaches. Unfortunately, the query algorithm needs good *upper* bounds to work correctly. Upper bounding algorithms are considerably more complex, as Section 5 will show.

4 Queries Using Upper Bounds on Reaches

In this section, we describe how to make the bidirectional Dijkstra's algorithm more efficient assuming we have upper bounds on the reaches of every vertex. As described in Section 3, to prune the search based on the reach of some vertex v, we need a lower bound on the distance from the source to v and a lower bound on the distance from v to the sink. We show how we can use lower bounds implicit in the search itself to do the pruning, thus obtaining a new algorithm.

During the bidirectional Dijkstra's algorithm, consider the search in the forward direction, and let γ be the smallest distance label of a labeled vertex in the reverse direction (i.e., the topmost label in the reverse heap). If a vertex v has not been scanned in the reverse direction, then γ is a lower bound on the distance from v to the destination t. (The same idea applies to

Figure 1: Bidirectional bound algorithm. Assume v is about to be scanned in the forward direction, has not yet been scanned in the reverse direction, and that the smallest distance label of a vertex not yet scanned in the reverse direction is γ. Then v can be pruned if $\bar{r}(v) < d_f(v)$ and $\bar{r}(v) < \gamma$.

the reverse search: we use the topmost label in the forward heap as a lower bound on the distance from s for unscanned vertices in the reverse direction.) When we are about to scan v we know that $d_f(v)$ is the distance from the source to v. So we can prune the search at v if all the following conditions hold: (1) v has not been scanned in the reverse direction, (2) $\bar{r}(v) < d_f(v)$, and (3) $\bar{r}(v) < \gamma$. When using these bounds, the stopping criterion is the same as for the standard bidirectional algorithm (without pruning). We call the resulting procedure the *bidirectional bound* algorithm. See Figure 1.

An alternative is to use the distance label of the vertex itself for pruning. Assume we are about to scan a vertex v in the forward direction (the procedure in the reverse direction is similar). If $\bar{r}(v) < d_f(v)$, we prune the vertex. Note that if the distance from v to t is at most $\bar{r}(v)$, the vertex will still be scanned in the reverse direction, given the appropriate stopping condition. More precisely, we stop the search in a given direction when either there are no labeled vertices or the minimum distance label of labeled vertices for the corresponding search is at least half the length of the shortest path seen so far. We call this the *self-bounding algorithm.*

The reason why the self-bounding algorithm can safely ignore the lower bound to the destination is that it leaves to the other search to visit vertices that are closer to it. Note, however, that when scanning an arc (v,w), even if we end up pruning w, we must check if w has been scanned in the opposite direction and, if so, check whether the candidate path using (v,w) is the shortest path seen so far.

The following natural algorithm falls into both of the above categories. The algorithm balances the radius of the forward and reverse search regions by picking the labeled vertex with minimum distance label considering both search directions. Note that the distance label of this vertex is also a lower bound on the distance to the target, as the search in the opposite direction has not selected the vertex yet. We refer to this algorithm as *distance-balanced.* Note that one could also use explicit lower bounds in combination with the implicit bounds.

We call our implementation of the bidirectional Dijkstra's algorithm with reach-based pruning RE. The query is distance-balanced and uses two optimizations: early pruning and arc sorting. The former avoids labeling unscanned vertices if reach and distance bounds justify this. The latter uses adjacency lists sorted in decreasing order by the reach of the head vertex, which allows some vertices to be early-pruned without explicitly looking at them. The resulting code is simple, with just a few tests added to the implementation of the bidirectional Dijkstra's algorithm.

5 Preprocessing

In this section we present an algorithm for efficiently computing upper bounds on vertex reaches. Our algorithm combines three main ideas, two introduced in [17], and the third implicit in [28].

The first idea is the use of *partial trees.* Instead of running a full shortest path computation from each vertex, which is expensive, we stop these computations early and use the resulting partial shortest path trees, which contain all shortest paths with length lower than a certain threshold. These trees allow us to divide vertices into two sets, those with small reaches and those with large reaches. We obtain upper bounds on the reaches of the former vertices. The second idea is to delete these low-reach vertices from the graph, replacing them by penalties used in the rest of the computation. Then we recursively bound reaches of the remaining vertices. The third idea is to introduce *shortcuts arcs* to reduce the reach of some vertices. This speeds up both the preprocessing (since the graph will shrink faster) and the queries (since more vertices will be pruned).

The preprocessing algorithm works in two phases: during the *main phase*, partial trees are grown and shortcuts are added; this is followed by the *refinement phase*, when high-reach vertices are re-evaluated in order to improve their reach bounds.

The main phase uses two subroutines: one adds shortcuts to the graph (*shortcut step*), and the other runs the partial-trees algorithm and eliminates low-reach vertices (*partial-trees step*). The main phase starts by applying the shortcut step. Then it proceeds in iterations, each associated with a threshold ϵ_i (which increases with i, the iteration number). Each iteration applies a partial-trees step followed by the shortcut step. By the end of the i-th iteration, the algorithm eliminates every vertex which it can prove has reach less than ϵ_i. If there are still vertices left in the graph, we set $\epsilon_{i+1} = \alpha\,\epsilon_i$ (for some $\alpha > 1$) and proceed to the next iteration.

Approximate reach algorithms, including ours, need the notion of a *canonical path*, which is a shortest path with additional properties. In particular, between every pair (s,t) there is a unique canonical path. We implement canonical paths as follows. For each arc a, we generate a length perturbation $\ell'(a)$. When computing the length of a path, we separately sum lengths and perturbations along the path, and use the perturbation to break ties in path lengths.

Next we briefly discuss the major components of the algorithm. Due to space limitations, we discuss a variant based on vertex reaches. We indeed use vertex reaches for pruning the query, but our best preprocessing algorithm uses arc reaches instead to gain efficiency (see [14] for details). The main ideas behind our arc-based preprocessing are the same as for the vertex-based version that we describe.

5.1 Growing Partial Trees. To gain intuition on the construction and use of partial trees, we consider a graph such that all shortest paths are unique (and therefore canonical) and a parameter ϵ. We outline an algorithm that partitions vertices into two groups, those with high reach (ϵ or more) and those with low reach (less than ϵ). For each vertex x in the graph, the algorithm runs Dijkstra's shortest path algorithm from x with an early termination condition. Let T be the current tentative shortest path tree maintained by the algorithm, and let T' be the subtree of T induced by the scanned vertices. Note that any path in T' is a shortest path. The tree construction stops when for every leaf y of T', one of two conditions holds: (1) y is a leaf of T or (2) the length of the x'-y path in T' is at least 2ϵ, where x' is the vertex adjacent to x on the x-y path in T'.

Let T_x, the *partial tree of* x, denote T' at the time the tree construction stops. The algorithm marks all vertices that have reach at least ϵ with respect to a path in T_x as high-reach vertices.

It is clear that the algorithm will never mark a vertex whose reach is less than ϵ, since its reach restricted to the partial trees cannot be greater than its actual reach. Therefore, to prove the correctness of the algorithm, it is enough to show that every vertex v with high reach is marked at the end. Consider a minimal canonical path P such that the reach of v with respect to P is high (at least ϵ). Let x and y be the first and the last vertices of P, respectively. Consider T_x. By uniqueness of shortest paths, either P is a path in T_x, or P contains a subpath of T_x that starts at x and ends at a leaf, z, of T_x. In the former case v is marked. For the latter case, note that z cannot be a leaf of T as z has been scanned and the shortest path P continues past z. The distance from x to v is at least ϵ and the distance from x', the successor of x on P, to v is less than ϵ (otherwise P would not be minimal). By the algorithm, the distance from x' to z is at least 2ϵ and therefore the distance from v to z is at least ϵ. Thus in this case v is also marked.

Note that long arcs pose an efficiency problem for this approach. For example, if x has an arc with length 100ϵ adjacent to it, the depth of T_x is at least 102ϵ. Building T_x will be expensive. All partial-tree-based preprocessing algorithms, including ours, deal with this problem by building smaller trees in such cases and potentially classifying some low-reach vertices as having high reach. This results in weaker upper bounds on reaches and potentially slower query times, but correctness is preserved.

Our algorithm builds the smaller trees as follows. Consider a partial shortest path tree T_x rooted at a vertex x, and let $v \neq x$ be a vertex in this tree. Let $f(v)$ be the vertex adjacent to x on the shortest path from x to v. The *inner circle* of T_x is the set containing the root x and all vertices $v \in T_x$ such that $d(v) - \ell(x, f(v)) \leq \epsilon$. We call vertices in the inner circle *inner vertices*; all other vertices in T_x are *outer vertices*. The *distance* from an outer vertex w to the inner circle is defined in the obvious way, as the length of the path (in T_x) between the closest (to w) inner vertex and w itself. The partial tree stops growing when all labeled vertices are outer vertices and have distance to the inner circle greater than ϵ.

Our preprocessing runs the partial-trees algorithm in iterations, multiplying the value of ϵ by a constant α, each time it starts a new iteration. Iteration i applies the partial-trees algorithm to a graph $G_i = (V_i, A_i)$. This is the graph induced by all arcs that have not been eliminated yet (considering not only the original arcs,

but also shortcuts added in previous iterations). All vertices in V_i have reach estimates above ϵ_{i-1} (for $i > 1$). To compute valid upper bounds for them, the partial-trees algorithm must take into account the vertices that have been deleted. It does so by using the concept of *penalties*, which implicitly increase the depths and heights of vertices in the partial trees. This ensures the algorithm will compute correct upper bounds.

Next we introduce arc reaches, which are similar to vertex reaches but carry more information and lead to faster preprocessing. They are useful for defining the penalties as well.

5.2 Arc Reaches. Let (v, w) be an arc on the shortest path P between s and t. The reach of this arc with respect to P is the minimum of the length of the prefix of P (the distance between s and w) and the length of the suffix of P (the distance between v and t). Note that the arc belongs to both the prefix and the suffix (a definition that excluded the arc from both would be equivalent). The arc reach of (v, w) with respect to the entire graph, denoted by $r(v, w)$, is the maximum reach of this arc with respect to all shortest paths P containing it.

During the partial-trees algorithm, we actually try to bound arc reaches instead of vertex reaches—the procedure is essentially the same as described before, and arc reaches are more powerful (the reach of an arc may be much smaller than the reaches of its endpoints). Once all arc reaches are bounded, they are converted into vertex reaches: a valid upper bound on the reach of a vertex can be obtained from upper bounds on the reaches of all incident arcs.

Penalties are computed as follows. The *in-penalty* of a vertex $v \in V_i$ is defined as

$$\textit{in-penalty}(v) = \max_{(u,v) \in A^+ : (u,v) \notin A_i} \{\bar{r}(u, v)\},$$

if v has at least one eliminated incoming arc, and zero otherwise. In this expression, A^+ is the set of original arcs augmented by the shortcuts added up to iteration i. The *out-penalty* of v is defined similarly, considering outgoing arcs instead of incoming arcs:

$$\textit{out-penalty}(v) = \max_{(v,w) \in A^+ : (v,w) \notin A_i} \{\bar{r}(v, w)\}.$$

If there is no outgoing arc, the out-penalty is zero.

The partial-trees algorithm works as described above, but increases the lengths of path suffixes and prefixes by out- and in-penalties, respectively, for the purpose of reach computation.

5.3 Shortcut Step. We call a vertex v *bypassable* if it has exactly two neighbors (u and w) and one of the following condition holds: (1) v has exactly one incoming arc, (u, v), and one outgoing arc, (v, w); or (2) v has exactly two outgoing arcs, (v, u) and (v, w), and exactly two incoming arcs, (u, v) and (w, v). In the first case, we say v is a candidate for a *one-way bypass*; in the second, v is a candidate for a *two-way bypass*. Shortcuts are used to go around bypassable vertices.

A *line* is a path in the graph containing at least three vertices such that all vertices, except the first and the last, are bypassable. Every bypassable vertex belongs to exactly one line, which can either be *one-way* or *two-way*. Once a line is identified, we may bypass it. The simplest approach would be to do it in a single step: if its first vertex is u and the last one is w, we can simply add a shortcut (u, w) (and (w, u), in case it is a two-way line). The length and the perturbation associated with the shortcut is the sum of the corresponding values of the arcs it bypasses. We break the tie thus created by making the shortcut *preferred* (i.e., implicitly shorter). If v is a bypassed vertex, any shortest path that passes through u and w will no longer contain v. This potentially reduces the reach of v. If the line has more than two arcs, we actually add "sub-lines" as well: we recursively process the left half, then the right half, and finally bypass the entire line. This reduces reaches even further, as the example in Figure 2 shows.

Once a vertex is bypassed, we immediately delete it from the graph to speed up the reach computation. As long as the appropriate penalties are assigned to its neighbors, the computation will still find valid upper bounds on all reaches.

One issue with the addition of shortcuts is that they may be very long, which can hurt the performance of the partial-trees algorithm in future iterations. To avoid this, we limit the length of shortcuts that may be added in iteration i to at most $\epsilon_{i+1}/2$.

5.4 The Refinement Phase. The fact that penalties are used to help compute valid upper bounds tends to make the upper bounds less tight (in absolute terms) as the algorithm progresses, since penalties become higher. Therefore, additive errors tend to be larger for vertices that remain in the graph after several iterations. Since they have high reach, they are visited by more queries than other vertices. If we could make these reaches more precise, the query would be able to prune more vertices. This is the goal of the *refinement phase* of our algorithm: it recomputes the reach estimates of the δ vertices with highest (upper bounds on) reaches found during the main step, where δ is a user-defined parameter (we used $\delta = \lceil 10\sqrt{n} \rceil$).

Let V_δ be this set of high-reach vertices of G. To

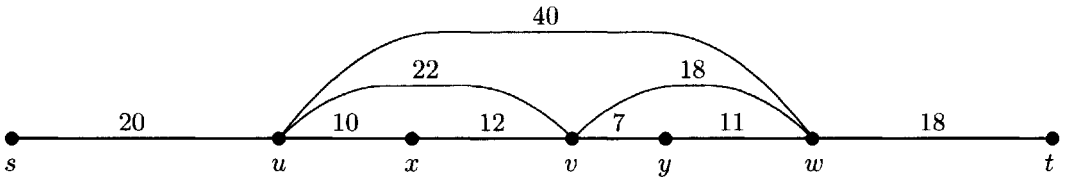


Figure 2: In this graph, (s,u), (u,x), (x,v), (v,y), (y,w), and (w,t) are the original edges (for simplicity, the graph is undirected). Without shortcuts, their reaches are $r(s) = 0$, $r(u) = 20$, $r(x) = 30$, $r(v) = 36$, $r(y) = 29$, $r(w) = 18$, and $r(t) = 0$. If we add just shortcut (u,w), the reaches of three vertices are reduced: $r(x) = 19$, $r(v) = 12$, and $r(y) = 19$. If we also add shortcuts (u,v) and (v,w), the reaches of x and y are reduced even further, to $r(x) = r(y) = 0$.

recompute the reaches, we first determine the subgraph $G_\delta = (V_\delta, A_\delta)$ induced by V_δ. This graph contains not only original arcs, but also the shortcuts between vertices in V_δ added during the main phase. We then run an exact vertex reach computation on G_δ by growing a complete shortest path tree from each vertex in V_δ. Because these shortest path trees include vertices in G_δ only, we still have to use penalties to account for the remaining vertices.

5.5 Additional Parameters. The choice of ϵ_1 and α is a tradeoff between preprocessing efficiency and the quality of reaches and shortcuts. To choose ϵ_1, we first pick $k = \min\{500, \lfloor\lceil\sqrt{n}\rceil/3\rfloor\}$ vertices at random. For each vertex, we compute the radius of a partial shortest path tree with exactly $\lfloor n/k \rfloor$ scanned vertices. (This radius is the distance label of the last scanned vertex.) Then we set ϵ_1 to be twice the minimum of all k radii. We use $\alpha = 3.0$ until we reach an iteration in the main phase where the number of vertices is smaller than δ, then we reduce it to 1.5. This change allows the algorithm to add more shortcuts in the final iterations. The refinement step ensures that the reach bounds of the last δ vertices are still good.

6 Reach and the ALT Algorithm

6.1 A^* Search and the ALT Algorithm. A *potential function* is a function from the vertices of a graph G to reals. Given a potential function π, the *reduced cost* of an arc is defined as $\ell_\pi(v,w) = \ell(v,w) - \pi(v) + \pi(w)$. Suppose we replace the original distance function ℓ by ℓ_π. Then for any two vertices x and y, the length of every x-y path (including the shortest) changes by the same amount, $\pi(y) - \pi(x)$. Thus the problem of finding shortest paths in G is equivalent to the problem of finding shortest paths in the transformed graph.

Now suppose we are interested in finding the shortest path from s to t. Let π_f be a (perhaps domain-specific) potential function such that $\pi_f(v)$ gives an estimate on the distance from v to t. In the context of this paper, A^* *search* [6, 18] is an algorithm that works like Dijkstra's algorithm, except that at each step it selects a labeled vertex v with the smallest *key*, defined as $k_f(v) = d_f(v) + \pi_f(v)$, to scan next. It is easy to see that A^* search is equivalent to Dijkstra's algorithm on the graph with length function ℓ_{π_f}. If π_f is such that ℓ_{π_f} is nonnegative for all arcs (i.e., if π_f is *feasible*), the algorithm will find the correct shortest paths. We refer to the class of A^* search algorithms that use a feasible function π_f with $\pi_f(t) = 0$ as *lower-bounding algorithms*. As shown in [16], better estimates lead to fewer vertices being scanned. In particular, a lower-bounding algorithm with a nonnegative potential function visits no more vertices than Dijkstra's algorithm, which uses the zero potential function.

We combine A^* search and bidirectional search as follows. Let π_f be the potential function used in the forward search and let π_r be the one used in the reverse search. Since the latter works in the reverse graph, each original arc (v,w) appears as (w,v), and its reduced cost w.r.t. π_r is $\ell_{\pi_r}(w,v) = \ell(v,w) - \pi_r(w) + \pi_r(v)$, where $\ell(v,w)$ is in the original graph. We say that π_f and π_r are *consistent* if, for all arcs (v,w), $\ell_{\pi_f}(v,w)$ in the original graph is equal to $\ell_{\pi_r}(w,v)$ in the reverse graph. This is equivalent to $\pi_f + \pi_r = \text{const}$.

If π_f and π_r are not consistent, the forward and reverse searches use different length functions. When the searches meet, we have no guarantee that the shortest path has been found. Assume π_f and π_r give lower bounds to the sink and from the source, respectively. We use the *average function* suggested by Ikeda et al. [19], defined as $p_f(v) = (\pi_f(v) - \pi_r(v))/2$ for the forward computation and as $p_r(v) = (\pi_r(v) - \pi_f(v))/2 = -p_f(v)$ for the reverse one. Although p_f and p_r usually do not give lower bounds as good as the original ones, they are feasible and consistent.

The ALT algorithm [13, 16] is based on A^* and uses landmarks and triangle inequality to compute feasible lower bounds. We select a small subset of vertices as *landmarks* and, for each vertex in the graph, precompute distances to and from every landmark.

Consider a landmark L: if $d(\cdot)$ is the distance *to* L, then, by the triangle inequality, $d(v) - d(w) \leq \text{dist}(v, w)$; if $d(\cdot)$ is the distance *from* L, $d(w) - d(v) \leq \text{dist}(v, w)$. To get the tightest lower bound, one can take the maximum of these bounds, over all landmarks. Intuitively, the best lower bound on $\text{dist}(v, w)$ is given by a landmark that appears "before" v or "after" w. We use the version of ALT algorithm used in [16], which balances the work of the forward search and the reverse search.

6.2 Reach and A^* search. Reach-based pruning can be easily combined with A^* search. Gutman [17] noticed this in the context of unidirectional search. The general approach is to run A^* search and prune vertices (or arcs) based on reach conditions. When A^* is about to scan a vertex v, we can extract the length of the shortest path from the source to v from the key of v (recall that $k_f(v) = d_f(v) + \pi_f(v)$). Furthermore, $\pi_f(v)$ is a lower bound on the distance from v to the destination. If the reach of v is smaller than both $d_f(v)$ and $\pi_f(v)$, we prune the search at v.

The reason why reach-based pruning works is that, although A^* search uses transformed lengths, the shortest paths remain invariant. This applies to bidirectional search as well. In this case, we use $d_f(v)$ and $\pi_f(v)$ to prune in the forward direction, and $d_r(v)$ and $\pi_r(v)$ to prune in the reverse direction. Pruning by reach does not affect the stopping condition of the algorithm. We still use the usual condition for A^* search, which is similar to that of the standard bidirectional Dijkstra, but with respect to reduced costs [16]. We call our implementation of the bidirectional A^* search algorithm with landmarks and reach-based pruning REAL. As in ALT, we used a version of REAL that balances the work of the forward search and the reverse search. Our implementation of REAL uses variants of early pruning and arc sorting, modified for the context of A^* search.

Note that we cannot use implicit bounds with A^* search. The implicit bound based on the radius of the ball searched in the opposite direction does not apply because the ball is in the transformed space. The self-bounding algorithm cannot be combined with A^* search in a useful way, because it assumes that the two searches will process balls of radius equal to half of the s-t distance. This defeats the purpose of A^* search, which aims at processing a smaller set.

The main gain in the performance of A^* search comes from the fact that it directs the two searches towards their goals, reducing the search space. Reach-based pruning sparsifies search regions, and this sparsification is effective for regions searched by both Dijkstra's algorithm and A^* search.

Note that REAL has two preprocessing algorithms: the one used by RE (which computes shortcuts and reaches) and the one used by ALT (which chooses landmarks and computes distances from all vertices to it). These two procedures are independent from each other: since shortcuts do not change distances, landmarks can be generated regardless of what shortcuts are added. Furthermore, the query is still independent of the preprocessing algorithm: the query only takes as input the graph with shortcuts, the reach values, and the distances to and from landmarks. The actual algorithms used to obtain this data can be changed at will.

7 Other Reach Definitions and Related Work

7.1 Gutman's Algorithm. In [17], Gutman computes shortest routes with respect to travel times. However, his algorithm, which is unidirectional, uses Euclidean bounds on travel *distances*, not times. This requires a more general definition of reach, which involves, in addition to the metric induced by graph distances (*native metric*), another metric M, which can be different. To define reach, one considers native shortest paths, but takes subpath lengths and computes reach values for M-distances. It is easy to see how these reaches can be used for pruning. Note that Gutman's algorithm can benefit from shortcuts, although he does not use them. All our algorithms have natural distance bounds for the native metric, so we use it as M.

Other major differences between RE and Gutman's algorithm are as follows. First, RE is bidirectional, and bidirectional shortest path algorithms tend to scan fewer vertices than unidirectional ones. Second, RE uses implicit lower bounds and thus does not need the vertex coordinates required by Gutman's algorithm. Finally, RE preprocessing creates shortcuts, which Gutman's algorithm does not. There are some other differences in the preprocessing algorithm, but their effect on performance is less significant. In particular, we do not grow partial trees from eliminated vertices, which requires a slightly different interpretation of penalties.

A variant of Gutman's algorithm uses A^* search with Euclidean lower bounds. In addition to the differences mentioned in the previous paragraph, REAL differs in using tighter landmark-based lower bounds.

7.2 Cardinality Reach and Highway Hierarchies. We now discuss the relationship between our reach-based algorithm (RE) and the HH algorithm of Sanders and Schultes. Since HH is described for undirected graphs, we restrict the discussion to this case.

We introduce a variant of reach that we call *c-reach* (cardinality reach). Given a vertex v on a shortest path P, grow equal-cardinality balls centered at its endpoints until v belongs to one of the balls. Let $c_P(v)$ be the

Table 1: Road Networks

NAME	DESCRIPTION	VERTICES	ARCS	LATITUDE (N)	LONGITUDE (W)
NA	North America	29 883 886	70 297 895	$[-\infty, +\infty]$	$[-\infty, +\infty]$
E	Eastern USA	4 256 990	10 088 732	[24.0; 50.0]	$[-\infty; 79.0]$
NW	Northwest USA	1 649 045	3 778 225	[42.0; 50.0]	[116.0; 126.0]
COL	Colorado	585 950	1 396 345	[37.0; 41.0]	[102.0; 109.0]
BAY	Bay Area	330 024	793 681	[37.0; 39.0]	[121; 123]

cardinality of each of the balls at this point. The c-reach of v, $c(v)$, is the maximum, over all shortest paths P, of $c_P(v)$. Note that if we replace cardinality with radius, we get the definition of reach. To use c-reach for pruning the search, we need the following values. For a vertex v and a nonnegative integer i, let $\rho(v, i)$ be the radius of the smallest ball centered at v that contains i vertices. Consider a search for the shortest path from s to t and a vertex v. We do not need to scan v if $\rho(s, c(v)) < \text{dist}(s, v)$ and $\rho(t, c(v)) < \text{dist}(v, t)$. Implementation of this pruning method would require maintaining $n - 1$ values of ρ for every vertex.

The main idea behind HH preprocessing is to use the partial-trees algorithm for c-reaches instead of reaches. Given a threshold h, the algorithm identifies vertices that have c-reach below h (local vertices). Consider a bidirectional search. During the search from s, once the search radius advances past $\rho(s, h)$, one can prune local vertices in this search. One can do similar pruning for the reverse search. This idea is applied recursively to the graph with low c-reach vertices deleted. This gives a hierarchy of vertices, in which each vertex needs to store a ρ-value for each level of the hierarchy it is present at. The preprocessing phase of HH also shortcuts lines and uses other heuristics to reduce the graph size at each iteration.

An important property of the HH query algorithm, which makes it similar to the self-bounding algorithm discussed in Section 4, is that the search in a given direction never goes to a lower level of the hierarchy. Our self-bounding algorithm can be seen as having a "continuous hierarchy" of reaches: once a search leaves a reach level, it never comes back to it. Like the self-bounding algorithm, HH cannot be combined with A^* search in a natural way.

8 Experimental Results

8.1 Experimental Setup. We implemented our algorithms in C++ and compiled them with Microsoft Visual C++ 7.0. All tests were performed on an AMD Opteron with 16 GB of RAM running Microsoft Windows Server 2003 at 2.4 GHz.

We use a standard cache-efficient graph representation. All arcs are stored in a single array, with each arc represented by its head and its length.[1] The array is sorted by arc tail, so all outgoing arcs from a vertex appear consecutively. An array of vertices maps the identifier of a vertex to the position (in the list of arcs) of the first element of its adjacency list. All query algorithms use standard four-way heaps.

We conduct most of our tests on road networks. We test our algorithm on the five graphs described in Table 1. The first graph in the table, North America (NA), was extracted from `Mappoint.NET` data and represents Canada, the United States (including Alaska), and the main roads of Mexico. The other four instances are representative subgraphs of NA (for tests on more subgraphs, see [14]). All graphs are directed and biconnected. We ran tests with two length functions: travel times and travel distances.

For a comparison with HH, we use the graph of the United States built by Sanders and Schultes [28] based on Tiger-Line data [35]. Because our implementations of ALT and REAL assume the graph to be connected (to simplify implementation), we only take the largest connected component of this graph, which contains more than 98.6% of the vertices. The graph is undirected, and we replace each edge $\{v, w\}$ by arcs (v, w) and (w, v). Our version of the graph (which we call USA) has 23 947 347 vertices and 57 708 624 arcs.

We also performed experiments with grid graphs. Vertices of an $x \times y$ grid graph correspond to points on a two-dimensional grid with coordinates i, j for $0 \leq i < x$ and $0 \leq j < y$. Each vertex has arcs to the vertices to its left, right, up, and down neighbors, if present. Arc lengths are integers chosen uniformly at random from $[1, 1024]$. We use square grids (i.e., $x = y$).

Unless otherwise noted, in each experiment we run the algorithms with a fixed set of parameters. For ALT we use the same parameters as in [16]: for each graph we generated one set of 16 *maxcover* landmarks, and each s-t search uses dynamic selection to pick between two and

[1] The length is stored as a 16-bit integer on the original graphs and as a 32-bit integer for the graphs with shortcuts. The head is always a 32-bit integer.

six of those. The same set of landmarks was also used by REAL. Upper bounds on reaches were generated with the algorithm described in Section 5. The reaches thus obtained (alongside with the corresponding shortcuts) were used by both RE and REAL.

8.2 Road Networks. Tables 2 and 3 present the results obtained by our algorithms when applied to the `Mappoint.NET` graphs with the travel-time and travel-distance metrics, respectively. In these experiments, we used 1000 random s-t pairs for each graph. We give results for preprocessing, average-case performance, and worst-case performance. For queries, we give both absolute numbers and the *speedup* with respect to an implementation of the bidirectional Dijkstra's algorithm (to which we refer as B).

For queries, the running time generally increases with graph size. While the complexity of ALT grows roughly linearly with the graph size, RE and REAL scale better. For small graphs, ALT is competitive with RE, but for large graphs the latter is more than 20 times faster. REAL is 3 to 4 times faster than RE.

In terms of preprocessing, we note that computing landmarks is significantly faster than finding good upper bounds on reaches. However, landmark data (with a reasonable number of landmarks) takes up more space than reach data; compare the space usage of RE and ALT. In fact, the reaches themselves are a minor part (less than 20%) of the total space required by RE. The rest of the space is used up by the graph with shortcuts (typically, the number of arcs increases by 35% to 55%) and by the shortcut translation map, used to convert shortcuts into its constituent arcs. The time for actually performing this conversion after each query is not taken into account in our experiments, since not all applications require it.

Next we compare the results for the two metrics. With the travel distance metric, the superiority of highways over local roads becomes much less pronounced than with travel times. As a result, RE become twice as slow for queries, and preprocessing takes 2.5 times longer on NA. On the other hand, ALT queries slow down only by about 20%, and preprocessing slows down even less. Changes in the performance of REAL fall in-between, which implies that its speedup with respect to RE becomes higher (on NA, REAL visits less than one tenth as many vertices as RE on average). All algorithms require a similar amount of space for travel times and for travel distances. While not quite as good as with travel times, the performance for travel distances is still excellent: REAL can find a shortest path on NA in less than 6 milliseconds on average.

While s and t are usually far apart on random s-t pairs, queries for driving directions tend to be more local. We used an idea from [28] to generate queries with different degrees of locality for NA. See Figure 3. When s and t are close together, ALT visits fewer vertices than RE. However, since the asymptotic performance of ALT is worse, RE quickly surpasses it as s and t get farther apart. REAL is the best algorithm in every case. Comparing plots for travel time and distance metrics, we note that ALT is less affected by the metric change than the other algorithms.

8.3 Comparison to Highway Hierarchies. As already mentioned, HH is the most practical of the previous P2P algorithms. Recall that HH works for undirected graphs only, while our algorithms work on directed graphs (which are more general). To compare our algorithms with HH, we use the (undirected) USA graph. Data for HH on USA, which we take from [28] and from a personal communication from Dominik Schultes, is available for the travel time metric only.

We compare both operation counts and running times. Since for all algorithms queries are based on the bidirectional Dijkstra's algorithm, comparing the number of vertices scanned is informative. For the running times, note that the HH experiments were conducted on a somewhat different machine. It was slightly slower than ours: an AMD Opteron running at 2.2 GHz (ours is an AMD Opteron running at 2.4 GHz) using the Linux operating system (ours uses Windows). Furthermore, implementation styles may be different. This introduces an extra error margin in the running time comparison. (To emphasize this, we use $\approx$ when stating running times for HH in Table 4.) However, comparing running times gives a good sanity check, and is necessary for preprocessing algorithms, which differ more than the query algorithms.

While in all our experiments we give the maximum number of vertices visited during the 1 000 queries we tested, Sanders and Schultes also obtain an upper bound on the worst-case number by running the search from each vertex in the graph to an unreachable dummy vertex and doubling the maximum number of vertices scanned. We did the same for RE. Note that this approach does not work for the landmark-based algorithms, as preprocessing would determine that no landmark is reachable to or from the dummy vertex. For both metrics, the upper bound is about a factor 1.5 higher than the lower bound given by the maximum over 1 000 trials, suggesting that the latter is a reasonable approximation of the worst-case behavior.

Data presented in Table 4 for the travel time metric suggests that RE and HH have similar performance and memory requirements. REAL queries are faster, but

Table 2: Algorithm performance on road networks with travel times as arc lengths: total preprocessing time, total space in disk required by the preprocessed data (in megabytes), average number of vertices scanned per query (over 1000 random queries), maximum number of vertices scanned (over the same queries), and average running times. Query data shown in both absolute values and as a speedup with respect to the bidirectional Dijkstra algorithm.

		PREP.	DISK	QUERY					
		TIME	SPACE	AVG SCANS		MAX SCANS		AVG TIME	
GRAPH	METHOD	(min)	(MB)	COUNT	SPD	COUNT	SPD	ms	SPD
BAY	ALT	0.7	26	4052	29	54818	5	3.39	16
	RE	3.2	19	1590	74	3438	85	1.17	48
	REAL	3.9	40	290	404	1691	172	0.45	123
COL	ALT	1.6	47	7373	26	85246	6	5.84	15
	RE	5.2	36	2181	88	5074	103	1.80	49
	REAL	6.9	73	306	624	1612	324	0.59	149
NW	ALT	3.9	132	14178	36	144082	8	12.52	21
	RE	17.5	100	2804	184	5877	203	2.39	112
	REAL	21.4	204	367	1408	1513	789	0.73	365
E	ALT	15.2	342	35044	42	487194	8	44.47	18
	RE	84.7	255	6925	212	13857	277	7.06	116
	REAL	99.9	523	795	1843	4543	844	1.61	510
NA	ALT	95.3	2398	250381	41	3584377	8	393.41	19
	RE	678.8	1844	14684	698	24618	1104	17.38	439
	REAL	774.2	3726	1595	6430	7450	3647	3.67	2080

Table 3: Algorithm performance on road networks with travel distances as arc lengths: total preprocessing time, total space in disk required by the preprocessed data (in megabytes), average number of vertices scanned per query (over 1000 random queries), maximum number of vertices scanned (over the same queries), and average running times. Query data shown in both absolute values and as a speedup with respect to the bidirectional Dijkstra algorithm.

		PREP.	DISK	QUERY					
		TIME	SPACE	AVG SCANS		MAX SCANS		AVG TIME	
GRAPH	METHOD	(min)	(MB)	COUNT	SPD	COUNT	SPD	ms	SPD
BAY	ALT	0.8	27	3383	35	42192	7	3.25	18
	RE	4.6	19	2761	43	6313	45	2.05	28
	REAL	5.4	41	335	356	2717	105	0.45	128
COL	ALT	1.8	48	7793	24	126755	4	6.34	14
	RE	9.7	36	3792	50	10067	50	3.16	28
	REAL	11.5	75	406	469	2805	178	0.72	123
NW	ALT	4.2	136	20662	26	426069	3	21.61	12
	RE	21.3	101	4217	125	10630	121	3.81	71
	REAL	25.4	208	478	1103	3058	419	0.89	302
E	ALT	14.6	353	43737	35	582663	7	61.98	15
	RE	158.9	258	14025	108	28144	141	13.28	69
	REAL	173.4	537	1142	1323	7097	560	2.27	404
NA	ALT	97.2	2511	292777	36	3588684	8	476.86	17
	RE	1623.0	1866	30962	336	56794	485	34.92	231
	REAL	1720.2	3860	2653	3922	17527	1570	5.97	1351

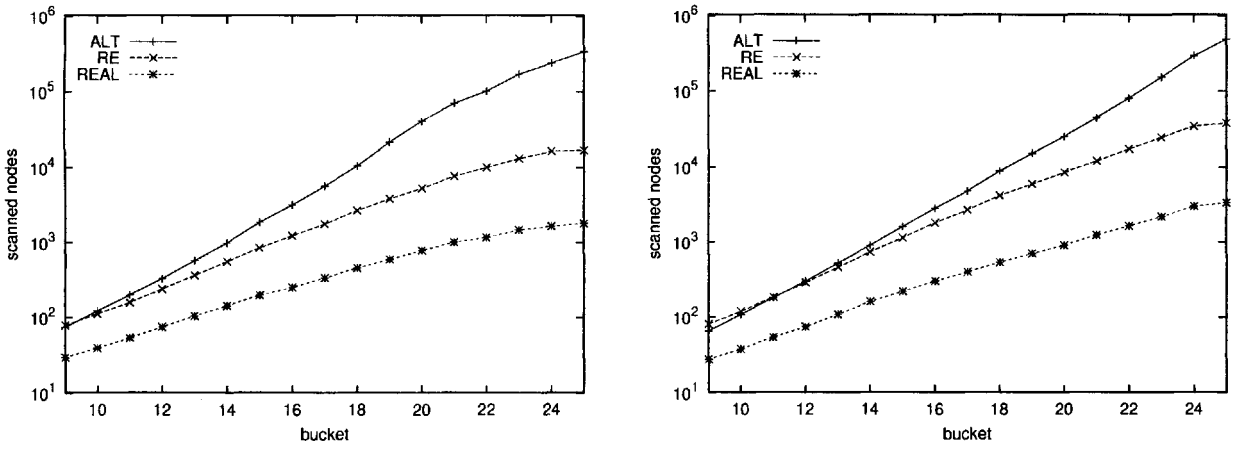


Figure 3: Average number of scanned vertices for local queries on NA with travel times (left) and distances (right). The horizontal axis refers to *buckets* with 1 000 pairs each. Each pair s-t in bucket i is such that s is chosen at random and t is the j-th farthest vertex from s, where j is selected uniformly at random from the range $(2^{i-1}, 2^i]$. The vertical axis is in log scale.

Table 4: Results for the undirected USA graph (same measures as in Table 2). For HH, averages are taken over 10 000 random queries (but the maximum is still taken over 1 000). For HH and RE we also give an upper bound on the maximum number of scans (UB). Data for HH with travel distances is not available.

		PREP.	DISK	QUERY						
		TIME	SPACE	AVG SCANS		MAX SCANS			AVG TIME	
METRIC	METHOD	(min)	(MB)	COUNT	SPD	COUNT	SPD	UB	ms	SPD
TIMES	ALT	92.7	1 984	177 028	44	2 587 562	8	—	322.78	21
	RE	365.9	1 476	3 851	2 000	8 722	2 330	13 364	4.50	1 475
	REAL	458.5	3 038	891	8 646	3 667	5 541	—	1.84	3 601
	HH	≈ 258.0	1 457	3 912	1 969	5 955	3 412	8 678	≈ 7.04	≈ 937
DISTANCES	ALT	99.9	1 959	256 507	33	2 674 150	8	—	392.84	15
	RE	981.5	1 503	22 377	376	44 130	500	68 672	25.59	236
	REAL	1 081.4	3 040	2 119	3 973	11 163	1 977	—	4.89	1 235

it needs more memory. ALT queries are substantially slower, but preprocessing is faster.

Lacking data for HH, we cannot compare it to our algorithms for the travel distance metric. Performance of our algorithms on USA with this metric is similar to that on NA with the same metric. This suggests that directed graphs are not much harder than undirected ones for our algorithms. In contrast, with the travel time metric, the performance of RE (and, to a lesser extent, REAL) is much better on USA than on NA. This suggests that the hierarchy on the USA graph with travel times is more evident than on NA, probably because USA has a small number of road categories.

8.4 Grids. Although road networks are our motivating application, we also tested our algorithms on grid graphs. As with road networks, for each graph we generated 1 000 pairs of vertices, each selected uniformly at random. These graphs have no natural hierarchy of shortest paths, which results in a large fraction of the vertices having high reach. For these tests, we used the same parameter settings as for road networks. It is unclear how much one can increase performance by tuning parameter values. As preprocessing for grids is fairly expensive, we limited the maximum grid size to about half a million vertices. The results are shown in Table 5.

As expected, RE does not get nearly as much speedup on grids as it does on road networks (see Tables 2 and 3). However, there is some speedup, and it does grow (albeit slowly) with grid size. ALT is significantly faster than RE: in fact, its speedup on grids is comparable to that on road networks. However, the speedup does not appear to change much with grid size, and it is likely that for very large grids RE would be faster.

An interesting observation is that REAL remains the

Table 5: Algorithm performance on grid graphs with random arc lengths. For each graph and each method, the table shows the total time spent in preprocessing, the total size of the data stored on disk after preprocessing, the average number of vertices scanned (over 1 000 random queries), the maximum number of vertices scanned (over the same queries), and the average running time. For the last three measures, we show both the actual value and the speedup (SPD) with respect to B.

		PREP.	DISK	QUERY					
		TIME	SPACE	AVG SCANS		MAX SCANS		AVG TIME	
VERTICES	METHOD	(min)	(MB)	COUNT	SPD	COUNT	SPD	msec	SPD
65 536	ALT	0.2	6.2	686	29.6	8 766	5.5	0.52	17.6
	RE	12.3	5.2	5 514	3.7	10 036	4.8	3.09	2.9
	REAL	12.5	9.6	363	55.9	2 630	18.4	0.34	26.4
131 044	ALT	0.6	12.4	1 307	32.6	14 400	7.2	1.42	13.9
	RE	44.7	10.4	9 369	4.6	16 247	6.4	5.94	3.3
	REAL	45.3	19.3	551	77.4	3 174	32.6	0.77	25.8
262 144	ALT	0.9	25.1	2 382	35.9	27 399	7.3	2.81	16.1
	RE	131.4	20.7	14 449	5.9	24 248	8.3	9.75	4.6
	REAL	132.3	38.8	791	108.0	5 020	39.9	1.22	37.1
524 176	ALT	1.9	50.2	4 416	38.8	40 568	9.9	5.25	17.5
	RE	232.1	41.4	23 201	7.4	39 433	10.2	17.47	5.3
	REAL	234.1	77.7	1 172	146.3	7 702	52.3	1.61	57.2

best algorithm in this test, and its speedup grows with grid size. For our largest grid, queries for REAL improve on ALT by about a factor of four for all performance measures that we considered. The space penalty of REAL with respect to ALT is a factor of about 1.5. REAL is over 50 times better than B. This shows that the combination of reaches and landmarks is more robust than either ALT or RE individually.

The most important downside of the reach-based approach on grids is its large preprocessing time. An interesting question is whether this can be improved. This would require a more elaborate procedure for adding shortcuts to a graph (instead of just waiting for lines to appear during the preprocessing algorithm). Such an improvement may lead to a better preprocessing algorithm for road networks as well.

8.5 Additional Experiments. We ran our preprocessing algorithm on BAY with and without shortcut generation. The results are shown in Table 6. Without shortcuts, queries visited almost 10 times as many vertices, and preprocessing was more than 15 times slower; for larger graphs, the relative performance is even worse. Without shortcuts, preprocessing NA is impractical. The table also compares approximate and exact reach computations. Again, preprocessing for exact reaches is extremely expensive, and of course shortcuts do not make it any faster (note that the shortcuts in this case are the ones added by the approximate algorithm). Fortunately, our upper bounding heuristic seems to do a good enough job: on BAY, exact reaches improved queries by less than 25%.

We also experimented with the number of landmarks REAL uses on NA. With as few as four landmarks, REAL is already twice as fast as RE on average (while visiting less than one third of the vertices). In general, more landmarks give better results, but with more than 16 landmarks the additional speedup does not seem to be worth the extra amount of space required.

9 Conclusion and Future Work

The reach-based shortest path approach leads to simple query algorithms with efficient implementations. Adding shortcuts greatly improves the performance of these algorithms on road networks. We have shown that the algorithm RE, based on these ideas, is competitive with the best previous method. Moreover, it combines naturally with A^* search. The resulting algorithm, REAL, improves query times even more: an average query in North America takes less than 4 milliseconds.

However, we believe there is still room for improvement. In particular, we could make the algorithm more cache-efficient by reordering the vertices so that those with high reach appear close to each other. There are few of those, and they are much more likely to be visited during any particular search than low-reach vertices.

The number of vertices visited could also be reduced. With shortcuts added, a shortest path on NA with travel times has on average less than 100 vertices,

Table 6: Results for RE with different reach values on BAY, both with and without shortcuts.

METRIC	SHORTCUTS	REACHES	PREP. TIME (min)	QUERY AVG SCANS	QUERY MAX SCANS	QUERY TIME (ms)
TIMES	NO	APPROX.	52.8	13 369	28 420	6.44
		EXACT	966.1	11 194	24 358	6.05
	YES	APPROX.	3.2	1 590	3 438	1.17
		EXACT	980.7	1 383	3 056	0.97
DISTANCES	NO	APPROX.	82.5	17 448	37 171	9.47
		EXACT	956.9	13 986	30 788	7.61
	YES	APPROX.	4.6	2 761	6 313	2.05
		EXACT	1 078.9	2 208	5 159	1.55

but an average REAL search scans more than 1500 vertices. Simply adding more landmarks would require too much space, however. To overcome this, one could store landmark distances only for a fraction (e.g., 20%) of the vertices, those with reach greater than some threshold R. The query algorithm would first search balls of radius R around s and t without using landmarks, then would start using landmarks from that point on. Another potential improvement would be to pick a set of landmarks specific to REAL (in our current implementation, REAL uses the same landmarks as ALT).

Also, one could reduce the space required to store $\overline{r}$ values by picking a constant γ, rounding $\overline{r}$'s up to the nearest integer power of γ, and storing the logarithms to the base γ of the $\overline{r}$'s.

Our query algorithm is independent of the preprocessing algorithm, allowing us to state natural subproblems for the latter. What is a good number of shortcuts to add? Where to add them? How to do it efficiently?

Another natural problem, originally raised by Gutman [17], is that of efficient reach computation. Can one compute reaches in less than $\Theta(nm)$ time? What about provably good upper bounds on reaches? Our results add another dimension to this direction of research by allowing shortcuts to be added to improve performance.

Another interesting direction of research is to identify a wider class of graphs for which these techniques work well, and to make the algorithms more robust over that class.

Acknowledgments

We would like to thank Peter Sanders and Dominik Schultes for their help with the USA graph data.

References

[1] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation. *Math. Prog.*, 73:129–174, 1996.

[2] L. J. Cowen and C. G. Wagner. Compact Roundtrip Routing in Directed Networks. In *Proc. Symp. on Principles of Distributed Computation*, pages 51–59, 2000.

[3] G. B. Dantzig. *Linear Programming and Extensions.* Princeton Univ. Press, Princeton, NJ, 1962.

[4] E. V. Denardo and B. L. Fox. Shortest-Route Methods: 1. Reaching, Pruning, and Buckets. *Oper. Res.*, 27:161–186, 1979.

[5] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1:269–271, 1959.

[6] J. Doran. An Approach to Automatic Problem-Solving. *Machine Intelligence*, 1:105–127, 1967.

[7] D. Dreyfus. An Appraisal of Some Shortest Path Algorithms. Technical Report RM-5433, Rand Corporation, Santa Monica, CA, 1967.

[8] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. In *Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science*, pages 232–241, 2001.

[9] M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. Assoc. Comput. Mach.*, 34:596–615, 1987.

[10] G. Gallo and S. Pallottino. Shortest Paths Algorithms. *Annals of Oper. Res.*, 13:3–79, 1988.

[11] A. V. Goldberg. A Simple Shortest Path Algorithm with Linear Average Time. In *Proc. 9th ESA, Lecture Notes in Computer Science LNCS 2161*, pages 230–241. Springer-Verlag, 2001.

[12] A. V. Goldberg. Shortest Path Algorithms: Engineering Aspects. In *Proc. ESAAC '01, Lecture Notes in Computer Science.* Springer-Verlag, 2001.

[13] A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2005.

[14] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Efficient Point-to-Point Shortest Path Al-

gorithms. Technical Report MSR-TR-2005-132, Microsoft Research, 2005.

[15] A. V. Goldberg and C. Silverstein. Implementations of Dijkstra's Algorithm Based on Multi-Level Buckets. In P. M. Pardalos, D. W. Hearn, and W. W. Hages, editors, *Lecture Notes in Economics and Mathematical Systems 450 (Refereed Proceedings)*, pages 292–327. Springer Verlag, 1997.

[16] A. V. Goldberg and R. F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In *Proc. 7th International Workshop on Algorithm Engineering and Experiments*, pages 26–40. SIAM, 2005.

[17] R. Gutman. Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *Proc. 6th International Workshop on Algorithm Engineering and Experiments*, pages 100–111. SIAM, 2004.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on System Science and Cybernetics*, SSC-4(2), 1968.

[19] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A Fast Algorithm for Finding Better Routes by AI Search Techniques. In *Proc. Vehicle Navigation and Information Systems Conference*. IEEE, 1994.

[20] R. Jacob, M. Marathe, and K. Nagel. A Computational Study of Routing Algorithms for Realistic Transportation Networks. *Oper. Res.*, 10:476–499, 1962.

[21] P. Klein. Preprocessing an Undirected Planar Network to Enable Fast Approximate Distance Queries. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 820–827, 2002.

[22] J. L. R. Ford. Network Flow Theory. Technical Report P-932, The Rand Corporation, 1956.

[23] J. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, Princeton, NJ, 1962.

[24] U. Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In *IfGIprints 22, Institut fuer Geoinformatik, Universitaet Muenster (ISBN 3-936616-22-1)*, pages 219–230, 2004.

[25] U. Meyer. Single-Source Shortest Paths on Arbitrary Directed Graphs in Linear Average Time. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 797–806, 2001.

[26] T. A. J. Nicholson. Finding the Shortest Route Between Two Points in a Network. *Computer J.*, 9:275–280, 1966.

[27] I. Pohl. Bi-directional Search. In *Machine Intelligence*, volume 6, pages 124–140. Edinburgh Univ. Press, Edinburgh, 1971.

[28] P. Sanders and D. Schultes. Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proc. 13th Annual European Symposium Algorithms*, volume 3669 of *LNCS*, pages 568–579. Springer, 2005.

[29] D. Schultes. Fast and Exact Shortest Path Queries Using Highway Hierarchies. Master's thesis, Department of Computer Science, Universitt des Saarlandes, Germany, 2005.

[30] F. Schulz, D. Wagner, and K. Weihe. Using Multi-Level Graphs for Timetable Information. In *Proc. 4th International Workshop on Algorithm Engineering and Experiments*, pages 43–59. LNCS, Springer, 2002.

[31] R. Sedgewick and J. Vitter. Shortest Paths in Euclidean Graphs. *Algorithmica*, 1:31–48, 1986.

[32] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[33] M. Thorup. Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time. *J. Assoc. Comput. Mach.*, 46:362–394, 1999.

[34] M. Thorup. Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. In *Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science*, pages 242–251, 2001.

[35] D. US Census Bureau, Washington. UA Census 2000 TIGER/Line files. http://www.census.gov/geo/www/tiger/tugerua/ua.-tgr2k.html, 2002.

[36] D. Wagner and T. Willhalm. Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In *European Symposium on Algorithms*, 2003.

[37] F. B. Zhan and C. E. Noon. Shortest Path Algorithms: An Evaluation using Real Road Networks. *Transp. Sci.*, 32:65–73, 1998.

[38] F. B. Zhan and C. E. Noon. A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths. *Journal of Geographic Information and Decision Analysis*, 4, 2000.

Distributed Routing in Small-World Networks

Oskar Sandberg *

December 26, 2005

Abstract

So called small-world networks – clustered networks with small diameters – are thought to be prevalent in nature, especially appearing in people's social interactions. Many models exist for this phenomenon, with some of the most recent explaining how it is possible to find short routes between nodes in such networks. Searching for such routes, however, always depends on nodes knowing what their and their neighbors positions are relative to the destination. In real applications where one may wish to search a small-world network, such as peer-to-peer computer networks, this cannot always be assumed to be true.

We propose and explore a method of routing that does not depend on such knowledge, and which can be implemented in a completely distributed way without any global elements. The Markov Chain Monte-Carlo based algorithm takes only a graph as input, and requires no further information about the nodes themselves. The proposed method is tested against simulated and real world data.

1 Introduction

The modern view of the so called "small-world phenomenon" can be dated back to the famous experiments by Stanley Milgram in the 1960s [15]. Milgram experimented with people's ability to find routes to a destination within the social network of the American population. He concluded that people were remarkably efficient at finding such routes, even towards a destination on the other side of the country. More recent studies using the Internet have come to the same conclusion, see [6].

Models to explain why graphs develop a small diameter ([19], [4], [17]), have been around for some times. Generally, these models specify the mixing of a structured base graph, such a as grid, and random "shortcuts" edges between nodes. However, it was not until Jon Kleinberg's work in 2000 [11] that a mathematical model was developed for how efficient routing can take place in such networks. Kleinberg showed that the possibility of efficient routing depends on a balance between the proportion of shortcut edges of different lengths with respect to coordinates in the base grid. Under a specific distribution, where the frequency of edges of different lengths decreases inverse proportionally to the length, simple greedy routing (always walking towards the destination) can find routes in $O(\log^2(n))$ steps on average, where n is the size of the graph.

1.1 Motivation Kleinberg's result is sharp in the sense that graphs where edges are chosen from a different distribution are shown not to allow for efficient searching. However, the small-world experiments seem to show that greedy-like routing is efficient in the world's social network. This indicates that some element of Kleinberg's model is present in the real world. In [12] and [18] this is motivated by reason of people's group memberships[1]. Several dynamic processes by which networks can evolve to achieve a similar edge distribution have also been proposed recently, for example, in [5], as well as in forthcoming work by this author [16].

However, in Kleinberg's search algorithm, the individual nodes are assumed to be aware of their own coordinates as well as those of their neighbors and the destination node. In the case of real world data, it may be difficult to identify what these coordinates are. In fact the participant nodes may be unaware of anything but their immediate neighborhood and thus oblivious of the global structure of the graph, and, importantly for this work, of geographic (or other) coordinates. For example, in peer-to-peer overlay networks on the Internet, one may wish to automatically find routes without relying on information about the local user,

*The Department of Mathematical Sciences, Chalmers Technical University and Gothenburg University. ossa@math.chalmers.se. I thank my advisors Olle Häggström and Devdatt Dubhashi for their assistance in reading drafts of this paper, as well as the anonymous referees for their helpful comments.

[1] Roughly: When a group is twice as large, people in it are half as likely to know each other.

let alone his neighbors or the routes target. In such a situation, how can we search for short paths from one node to another?

1.2 Contribution With this in mind, this paper attempts to return to Milgram's original problem of finding paths between people in social networks. Starting from an unmarked shortcut graph and no other information on the coordinates, we attempt to fit it against Kleinberg's model so as to make efficient searches possible. Taking as hypothesis that the graph was generated by applying Kleinberg's distribution model to a base graph with co-ordinate information, we attempt to recover the *embedding*. We approach this as a statistical estimation problem, with the configuration of positions in the grid assigned to each node as a (multi–dimensional) unknown parameter. With a good estimate for this embedding, it is possible to make greedy routing work without knowing the original positions of the nodes when the graph was generated. We employ a Markov Chain Monte-Carlo (MCMC) technique for fitting the positions.

We summarize our contributions as follows:

1. We give an MCMC algorithm to generate an embedding of a given graph into a one or two dimensional (toric) grid which is tuned to the distributions of Kleinberg's model.
2. This method is tested using artificially generated and controlled data: graphs generated according to the ideal model in one and two dimensions. The method is demonstrated to work quite well.
3. It is then applied to real social network data, taken from the "web of trust" of the users of an email cryptography program.
4. Finally, it is observed that the method used can be fully distributed, working only with local knowledge at each vertex. This suggests an application to routing in decentralized networks of peers that only connect directly to their own trusted friends in the network. Such networks, known as Friend-to-Friend networks of Darknets, have so far been limited to communication only in small cliques, and may become much more useful if global routing is made possible.
5. Our algorithm can thus be viewed also as a general purpose routing algorithm on arbitrary networks. It is tailored to "small world" networks, but appears to also work quite well for a more general class of graphs.

1.3 Previous Work Different methods of searching social networks and similar graphs have been discussed in previous work. In [3] a method is proposed for searching so called "power-law networks", either by a random walk or by targeting searches at nodes with high degree. Because such graphs have a highly skewed degree distribution, where a small set of nodes are connected to almost everyone, the methods are found to work well. The first author of that paper and a co-author recently investigated the problem of searching social networks in [2]. There they found that power-law methods did not work well, and instead attempted to use Kleinberg's model by trying to identify people's positions in some base graph based on their characteristics (where they live, work, etc). This was found to work well on a network with a canonical, highly structured base graph (employees of Hewlett Packard) but less well on the social network of students at Stanford University. Similarly Liben-Nowell et. al. [13] performed greedy searches using the town names as locations in the network of writers on the website "LiveJournal". They claim positive results, but consider searches successful when the same town as the desired target is reached: a considerably easier task than routing all the way.

In [20] the authors attempt to find methods to search a network of references between scientific authors. They mention Kleinberg's model, but state:

> "The topology of referral networks is similar to a two-dimensional lattice, but in our settings there is no global information about the position of the target, and hence it is not possible to determine whether a move is toward or away from the target".

It is the necessity of having such information that we attempt to overcome here.

2 Kleinberg's Model

Kleinberg's small-world model, like that of Watts and Strogatz [19] which preceded it, starts with a base graph of local connections, onto which a random graph of shortcut edges (long range contacts) is added. In its most basic form, one starts with a k-dimensional square lattice as the base network, and then adds q directed random edges at each node, selected so that each such shortcut edge from x points to y with probability:

$$\ell(x,y) = \frac{1}{d(x,y)^k H_k(n)}$$

where d denotes lattice distance in the base graph, n the size of the network, and H_k is a normalizing constant.

Kleinberg showed that in this case so-called *greedy routing* finds a path from any point to any other in, on average, $O(\log^2(n))$ steps. Greedy routing means always picking the neighbor (either through a shortcut or the base graph) which is closest to the destination, in terms of the lattice distance d, as the next step. Since routing within the base graph is permitted, the path strictly approaches the destination, and the same point cannot be visited twice.

In order to make the model more applicable to the real world, it is desirable to use the base graph only as a distance function between nodes, and thus only use the shortcut edges when routing. The necessity of a strictly approaching path existing then disappears, and we are left with the possibility of coming to a dead-end node which has no neighbor closer to the destination than itself. Kleinberg himself dealt with this issue in [12], working on non-geographical models, and there used q (node degree) equal to $\kappa \log^2(n)$ for a constant κ. In this case it is rather easy to see that κ can be chosen so as to make the probability that any node in the network is dead-end for a given query is arbitrarily small for all sizes n.

Actually, it suffices to keep the probability that a dead-end is encountered in any given route small. By approximate calculations one can see that this should hold if $q = \Theta(\log(n) \log\log(n))$[2]. In practice we find that scaling the number of links with $\log(n)$ preserves the number of paths that do not encounter a dead end for all Kleinberg model graphs we have simulated.

3 The Problem

The problem we are faced with here is this: given a network, presumed to be generated as the shortcuts in Kleinberg's model (in some number of dimensions), but without any information on the position of the nodes, can we find a good way to embed the network into a base grid so as to make the routing between them possible? This may be viewed as a parametric statistical estimation problem. The embedding is thus seen as the model's parameter, and the data set is a single realization of the model.

Seen from another perspective, we are attempting to find an algorithmic approach to answering the fundamental question of greedy routing: which of my neighbors is closest to the destination? In Kleinberg's model this is given, since each node has a prescribed position, but where graphs of this type occur in real life, that is not necessarily the case. The appeal of the approach described below is that we can attempt to answer the question using no data other than the graph of long connections itself, meaning that we use the clustering of the graph to answer the question of who belongs near whom.

Our approach is as follows: we assign positions to the nodes according to the a-posteriori distribution of the positions, given that the edges present had been assigned according to Kleinberg's model. Since long edges occur with a small probability in the model, this will tend to favor positions so that there are few long edges, and many short ones.

4 Statement

Let V be a set of nodes. Let ϕ be a function from V onto G, a finite (and possibly toric) square lattice in k dimensions[3]. ϕ is the configuration of positions assigned ! to the nodes in a base graph G. Let d denote graph distance in G. Thus for $x, y \in V$, $d(\phi(x), \phi(y))$ denotes the distance between respective positions in the lattice.

Let E denote a set of edges between points in V, and let them be numbered $1, \ldots, m$. If we assume that the edges are chosen according to the Kleinberg's model, with one end fixed to a particular node and the other chosen randomly, then the probability of a particular E depends on the distance its edges cover with respect to ϕ and G. In particular, if we let x_j and y_j denote the start and end point, respectively, of edge j, then:

$$\Pr(E|\phi) = \prod_{i=1}^{m} \frac{1}{d(\phi(x_i), \phi(y_i))^k H_G} \tag{4.1}$$

where H_G is a normalizing constant.

When seen as a function of ϕ, (4.1) is the likelihood function of a certain configuration having been used to generate the graph. The most straightforward manner in which to estimate ϕ from a given realization E is to choose the maximum likelihood estimate, that is the configuration $\hat{\phi}$ which maximizes (4.1). Clearly, this is the same as configuration which minimizes the product (or, equivalently, log sum) of the edge distances. Explicitly finding $\hat{\phi}$ is clearly a difficult problem: in one dimension it has been proven to be NP-complete [7], and there is little reason to believe that higher

[2] Roughly: The probability that a link will not be dead-end to a query decreases with $(\log n)^{-1}$. With $c \log(n) \log\log(n)$ links per node, the probability that a given node is a dead-end is thus bounded by $(\log n)^{\theta}$. θ can be made large by choosing a large c, thus making the probability of encountering a node in the $O(\log n)^2$ nodes encountered in a walk small.

[3] In our experiments below, we focus mostly on the one dimensional case, with some two dimensional results provided for comparisson purposes.

dimensions will be easier. There may be hope in turning to stochastic optimization techniques.

Another option, which we have chosen to explore here, is to use a Bayesian approach. If we see ϕ as a random quantity chosen with some probability distribution from the set of all possible such configurations (in other words, as a parameter in the Bayesian tradition), we can write:

$$\Pr(\phi|E) = \frac{\Pr(E|\phi)\Pr(\phi)}{\Pr(E)} \tag{4.2}$$

which is the a-posteriori distribution of the node positions, having observed a particular set of edges E. Instead of estimating the maximum likelihood configuration, we will try to assign configurations according to this distribution.

4.1 Metropolis-Hastings Algorithm The Metropolis-Hastings algorithm is a remarkable algorithm used in the field of Markov Chain Monte-Carlo. It allows one, given a certain distribution π on a set S, to construct a Markov chain on S with π as its stationary distribution. While simulating a known distribution might not seem extraordinary, Metropolis-Hastings has many properties that make it useful in broad range of applications.

The algorithm starts with a selection kernel $\alpha : S \times S \mapsto [0,1]$. This assigns, for every state s, a distribution $\alpha(s,r)$ of states which may be selected next. The next state, r, is selected according to this distribution, and then accepted with a probability $\beta(s,r)$ given by a certain formula of α and π. If the state is accepted, it becomes the next value of the chain, otherwise the chain stays in s for another time-step. If r is the proposed state, then the formula is given by:

$$\beta(s,r) = \min\left(1, \frac{\pi(r)\alpha(r,s)}{\pi(s)\alpha(s,r)}\right).$$

The Markov chain thus defined, with transition Matrix $P(s,r) = \alpha(s,r)\beta(s,r)$ if $s \neq r$ (and the appropriate row normalizing value if $s = r$), is irreducible if α is, and can quite easily be shown to have π as its stationary distribution, see [9], [10]. The mixing properties of the Markov chain depend on α, but beyond that the selection kernel can be chosen as need be.

4.2 MCMC on the Positions Metropolis-Hastings can be applied to our present problem, with the aim of constructing a chain on the set of position functions, $S = G^V$, that has (4.2) as its stationary distribution [4]. Let α be a selection kernel on S, and ϕ_2 be chosen by α from ϕ_1. It follows that, if we let $\alpha(\phi_1,\phi_2) = \alpha(\phi_2,\phi_1)$, and assume a uniform a-priori distribution, then:

$$\begin{aligned}\beta(\phi_1,\phi_2) &= \min\left(1, \frac{\Pr(E|\phi_2)}{\Pr(E|\phi_1)}\right) \\ &= \min\left(1, \prod_{i=1}^{m} \frac{d(\phi_1(x_i),\phi_1(y_i))^k}{d(\phi_2(x_i),\phi_2(y_i))^k}\right)\end{aligned}$$

Let ϕ_2 be an x,y-switch of ϕ_1 if $\phi_1(x) = \phi_2(y)$, $\phi_1(y) = \phi_2(x)$, and $\phi_1(z) = \phi_2(z)$ for all $z \neq x,y$. In such cases, the above simplifies by cancellation to:

$$\beta(\phi_1,\phi_2) = \min\left(1, \prod_{i \in \mathrm{E}(x \vee y)} \frac{d(\phi_1(x_i),\phi_1(y_i))^k}{d(\phi_2(x_i),\phi_2(y_i))^k}\right) \tag{4.3}$$

where $\mathrm{E}(x \vee y)$ denotes the edges connected to x or y. This function depends only on edge information that is local to x and y.

We are now free to choose a symmetric selection kernel according to our wishes. The most direct choice is to choose x and y randomly and then to select ϕ_2 as the x,y-switch of ϕ_1. This is equivalent to the kernel:

$$\alpha(\phi_1,\phi_2) = \begin{cases} 1/(n + \binom{n}{2}) & \text{if } x,y\text{-switch} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The Markov chain on S with transition matrix

$$P(\phi_1,\phi_2) = \alpha(\phi_1,\phi_2)\beta(\phi_1,\phi_2)$$

with α and β given by (4.4) and (4.3) respectively, is thus the Metropolis-Hastings chain with (4.2) as its stationary distribution. Starting from any position function, it eventually converges to the sought a-posteriori distribution.

A problem with the uniform selection kernel is that we are attempting to find a completely distributed solution to our problem, but there is no distributed way of picking two nodes uniformly at random. In practice, we instead start a short random walk at x, and use as y the node where the walk terminates. This requires no central element. It is difficult to specify the kernel of selection technique explicitely, but we find it more or less equivalent to the one above. See Section 8 below.

[4] Another way of looking at this is as an example of *Simulated Annealing*, which uses the Metropolis-Hastings method to try to minimize an energy function. In this case, the energy function is just the log sum of the edge distances, and the β coefficient is 1.

5 Experiments

In order to test the viability of the Markov Chain Monte-Carlo method, we test the chain on several types of simulated data. Working with the one-dimensional case, where the base graph is a circle, we simulate networks of different sizes according to Kleinberg's model, by creating the shortcuts through random matching of nodes, and with the probability of shortcuts occurring inverse squarely proportional to their length. We then study the resulting configuration in several ways, depending on whether the base graph is recreated after the experiment, and whether, in case it is not, we stop when reaching a dead-end node of the type described above.

We also study the algorithm in two dimensions, by simulating data on a grid according to Kleinberg's model, and using the appropriate Markov chain for this case. Finally, we study some real life data sets of social networks, to try to determine if the method can be applied to find routes between real people.

The simulator used was implemented in C on Linux and Unix based systems. Source code, as well as the data files and the plots for all the experiments, can be found at:

`http://www.math.chalmers.se/~ossa/swroute/`

6 Experimental Methodology

6.1 One-Dimensional Case We generated different graphs of the size $n = 1000 * 2^r$, for r between 0 and 7. The base graph is taken to be a ring of n points. Each node is then given $3 \log_2 n$ random edges to other nodes. Since all edges are undirected, the actual mean degree is $6 \log_2 n$, with some variation above the base value. This somewhat arbitrary degree is chosen because it keeps the probability that a route never hits a dead end low when the edges are chosen according to Kleinberg's model. Edges are sent randomly clockwise or counterclockwise, and have length between 1 and $n/2$, distributed according to three different models.

1. Kleinberg's model, where the probability that the edge has length d is proportional to $1/d$.
2. A model with edges selected uniformly at random between nodes.
3. A model where the probability of an edge having length d is proportional to $1/d^2$.

Both the latter cases are non-optimal: the uniform case represents "too little clustering", while the inverse square case represents "too much". In Kleinberg's result, the two types of graphs are shown not to have log-polynomial search times in different ways: too much clustering means not enough long edges to quickly advance to our destination, too little means not enough edges that take even closer when we are near it.

Performance on the graphs can be measured in three different ways as well. In all cases, we choose two nodes uniformly, and attempt to find a greedy route between them by always selecting the neighbor closest (in terms of the circular distance) to the destination. The difference is when we encounter a dead end – that is to say a node that has no neighbor closer to the destination then itself. In this case we have the following choices on how to proceed:

1. We can terminate the query, and label it as unsuccessful.
2. We can continue the query, selecting the best node even if it is further from the destination. In this case it becomes important that we avoid loops, so we never revisit a node.
3. We can use a "local connection" to skip to a neighbor in the base from the current node, in the direction of the destination.

For the second case to be practical, it is necessary that we limit the number of steps a query can take. We have placed this limit as $(\log_2 n)^2$, at which point we terminate and mark the query unsuccessful. This value is of course highly arbitrary (except in order), and always represents a tradeoff between success rate and the mean steps taken by successful queries. This makes such results rather difficult to analyze, but it is included for being the most realistic option, in the sense that if one was using this to try to search in a real social network, the third case is unlikely to be an option, and giving up, as in the first case, is unnecessary.

We look at each result for the graph with the positions as they were when it was generated, after shuffling the positions randomly, and finally with positions generated by a running the Markov Chain for $6000n$ iterations. It would, of course, be ideal to be able to base such a number off a theoretical bound on the mixing time, but we do not have any such results at this time. The number has been chosen by experimentation, but also for practical purposes: for large n the numerical complexity makes it difficult to simulate larger orders of iterations in practical time-scales.

Due to computational limitations, the data presented is based off only one simulation at every size of the graph. However, at least for graphs of limited size, the variance in the important qualities has been seen to be small, so we feel that the results are still indicative of larger trends. The relatively regular behavior of the data presented below strengthens this assessment.

After shuffling and when we continue at dead ends, the situation is equivalent to a random walk, since the greedy routing gains from the node positions. Searching by random walk has actually been recommended in several papers ([3], [8]), so this gives the possibility of comparing our results to that.

6.2 Two Dimensional Case We also simulate Kleinberg's model in two dimensions, generating different graphs of the size $n = 1024 * 4^r$, for r between 0 and 3. A toric grid as the base graph (that is to say, each line is closed into a loop). Shortcuts were chosen with the vertex degrees as above, and with ideal distribution where the probability that two nodes are connected decreasing inverse squared with distance (the probability of an edge having length d is still proportional to $1/d$, but as d increases there are more choices of nodes at that distance). We do this to compare the algorithm in this setting to that in the one dimensional case.

We also try, for graphs with long range connections generated against a two dimensional base graph, to use the algorithm in one dimension, and vice versa. This is to ask how crucial the dimension of the base grid is to Kleinberg's model: whether the essential characteristics needed for routing carry over between dimensions. Any conclusion on the subject, of course, is subject to the question of the performance of the algorithm.

6.3 Real World Data Finally, we test the method on a real graph of social data. The graph is the "web of trust" of the email cryptography tool Pretty Good Privacy (PGP) [1]. In order to verify that the person who you are encrypting a message for really is the intended recipient, and that the sender really is who he claims to be, PGP has a system where users cryptographically sign each others keys, thereby vouching for the key's authenticity. The graph in question is thus a sample of people that know each other "in real life" (that is outside the Internet), since the veracity of a key can only be measured through face to face contact.

We do not look at the complete web of trust, which contained about 23,000 users, but only at smaller subsets. The reason for this is two-fold. Firstly, the whole network is not a connected component. Secondly a lot of the nodes in the graph are in fact leaves, or have only one or two vertices. Under such conditions, the algorithm (or any greedy routing for that matter) cannot be expected to work.

These were created by starting a single user as the new graph's only vertex, and recursively growing the graph in the following manner. If G_n is the new graph at step n:

1. Let ∂G_n be the vertices with at least one edge into G_n, but who are not in G_n themselves.
2. Select a node x randomly from those members of ∂G_n who have the greatest number of edges into G_n.
3. Let G_{n+1} be the graph induced by the vertices of G_n and x.
4. Repeat until G_{n+1} is of the desired size.

This procedure is motivated by allowing us to get a connected, dense, "local" subgraph to study. It is closest we can come to the case where, having access to the base graph, one uses a only the nodes in a particular section of it and the shortcuts between them.

Daily copies of the web of trust graph are available at the following URL:

`http://www.lysator.liu.se/~jc/wotsap/`

7 Experimental Results and Analysis

7.1 One Dimensional Case Experimental results in the one dimensional case were good in most, but not all, cases. Some of the simulated results can be seen in 1 through 8. Lines marked as "start" show the values with the graphs as they were generated, "random" show the values when the positions have been reassigned randomly (this was not done for the random matchings case, as there is no difference from the start), and "restored" show the values after our algorithm has been used to optimize the positions.

In the ideal graph model, when the original graph is known to allow log polynomial routing, we can see that the algorithm works well in restoring the query lengths. In particular, Figure 3, where queries have been able to use the base graph, shows nearly identical performance before and after restoration.

In the cases where queries cannot use the local connections, we see that proportion of queries that are successful is a much harder property to restore than the

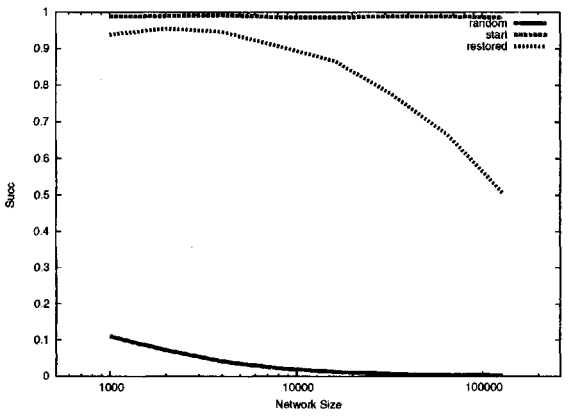


Figure 1: The success-rate of queries when terminating at dead-end nodes, on a graph generated by the ideal model.

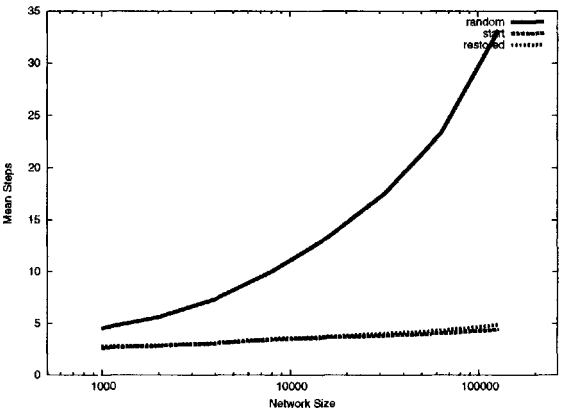


Figure 3: Mean number of steps of successful queries when allowed to use local connections, on a graph generated by the ideal model.

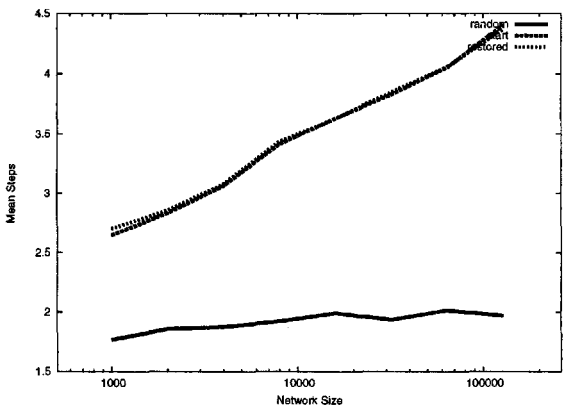


Figure 2: Mean number of steps of successful queries when terminating at dead-end nodes, on a graph generated by the ideal model.

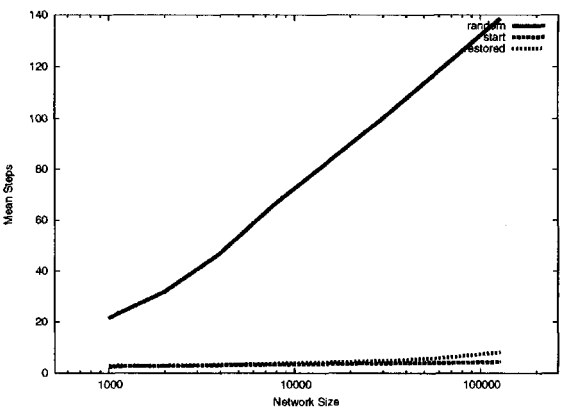


Figure 4: Mean number of steps of successful queries when terminating after $(\log_2(n))^2$ steps, on a graph generated by the ideal model.

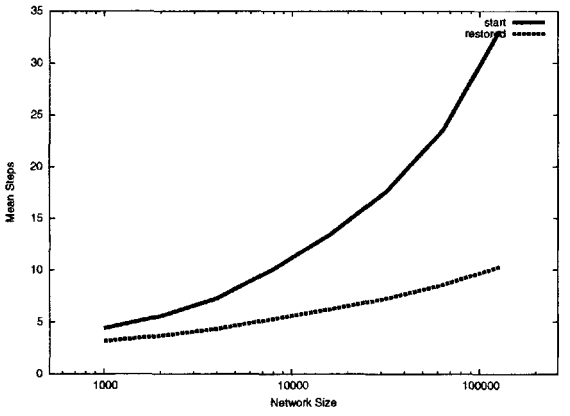


Figure 5: Mean number of steps of successful queries when allowed to use local connections, on a graph generated by random matchings.

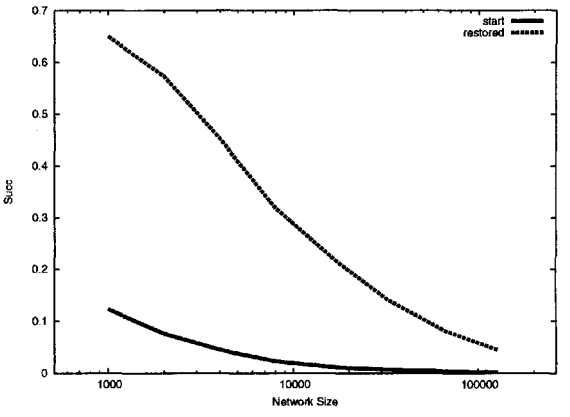


Figure 7: The success-rate of queries when terminating at dead-end nodes, on a graph generated by random matchings.

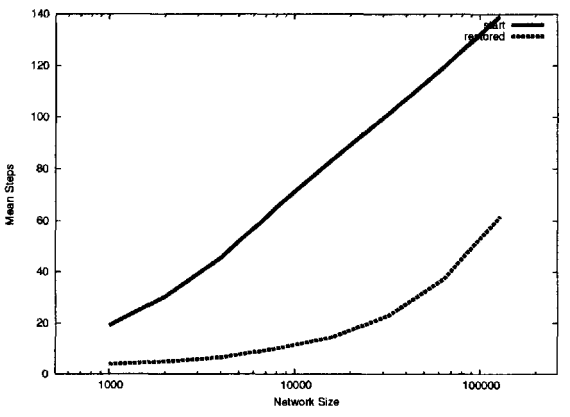


Figure 6: Mean number of steps of successful queries when terminating after $(\log_2(n))^2$ steps, on a graph generated by random matchings.

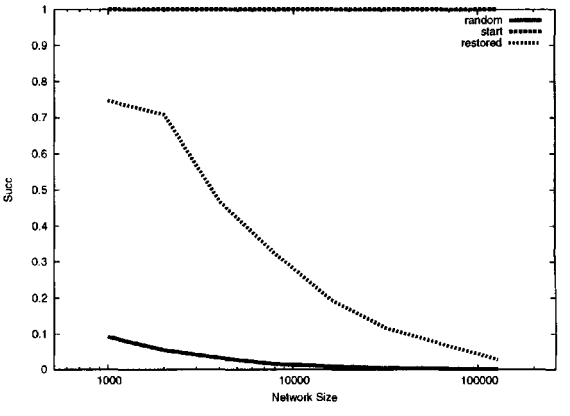


Figure 8: The success-rate of queries when terminating at dead-end nodes, on a graph generated with connection probabilities inverse square proportional to the length.

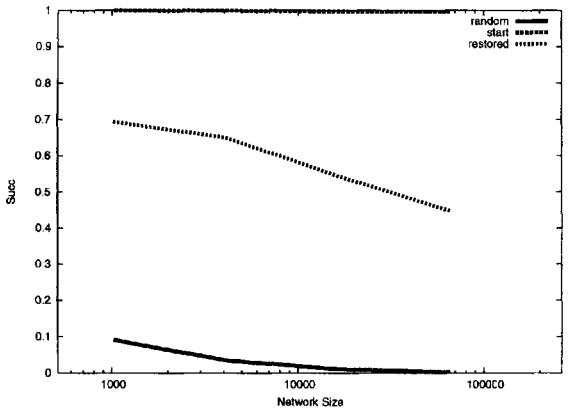


Figure 9: Matching Kleinberg's model in 2 dimensions against a graph generated according to it. Success rate when failing at dead-end nodes.

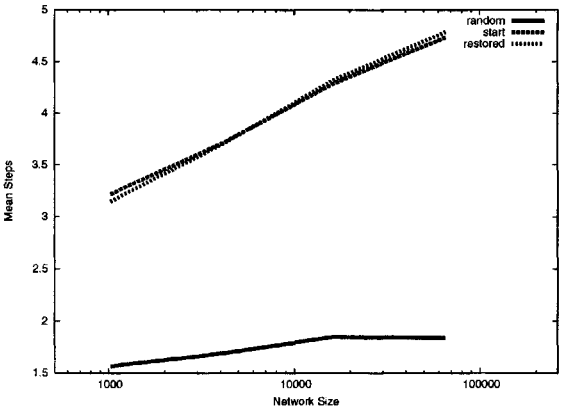


Figure 10: Matching Kleinberg's model in 2 dimensions against a graph generated according to it. Mean number of steps of successful queries when failing at dead-end nodes.

number of steps taken. Figure 1 shows this: for large graphs the number of queries that never encounter a dead-end falls dramatically. A plausible cause for this is that it is easy for the algorithm to place the nodes in the approximately right place, which is sufficient for the edges to have approximately the necessary distribution, but a good success rate depends on nodes being exactly by those neighbors to which they have a lot edges.

Along with the ideal data, two non-ideal cases were examined. In the first case, where the long range connections were added randomly, the algorithm performs surprisingly well. At least with regard to the number of steps, we can see a considerable improvement at all sizes tested. See in particular Figures 6 and 5. However, it is impossible for the success rate to be sustained for large networks when the base graph is not used - in this case there simply is no clustering in the graph - and as expected the number of successful queries does fall as n grows (Figure 7.

The other non-ideal case, that of too much clustering, was the one that faired the worst. Even though this leads to lots of short connections, which one would believe could keep the success rate up, this was not found to be the case. Both the success rate and the mean number of steps of the successful queries were not found to be significantly improved by the algorithm in this case. The results in Figure 8 if particularly depressing in this regard. It should be noted that it has been shown [14] that graphs generated in this way are not small-world graphs - their diameter is polynomial in their size, so there is no reason to believe that they can work well for this type of application.

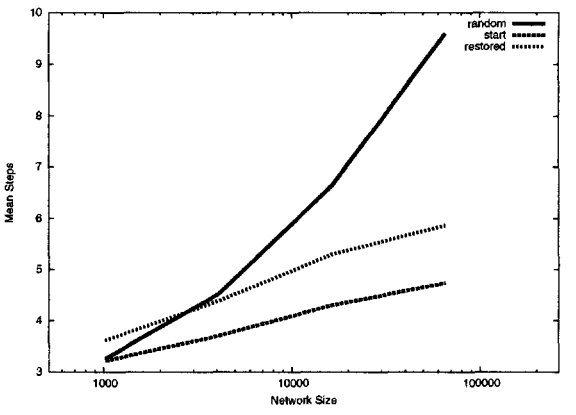


Figure 11: Matching Kleinberg's model in 2 dimensions against a graph generated according to it. Mean number of steps of queries when they are allowed to use local connections.

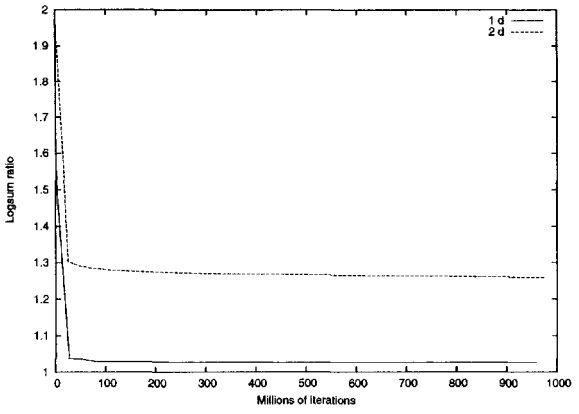


Figure 12: The target function of the optimization (log sum of shortcut distances) as the algorithm progresses. The graphs have 10000 nodes with edges generated using the ideal model. The values are normalized by dividing by the log sum of the original graph: it can be seen that we come much closer to restoring this value in 1 dimension.

7.2 Two Dimensional Case The algorithm was also simulated with a pure two dimensional model. In general, the algorithm does not perform as well as in the one dimensional case, but it performs better than against the one dimensional algorithm did on the graphs generated from non-ideal models. See Figures 9 to 11 for some of the data.

It seems that the algorithm proposed here simply does not function as well in the two-dimensional case. In Figure 12 the sum of the logarithms of the shortcut distances for a graph is plotted as the optimization is run for a very large number of iterations. It indicates that results in two-dimensions cannot be fixed by simply running more iterations, in fact, it seems like it fails to converge to one completely.

Graphs generated according to the two dimensional model were also given to the one dimensional algorithm, and vice versa. We found that data from either model was best analyzed by fitting it against a base graph of the same dimension - but the two dimensional method actually did slightly better on one-dimensional data than its own. For example at a network size of 4096, we were able to restore a success rate of 0.670 when failing at dead-ends using the two dimensional method for one dimensional data, but only 0.650 on data from the two dimensional model. This indicates that the worse performance in two dimensions may be largely due to Kleinberg's model in higher dimensions being more difficult to fit correctly.

7.3 Real World Data We treated the real world data in the same way as the simulated graphs. 2000 and 4000 vertex subgraphs were generated using the procedure defined above, the nodes were given random positions in a base graph, and then $6000n$ iterations of the Metropolis-Hastings algorithm was performed. We tried embedding the graph both in the one dimensional case (circle) and two (torus). In one dimension, the results were as follows:

Size	2000	4000
Mean degree	64.6	46.4
F Success	0.609	0.341
F Steps	2.99	3.24
C Succ	0.981	0.798
C Steps	13.4	26.0
LC Steps	4.58	7.21

Here "F Success/Steps" denotes the values when we fail upon hitting a dead end, "C Succ/Steps" when we continue and "LC steps" is the mean number of steps for queries that use the local connections at dead ends.

The data was also tested using two-dimensional coordinates and distance. The results are rather similar, with some of the tests performing a little bit better, and some (notably the success rate when failing on dead ends) considerably worse.

Size	2000	4000
F Success	0.494	0.323
F Steps	2.706	3.100
C Succ	0.984	0.874
C Steps	13.116	22.468
LC Steps	3.920	5.331

It perhaps surprising that using two dimensions does not work better, since one would expect the greater freedom of the two dimensional assignment to fit better with the real dynamics of social networks (people are, after all, not actually one a circle). The trend was similar with three-dimensional coordinates, which led to success rates of 0.42 and 0.26 respectively for the large and small graphs when failing at dead-ends, but similar results to the others when continuing. As can be seen from simulations above, the algorithm does not seem to perform very well in general in higher dimensions, and this may well be the culprit.[5]

[5]There is a general perception that the two-dimensional case represents reality, since peoples geographical whereabouts are two-dimensional. We find this reasoning somewhat specious. The true metric of what makes two people closer (that is, more likely to know one another) is probably much more complicated than

The two thousand node case has about the same degree as the simulated data from the graphs above, so we can compare the performance. From this we can see that the "web of trust" does not nearly match the data from the ideal model in any category. It does, however, seem to show better performance than the uniform matchings in some cases - most notably the crucial criteria of success rate when dropping at dead ends.

To look at the 4000 nodes case, the mean degree is considerably less than the experiments presented below, and it the results are unsurprisingly worse. In this case however, the dataset does have a lot of nodes with only a few neighbors, and it is easy to understand it is difficult for the algorithm to place those correctly.

At first glance, these results may seem rather negative, but we believe there is cause for cautious optimism. For one thing, success rates when searching in real social networks have always been rather low. In [13], when routing using actual geographic data, only 13% of the queries were successful. They used a considerably larger and less dense graph than ours, but on the other hand they required only that the query would reach the same town as the target. [2] showed similar results when attempting to route among university students. Real world Milgram type experiments have never had high success rates either: Milgram originally got only around 20% of his queries through to the destination, and a more recent replication of the experiment using the Internet [6] had as few as 1.5% of queries succeed.

Moreover, there have not been, to the authors knowledge, any previously suggested methods for routing when giving nothing but a graph. Methods suggested earlier for searching in such situations have been to either walk randomly, or send queries to nodes of high degree. With this in mind, even limited success may find practical applications.

8 Distributed Implementation and Practical Applications

The proposed model can easily be implemented in a distributed fashion. The selection kernel used in the simulations above is not decentralized, in that it involves picking two nodes x and y uniformly from the set. However, the alternative method is that nodes start random walks of some length at random times, and then propose to switch with the node at which the walk terminates. Simulating this with random walks of length $\log_2(n)/2$ (the log scaling motivated by the presumed log scaling of the graphs diameter) did not perform measurably worse in simulations than a uniform choice (nor on the collected data in the last section)[6]. For example, in a graph of 64,000 nodes generated with the ideal distribution, we get(with the tests as described above):

Test	Success Rate	Mean Steps
Fail	0.668	4.059
Continue	0.996	6.039
Base Graph	1.0	4.33

Once the nodes x and y have established contact (presumably via a communication tunnel through other nodes), they require only local data in order to calculate the value in (4.3) and decide whether to switch positions. The amount of network traffic for this would be relatively large, but not prohibitively so.

In a fully decentralized setting, the algorithm could be run with the nodes independently joining the network, and connecting to their neighbors in the shortcut graph. They then choose a position randomly from a continuum, and start initiating exchange queries at random intervals. It is hard to say when such a system could terminate, but nodes could, for example, start increasing the intervals between exchange queries after they have been in the network long. As long as some switching is going on, of course, a nodes position would not be static, but at any particular time they may be reachable.

The perhaps most direct application for this kind of process, when the base graph is a social network between people, is an overlay network on the Internet, where friends connect only to each other, and then wish to be able to communicate with people throughout the network. Such networks, because they are difficult to analyze, have been called "Darknets", and sometimes also "Friend-to-Friend" (F2F) networks.

9 Conclusion

We have approached a largely unexplored question regarding how to apply small-world models to actually find greedy paths when only a graph is presented. The method we have chosen to explore is a direct application of the well known Metropolis-Hastings algorithm, and

just geography (the author of this article is, for instance, perhaps more likely to know somebody working in his field in New Zealand, than a random person a town or two away). In any case, there is a trade-off between the realism of a certain base graph, and how well the optimization seems to function, which may well motivate less realistic choices.

[6]The most direct decentralized method, that nodes only ever switch positions with their neighbors, did not work well in simulation.

it yields satisfactory results in many cases. While not always able to restore the desired behavior, it leads to better search performance than can be expected from simpler methods like random searches.

Much work remains to be done in the area. The algorithm depends, at its heart, on selecting nodes who attempt to switch positions with each other in the base graph. Currently the nodes that attempt to switch are chosen uniformly at random, but better performance should be possible with smarter choice of whom to exchange with. Something closer to the Gibbs sampler, where the selection kernel is the distribution of the sites being updated, conditioned on the current value of those that are not, might perhaps yield better results.

Taking a step back, one also needs to evaluate other methods of stochastic optimization, to see if they can be applicable and yield a better result. No other such method, to the author's knowledge, applies as directly to the situation as the Metropolis-Hastings/simulated annealing approach used here, but it may be possible to adapt other types of evolutionary methods to it.

Also, all the methods explored here are based on the geographic models that Kleinberg used in his original small-world paper [11]. His later work on the dynamics of information [12] (and also [18]), revisited the problem with hierarchical models, and finally a group based abstraction covering both. It is possible to apply the same techniques discussed below to the other models, and it is an interesting question (that goes to the heart of how social networks are formed) whether the results would be better for real world data.

The final question, whether this can be used successfully to route in real life social networks is not conclusively answered. The results on the limited datasets we have tried have shown that while it does work to some respect, the results are far from what could be hoped for. Attempting to apply this method, or any derivations thereof, to other real life social networks is an important future task.

References

[1] A. Abdul-Rahman. The pgp trust model. *EDI-Forum: the Journal of Electronic Commerce*, 1997.

[2] L. Adamic and E. Adar. How to search a social network. *Social Networks*, 27:187–203, 2005.

[3] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Physical Review E*, 64 (46135), 2001.

[4] B. Bollobas and F. Chung. The diameter of a cycle plus a random matching. *SIAM Journal on Discrete Mathematics*, 1:328–333, 1988.

[5] A. Clauset and C. Moore. How do networks become navigable? *Preprint*, 2003.

[6] P. S. Dodds, M. Roby, and D. J. Watts. An experimental study of search in global social networks. *Science*, 301:827, 2003.

[7] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. *Theory of Computer Science*, 1:237–267, 1978.

[8] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *INFOCOM*, 2004.

[9] O. Häggström. *Finite Markov Chains and Algorithmic Applications.* Number 52 in London Mathematical Society Student Texts. Cambridge University Press, 2002.

[10] W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[11] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[12] J. Kleinberg. Small-world phenomena and the dynamics of information. In *Advances in Neural Information Processing Systems (NIPS) 14*, 2001.

[13] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins. Geograph routing in social networks. In *Proceedings of the National Academy of Science*, volume 102, pages 11623–11628, 2005.

[14] C. Martel and V. Nguyen. Analyzing kleinberg's (and other) small-world models. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 179–188, New York, NY, USA, 2004. ACM Press.

[15] S. Milgram. The small world problem. *Psychology Today*, 1:61, 1961.

[16] O. Sandberg and I. Clarke. An evolving model for small world neighbor selection. draft, 2005.

[17] D.J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness.* Princeton University Press, 1999.

[18] D.J. Watts, P. Dodds, and M. Newman. Identity and search in social networks. *Science*, 296:1302–1305, 2002.

[19] D.J. Watts and S. Strogatz. Collective dynamics of small world networks. *Nature*, 393:440, 1998.

[20] B. Yu and M. Singh. Search in referral network. In *Proceedings of AAMAS Workshop on Regulated Agent-Based Social Systems: Theories and Applications*, 2002.

Engineering Multi-Level Overlay Graphs for Shortest-Path Queries*

Martin Holzer† Frank Schulz† Dorothea Wagner†

Abstract

An overlay graph of a given graph $G = (V, E)$ on a subset $S \subseteq V$ is a graph with vertex set S that preserves some property of G. In particular, we consider variations of the multi-level overlay graph used in [21] to speed up shortest-path computations. In this work, we follow up and present general vertex selection criteria and strategies of applying these criteria to determine a subset S inducing an overlay graph. The main contribution is a systematic experimental study where we investigate the impact of selection criteria and strategies on multi-level overlay graphs and the resulting speed-up achieved for shortest-path queries. Depending on selection strategy and graph type, a centrality index criterion, a criterion based on planar separators, and vertex degree turned out to be good selection criteria.

1 Introduction

Given a graph and a subset of distinguished vertices, an *overlay graph* describes a topology defined on these vertices, where edges correspond to paths in the underlying graph[1]. Consider, for example, as base graph the Internet graph representing connections between hosts and as overlay graph the topology of a peer-to-peer network. Depending on the application, overlay graphs are demanded to fulfill certain requirements such as high connectivity and reliability in the case of peer-to-peer networks.

In the course of this paper, we give several criteria and strategies for selecting the vertices upon which an overlay graph is to be built. Following the *multi-level graph approach* introduced in [21], a speed-up technique for exact single-source shortest-path computations, here we focus on overlay graphs used for this kind of application. With the multi-level approach, one or more levels of overlay graphs inheriting shortest-path lengths from the base graph are constructed. Then a shortest-path computation takes place in a graph consisting basically of one of the overlay graphs and some additional edges.

We further refine this approach by introducing *shortest-path overlay graphs*, which are overlay graphs preserving the shortest-path property and additionally have minimal number of edges. Besides providing a procedure to construct such overlay graphs, we prove their optimality.

In [20], it is shown that the multi-level approach with one additional level is quite successful applied to timetable information, and [21] provides a generalization to multiple levels. In both studies, application-specific information is used to determine the overlay graphs.

In this work, however, we focus on getting along without application-specific information. By the help of extensive experiments we investigate the impact of our general selection criteria on the quality of multi-level graphs and measure speed-up when using them for shortest-path search. To assure general applicability, we consider different real-world and generated graphs. It turns out that with global selection strategy, vertex degree is a good criterion. The best results, however, are achieved with recursive decomposition strategy and either centrality index or planar-separator criterion.

The paper is organized as follows: In Section 2, shortest-path overlay and multi-level graphs are introduced and their application to speed up shortest-path computation is presented in Section 3. Regular multi-level graphs are theoretically analyzed in Section 4. Section 5 provides an extensive experimental study.

1.1 Related Work There are numerous approaches to speed up single-pair shortest-path computations, most of them improving Dijkstra's algorithm [5] (see [24] for a survey): On the one hand there are speed-up techniques that can be applied without any preprocessed information, for example, goal-directed and bidirectional search [1]. On the other hand, much better speed-up factors can be reached when additional information computed in a preprocessing step can be used. Such techniques represent a trade-off between precomputing all-pairs shortest paths (requiring space and preprocessing time at least quadratic in the number of ver-

*This work was partially supported by the IST Programme of EC under contract no. IST-2002-001907 (DELIS).

†Address: Universität Karlsruhe (TH), Fakultät für Informatik, Postfach 6980, 76128 Karlsruhe, Germany. Email: {mholzer,fschulz,dwagner}@ira.uka.de.

[1] We use the term "overlay graph" in this general sense. Note that sometimes "overlay graph" is used only in a specific context, for example, associated with certain properties like uniform vertex degree.

tices) and no additional information at all. Also, combinations of speed-up techniques are possible [7, 11, 20].

One class of preprocessing techniques [6, 8, 15, 16, 18, 20, 22] prunes the search space of Dijkstra's algorithm: some of the edges and vertices Dijkstra's algorithm would consider can safely be ignored since by the additional information, the algorithm knows that they cannot be part of a shortest path from the given source to the given target.

Another common approach, which makes use of an auxiliary graph and is also followed in this paper, is to hierarchically decompose the graph into connected components [9, 12, 13, 20, 21]. Such techniques exploit the fact that—if the source s and the target t are not in the same component at some level—any s-t-path will pass through the border of the respective components. Additional edges connecting border vertices are maintained and to find the shortest path one can show that an appropriate choice of edges in the components around the source and target and between border vertices suffices.

Now, we discuss two closely related hierarchical techniques in more detail and show how shortest-path overlay graphs, which will be defined later on in this paper, can be applied therein.

HiTi graphs introduced by Jung and Pramanik [13] are applied in the area of car navigation. The basic difference to our approach is that the input graph is decomposed into connected components by edge separators instead of vertex separators. For each component, a complete graph on the boundary vertices stores the shortest-path lengths between these vertices. As in our approach, an appropriate subgraph is used to answer a shortest-path query. The additional shortest-path information on boundary vertices can be alternatively maintained by shortest-path overlay graphs.

Another recent study [18] precomputes a hierarchy on the edges. Shortest paths are computed by a variant of bidirectional search. The precomputed edge hierarchy helps in that during the bidirectional search only edges of a certain level have to be considered, depending on the distance from the source and to the target. The subgraph induced by edges belonging to one level is compressed: iteratively, all vertices of degree 1 are removed and paths of degree-2 vertices are replaced by edges. This compression technique can be regarded as the construction of a (not necessarily minimal) overlay graph with inherited shortest-path lengths on the remaining set of vertices.

2 Overlay Graphs

Let $G = (V, E)$ be a directed, connected graph with non-negative edge lengths. Unless stated otherwise, n denotes the number of vertices and m the number of edges in G. The *length* of a path is the sum of the lengths of the edges on the path. In this section, we will define shortest-path overlay graphs and two variants of multi-level graphs, which refines the approach from [21].

2.1 Shortest-Path Overlay Graph For a subset S of the graph's vertices we seek a graph G' with vertex set S and shortest-path lengths inherited from G: For each pair of vertices $u, v \in S$, shortest u-v-paths have the same length in G and G'. Additionally, we want G' to contain as few edges as possible. First, we formally define shortest-path overlay graphs and show afterwards that these graphs meet the above requirements.

DEFINITION 2.1. *Given a subset $S \subseteq V$, the* shortest-path overlay graph $G' := (S, E')$ *is defined as follows: for each $(u, v) \in S \times S$, there is an edge (u, v) in E' if and only if for* any *shortest u-v-path in G no internal vertex belongs to S. (Internal vertices are all vertices on the path except u and v.)*

Note that in [21], a different condition for $e \in E'$ is used: For each vertex $u \in S$, a standard Dijkstra's algorithm is used to compute a shortest-path tree T_u. Then an edge is added to E' if *the* path in T_u contains no internal vertex belonging to S—there can be another shortest path that contains an internal vertex belonging to S and G' contains redundant edges in this case. In contrast, for our new approach we can prove uniqueness and minimality of the shortest-path overlay graph. A minimal number of edges is desirable for applications like speed-up techniques for shortest-path algorithms since fewer edges result in a smaller search space and hence a better running time. Figure 1 depicts two sample overlay graphs, one computed by the procedure suggested in [21], and the minimal shortest-path overlay graph computed by the subsequent new procedure `min-overlay`, which implements the above Definition 2.1:

> Procedure `min-overlay`(G, l, S)
>
> For each vertex $u \in S$ run Dijkstra's algorithm in G with pairs (l_e, s_e) as edge weights, where l_e is the edge length, $s_e := -1$ if the tail of e belongs to $S \setminus \{u\}$, and $s_e := 0$ otherwise. Addition is done pairwise, and the order is lexicographic. The result of Dijkstra's algorithm are distance labels (l_v, s_v) at the vertices. For each $v \in S \setminus \{u\}$ we introduce an edge (u, v) in E' with length l_v if and only if $s_v = 0$.

THEOREM 2.1. *The shortest-path overlay graph $G' = (S, E')$ computed by* `min-overlay` *inherits shortest-path lengths from G and the number of edges $|E'|$ is minimal among all graphs with vertex set S and inherited*

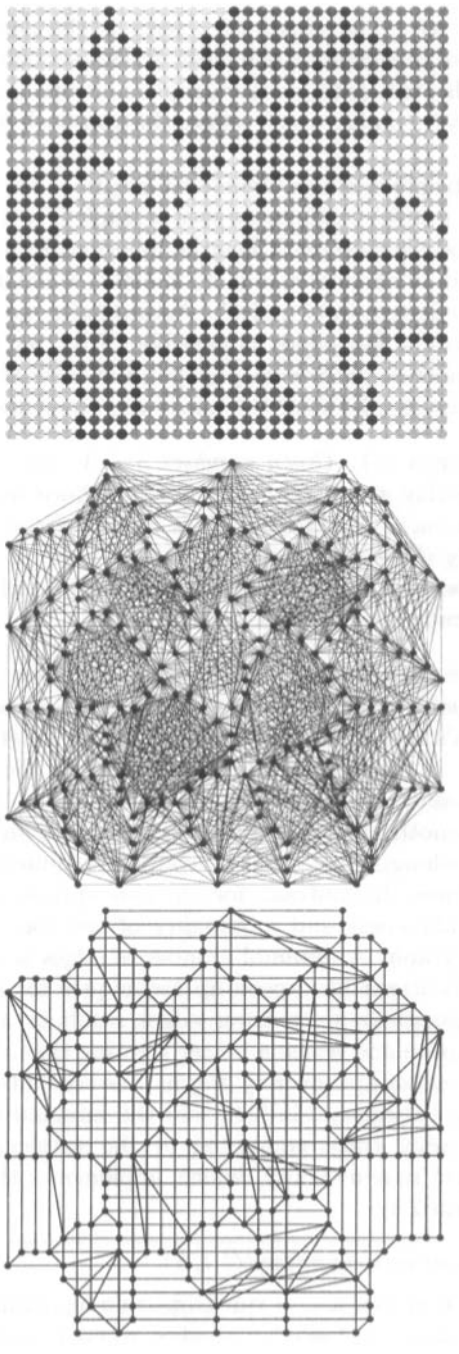


Figure 1: From top to bottom: a decomposition (by the planar-separator algorithm) of a 32 × 32 grid by a separator of 208 vertices (black), an overlay graph which is computed by a standard Dijkstra's algorithm (2865 directed edges), and the corresponding minimal shortest-path overlay graph (1076 directed edges).

shortest-path lengths. G' is the only overlay graph under these constraints.

Proof. Let p be a shortest s-t-path in G with $s, t \in S$; consider the subpaths $p_1, \ldots, p_k$ of p divided at all vertices in S (i.e., the first and the last vertex of every p_i is in S, and no internal vertex of p_i belongs to S). Consider such a subpath $p_i = (v_1, \ldots, v_l)$. Due to the condition for edges being in E', either (i) any shortest path from v_1 to v_l in G also has no internal vertex in S and in this case there is an edge (v_1, v_l) in E'; or (ii) there is a shortest path from v_1 to v_l via a vertex $u \in S$. In that case we replace the path p_i by two subpaths $p_i' = (v_1, \ldots, u)$ and $p_i'' = (u, v_l)$. Since case (ii) can only occur a finite number of times (the two new subpaths have shorter lengths as the replaced subpath), this procedure will end up with a subdivision of p into subpaths each of which has a corresponding edge in the overlay graph G'. Hence, there is also an s-t path in G' with the same length as p. Clearly, there is no shorter path in G' since edge lengths in E' always correspond to shortest-path lengths in G.

To prove minimality and uniqueness of G', assume that there is an overlay graph (S, E'') that also inherits shortest-path lengths from G and there is an edge $e' = (u, v) \in E'$ which is not in E''. Let $(u = v_1, \ldots, v_k = v)$ be a shortest u-v-path in (S, E''). It holds that $k > 2$ because $e' = (u, v) \notin E''$. Since subpaths of shortest paths are also shortest paths, $l(v_i, v_{i+1})$ is the length of a shortest v_i-v_{i+1}-path in G $(1 \leq i < k)$. Hence, there must also be a shortest u-v-path $(u = v_1, \ldots, v_2, \ldots, v_{k-1}, \ldots, v_k = v)$ in G with (at least) one internal vertex $v_2 \in S$, in contradiction to $e \in E'$. □

Note that in the above described procedure, Dijkstra's algorithm can be terminated when $s_w < 0$ for all vertices w in the queue, since the condition $s_v = 0$ cannot be true for any vertex v whose label has not been computed yet. This yields a better running time for the construction of G'.

2.2 Basic Multi-Level Graph Iteratively repeating the `min-overlay` procedure, we obtain multiple levels of shortest-path overlay graphs with decreasing number of vertices which we call *basic multi-level (overlay) graph.* It is based on a sequence of l subsets of vertices S_i $(1 \leq i \leq l)$ which are decreasing with respect to set inclusion: $V = S_0 \supset S_1 \supset S_2 \supset \ldots \supset S_l$. A vertex v is a level-i vertex if i is the highest index such that $v \in S_i$. The additional edge sets are denoted by E_i $(1 \leq i \leq l)$. To emphasize the dependence on G and the sets $S_1, \ldots, S_l$, we shall refer to the multi-level graph by $\mathcal{M}(G; S_1, \ldots, S_l)$. Together with G as level 0, we say that $\mathcal{M}$ is a $l + 1$ level graph.

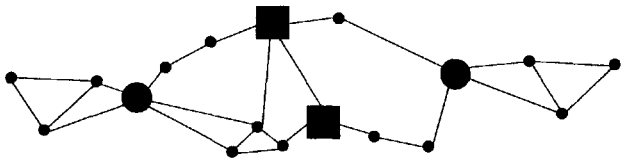


Figure 2: Sample graph with vertex selections S_1 (big circles and squares) and S_2 (squares).

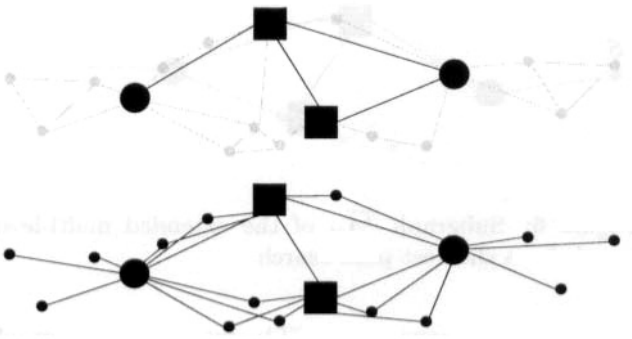


Figure 3: Additional edges in the basic (top) and extended (bottom) multi-level graph based on the selections shown in Figure 2.

2.3 Extended Multi-Level Graph In [21], multi-level graphs are introduced in a slightly different way, which will be called *extended* multi-level (overlay) graphs here, denoted by $\overline{\mathcal{M}}$. Especially when the multi-level graphs are applied to speed up shortest-path computations, further edges are helpful: *upward edges* U_i from vertices in $S_{i-1} \setminus S_i$ to vertices in S_i, and *downward edges* D_i from vertices in S_i to vertices in $S_{i-1} \setminus S_i$ $(1 \leq 1 \leq l)$. The edges E_i of the shortest-path overlay graph at level i are called *level edges*. The `min-overlay` procedure can be extended to construct also downward and upward edges: In the last step introducing the edges we consider now also vertices $v \in S \setminus V$ and introduce a downward edge if and only if $s_v = 0$. To construct upward edges, we run Dijkstra's algorithm not only for vertices $u \in S$ but also for $u' \notin S$ and introduce an edge (u', v') to vertices $v' \in S$ if and only if $s_{v'} = 0$.

Figure 2 shows a sample graph and Figure 3 the additional edges in the basic and the extended multi-level graph, respectively.

3 Shortest-Path Search

Multi-level graphs can be used to speed up single-pair shortest-path algorithms: Based on the source and the target vertex, a subgraph of the multi-level graph is determined and the shortest path is computed in that subgraph. We review the definition of the *tree of connected components* and the speed-up technique using extended multi-level graphs from [21] and provide a new variant applying the basic multi-level graph; we also briefly discuss the correctness of the new variant.

3.1 Tree of Connected Components Consider the subgraph of G that is induced by the vertices $V \setminus S_i$. We will use the following notation: The set of connected components is denoted by $\mathcal{C}_i$, and a single component is usually referred to by C. For a vertex $v \in V \setminus S_i$, let C_i^v denote the component in $\mathcal{C}_i$ that contains v.

As data structure to determine the appropriate subgraph of a multi-level graph for a pair of vertices $s, t \in V$ a tree with the connected components $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_l$ as vertices is used. Additionally, there is a root C_{l+1} and for every vertex $v \in V$ a leaf C_0^v in the tree. The parent of a leaf C_0^v is determined as follows: Let i be the largest i with $v \in S_i$. If $i = l$, the parent is the root C_{l+1}. Otherwise, the level with smallest index where v is contained in a connected component is level $i+1$, and the parent of C_0^v is the component $C_{i+1}^v \in \mathcal{C}_{i+1}$. The parent of the components in $\mathcal{C}_l$ is also the root C_{l+1}. For one of the remaining components $C_i \in \mathcal{C}_i$, the parent is the component C_{i+1}^u for a vertex $u \in C_i$.

For the given pair of vertices $s, t \in V$, we consider the C_0^s-C_0^t path in the component tree. Let L be the smallest L with $C_L^s = C_L^t$ (i.e., $C_L^s = C_L^t$ is the lowest common ancestor of C_0^s and C_0^t in the tree). Then the C_0^s-C_0^t path is

$$(C_0^s, C_k^s, C_{k+1}^s, \ldots, C_L^s = C_L^t, \ldots, C_{k'+1}^t, C_{k'}^t, C_0^t),$$

where $k > 0$ and $k' > 0$ are the levels of the parents of C_0^s and C_0^t, as defined above.

Figure 4 shows an illustration hereof.

3.2 Definition of the Subgraph The above defined path in the component tree induces a subgraph $\mathcal{M}_{st}$ of the basic multi-level graph $\mathcal{M}$ as follows: For each component $C = C_i^x$ with $x \in \{s, t\}$ and $i < L$ in the path, all edges of level i incident to a vertex in component C belong to the edge set E_{st} of the subgraph $\mathcal{M}_{st}$. Further, all edges of level L belong to E_{st}. The vertex set of $\mathcal{M}_{st}$ is induced by these edges. Figure 5 shows $\mathcal{M}_{st}$ for our sample graph.

A shortest-path search in the subgraph $\mathcal{M}_{st}$ proceeds as follows: Starting from a vertex s at level k, the original graph is searched until a vertex at a higher level is reached. Outside this component, shortest-path information is completely contained at the next higher level $k+1$. By definition of the component tree, any s-t-path leaves the component C_k^s. Hence, it suffices to maintain only level-k edges incident to vertices in component C_k^s. The part of the shortest path between

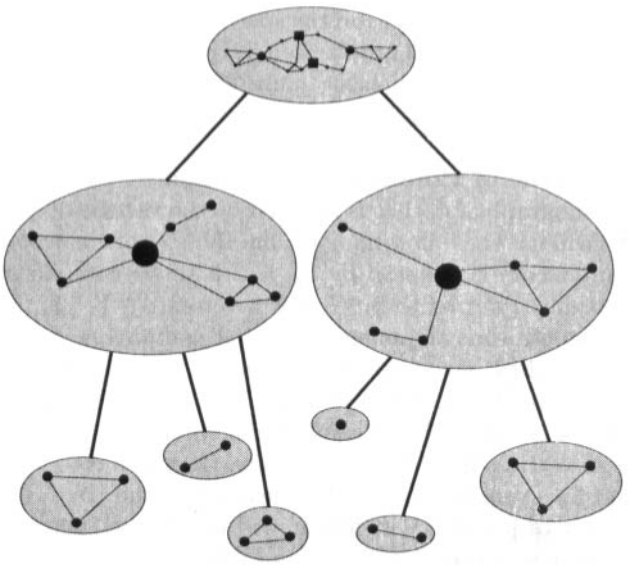


Figure 4: Component tree of the sample graph in Figure 2 (without leaves C_0^v).

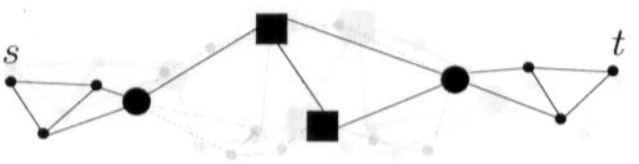


Figure 5: Subgraph $\mathcal{M}_{st}$ of the basic multi-level graph for a shortest-path search.

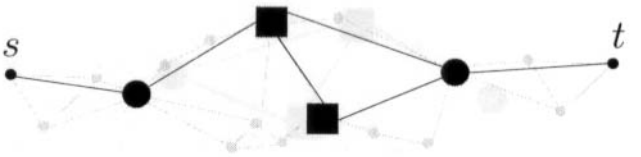


Figure 6: Subgraph $\overline{\mathcal{M}}_{st}$ of the extended multi-level graph for a shortest-path search.

level-$(k+1)$ vertices is then found using level edges of the higher levels. The same argument applies iteratively to higher levels, and symmetrically for components around t. At the highest level L, s and t belong to the same component, and all level-L edges are necessary. Concluding, we state the outcome of the above description as

LEMMA 3.1. *The length of a shortest s-t-path is the same in G and the subgraph $\mathcal{M}_{st}$ of the basic multi-level graph $\mathcal{M}(G; S_1, \ldots, S_l)$.*

In a similar way, a subgraph $\overline{\mathcal{M}}_{st}$ of $\overline{\mathcal{M}}$ with the same s-t-shortest-path length as G is induced by the following edge set: Basically, paths from vertices in component C_i^s to higher level vertices adjacent to that component are replaced by direct edges in U_i. Symmetrically, paths in C_i^t are replaced by direct edges in D_i (cf. Figure 6). The correctness of this shortest-path approach applying the extended multi-level graph is proven in [21].

3.3 Selecting Vertices for Multi-Level Graphs

The crucial point to the construction of a multi-level graph belonging to a given graph is the selection of vertices by removal of which a decomposition of the original graph into connected components is induced.

3.3.1 Selection Criteria We applied ten vertex selection criteria: One random, benchmark criterion (RND); two criteria related to vertex degree: degree (DEG) and percentage (PCT); one related to graph cores (COR) [4]; four coming from *centrality indexes*: reach (RCH) [8], closeness (CLO), betweenness (BET) and betweenness approximation (BAP) [4]; and one involving a *planar-separator algorithm* (PLS) [10]:

Random (RND). Vertices are selected uniformly at random.

Degree (DEG). The desired number of vertices with the highest degrees are selected.

Percentage (PCT). We consider for each vertex v its *percentage value*, which is the share of v's adjacent vertices that have smaller degree than v in all adjacent vertices; an isolated vertex is assigned -1. The vertices with the highest percentage values are selected.

Core (COR). The k-core of a graph (for an integer k) is the maximum subgraph such that all vertices in that subgraph have degree at least k. The core number of a vertex is defined to be the maximum k such that it belongs to the k-core; see [4] for further details. The vertices with the highest core numbers are selected.

Reach (RCH). The reach $r(v, P)$ of a vertex v on an s-t path P is defined to be $\min\{l(P_{sv}), l(P_{vt})\}$, where P_{sv} and P_{vt} are the subpaths of P with respect to v, and $l(P.)$ the respective path length. The reach $r(v)$ of v is defined to be $\max\{r(v, P) \mid P$ shortest s-t-path over $v\}$; see [8]. Reach thus denotes the greatest distance of v to the nearer of either end-vertex over all shortest paths containing v. Here, as with the two following criteria, the vertices with the greatest respective values are selected.

Closeness (CLO). Closeness of v is defined as $c(v) = 1/\sum_{t \in V} d(v,t)$, letting $d(v,t)$ denote the distance from v to t (with $1/0 := 0$); see [4] for further details. Intuitively speaking, a vertex with great closeness has short distances to most of the other vertices.

Betweenness (BET). We define betweenness of v as $b(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$, where $\sigma(s,t)$ stands for the number of shortest paths from s to t and $\sigma(s,t \mid v)$ for the number of shortest paths from s to t that contain v as an *inner* vertex (with $0/0 := 0$); see [4] for further details. Betweenness marks how important a vertex is to shortest paths.

Betweenness Approximation (BAP). The betweenness values as defined above are approximated by random sampling letting $s, t \in V' \subseteq V$, where $|V'| = (\log n)/\varepsilon^2$ for some appropriate choice of ε; see [4]. The goal of the approximation is to obtain "good enough" betweenness values in much shorter time than required for computation of the exact values (cf. Section 5.3.1). The probability of an error larger than $\varepsilon n(n-2)$ is at most $1/n$.

Planar-Separator Criterion (PLS). This criterion makes use of a planar-separator algorithm, and picks the vertices returned by this algorithm as selected vertices. We use the heuristic suggested in [10], which is based on the Planar-Separator Theorem by Lipton and Tarjan. In order to apply it also to non-planar graphs, we planarize such a graph first by introducing new vertices at crossings (assuming a straight-line embedding of the given graph). Then the planar-separator algorithm is applied to the planarized auxiliary graph. Finally, separator vertices for the original graph are inherited from the auxiliary graph, and conflicts with edges that connect two vertices of different components are resolved by declaring one of the both end vertices to be separator vertices.

Figure 7 illustrates the different criteria with a sample graph.

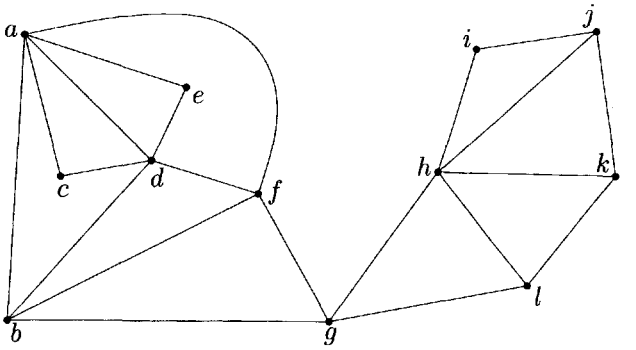


Figure 7: Highest-priority vertices in a sample graph according to the different selection criteria: DEG: a, d, h; PCT: h; COR: a, b, d, f; RCH: b, f, g; CLO: g; BET: g; PLS: $\{f, g\}$ (for instance).

3.3.2 Selection Strategies With each criterion except PLS, we consider two different strategies of selecting vertices. The first, called *global* application, is to determine, according to the criterion specified, a given number of vertices picked from the whole graph. With the second, referred to by *recursive decomposition*, a maximum component size is chosen. Recursively, for each connected component that is bigger than that threshold, the vertices are sorted according to the given criterion and from this list, the first vertices are selected so long until the graph splits or the number of non-selected vertices in this component falls below the threshold. With more than one additional level, vertices selected at one level are always selected at the lower levels, too.

PLS is a special case: it is used to decompose the graph recursively until all connected components are smaller than the desired maximum component size. Hence, PLS cannot be used as a global criterion.

4 Regular Multi-Level Graphs

The basic idea of multi-level graphs as speed-up technique for shortest path computation is that the subgraph $\overline{M}_{st}$ will be small and thus a shortest path can be found faster using that subgraph instead of the original graph G. In general, this is not necessarily true; clearly, a bad example would be if the vertices in S do not decompose the graph G at all. However, for *regular* multi-level graphs, under certain assumptions concerning the decomposition, we are able to prove a bound on the total number of edges in the multi-level graph and especially on the size of the subgraph $\overline{M}_{st}$. Note that these bounds always refer to the *extended* multi-level graphs.

The size of the subgraph $\overline{M}_{st}$ depends crucially on the highest level index L in the path of the component tree (cf. Section 3.1) determining the subgraph, so we are also interested in the probability that for a random query at least a given level L is reached. Another important parameter will be the maximum number of adjacent vertices of a component over all components, denoted by A. Due to lack of space, proofs are omitted; they can be found in [19].

4.1 Size of the Multi-Level Graph and the Search Space The first lemma shows that for multi-

level graphs that exhibit some regular hierarchical structure, the total number of additional edges can be bounded as follows. With $E_G(S)$ we denote the edge set induced by the vertex set S in the original graph G. The total number of upward, downward, and level edges is denoted by $|U|$, $|D|$, and $|LE|$, respectively.

LEMMA 4.1. *Assume that for the decomposition of the graph G into components, it holds that $\sum_{i=1}^{l} |\mathcal{C}_i| \leq n$, and that for each i, $1 \leq i < l$, the part of $E_G(S_i)$ that is not in $E_G(S_{i+1})$ is at most half as big as the same part in the previous level:*

$$|E_G(S_i) \setminus E_G(S_{i+1})| \leq |E_G(S_{i-1}) \setminus E_G(S_i)|/2.$$

Then, the total number of additional edges in the multi-level graph is at most

$$|U| + |D| + |LE| \leq m + \lceil A^2 + A \rceil n.$$

Let $(s,t) \in V \times V$ be a query selected uniformly at random, where $P(s,t) = 1/n^2$. We are interested in the probability that the lowest common ancestor of s and t in the component tree, denoted as *level*(s,t), is at least L $(1 \leq L \leq l+1)$. Assuming that all components in $\mathcal{C}_{L-1}$ have the same size C, this probability amounts to

$$(4.1) \quad P(\textit{level}(s,t) \geq L) = \frac{2|S_{L-1}|n - |S_{L-1}|^2 + (C-1)(n - |S_{L-1}|)^2/C}{n^2}.$$

Consider the highest level possible $L = l+1$, and further assume that C and the number of vertices in the smallest subset of vertices $|S_l|$ are constant. Then, with $n \to \infty$, for the probability that the highest level is used in the subgraph $\overline{\mathcal{M}}_{st}$, it holds that

$$(4.2) \quad P(\textit{level}(s,t) \geq l+1) \longrightarrow (C-1)/C.$$

Finally, we are able to give a bound on the number of edges and vertices of the subgraph $\overline{\mathcal{M}}_{st}$ used for solving a shortest-path query, depending on $L = \textit{level}(s,t)$, the level of the lowest common ancestor of s and t in the component tree.

LEMMA 4.2. *Given that the assumptions made above hold, the total number of edges in the subgraph $\overline{\mathcal{M}}_{st}$ is bounded by*

$$|E(\overline{\mathcal{M}}_{st})| \leq 2\lceil A + (L-2)A^2 \rceil + |E_{L-1}|.$$

It turns out that A and the number of edges at the highest level, $|E_{L-1}|$, are the crucial parameters to the size of $\overline{\mathcal{M}}_{st}$. For the majority of all queries, *level*$(s,t) = l+1$ (as we have seen above for random queries) and, assuming A being smaller than a constant α, the search space (the number of edges in $\overline{\mathcal{M}}_{st}$) is asymptotically dominated by the total number of levels l, which is usually in $O(\log n)$, and by E_l, the number of level edges in level l:

$$|E(\overline{\mathcal{M}}_{st})| \in O(l + |E_l|).$$

4.2 Discussion Given that the assumptions made in Lemma 4.1 are fulfilled by a reasonable decomposition of the graph, the total number of additional edges is less than $m + \lceil A^2 + A \rceil n$. Hence, the parameter A is crucial to the amount of additional space needed. Concerning the efficiency of the technique, the number of edges at the highest level, $|E_{L-1}|$, and the parameter A are important (cf. Lemma 4.2). Another issue is the similarity of the components. If there is one large component at the highest level, the probability that two vertices belong to that component is quite high such that the highest level is not used for the subgraph. In contrast, if all components have the same size C, the probability that the highest level is used is approximately $(C-1)/C$ (if the graph is large enough; cf. Inequation 4.2).

In summary, such decompositions should be used for which the number of vertices $|S_l|$ and edges E_l are reasonably small, the components $C_{i,j}$ have few adjacent vertices in the corresponding set S_i, and the components at each level should be preferably of equal size.

4.3 Component-Induced Random Graphs Motivated by the results concerning regular multi-level graphs, we will define now a random graph model that fulfills all the assumptions made above. We use such graphs afterwards in the experimental study.

4.3.1 Recursive Construction We construct a *component-induced random graph* $G_c(l, c, N, M, A)$ recursively depending on the following parameters: (i) the number of levels l; (ii) the number of components per level c; (iii) the number of new vertices N and new edges M per level; and (iv) the number of adjacent vertices per component A. Feasible values are $c > 1$, $M \leq N(N-1)$, and $A \leq N$.

The recursive procedure takes as argument a level index i, which is l at the beginning, and first computes a classical Erdös-Réyni random graph R with N vertices and M edges (i.e., R contains N vertices and M edges selected uniformly at random from all possible edges; see also [2]). If R is not connected, we repeat the construction. By setting $\mu_N = \log_2 N + 3$ and $M = \mu_N N$, the probability that R is connected is greater than 95% (cf. [2]). If the level index is 0 the procedure stops. Otherwise, c components are constructed by applying

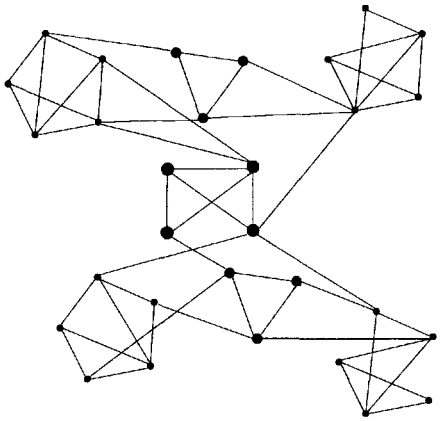


Figure 8: Sample component-induced graph with three construction levels (medium-sized and fat vertices mark selected vertices at level 2 and 3, respectively).

the recursive procedure c times with a level index of $i-1$. Finally, for each of these components, A vertices from R are selected randomly and two edges between each of these vertices and randomly selected vertices in the respective component are introduced.

An example of a component-induced graph with three construction levels can be seen in Figure 8.

4.3.2 Analysis The number of edges of a component-induced random graph is $m \leq n(\mu_N + 2)$. By setting $\mu_N = \log_2 N + 3$ as mentioned above, the graph G_c is sparse: for the number of edges in G_c, it holds that $m \leq n(\log_2 N + 5)$.

A regular multi-level graph $\overline{\mathcal{M}}(G_c; S_1, \ldots, S_l)$ can now be obtained by setting S_i to be the vertices generated at levels greater than or equal to i ($1 \leq i \leq l$). Since the multi-level graph is regular (cf. [19]), the results obtained above for regular multi-level graphs apply. Finally, we want to investigate the crucial parameter, namely the size of the subgraph $\overline{\mathcal{M}}_{st}$ for an s-t-query. By Lemma 4.2 and the fact that $|S_l| = N$ in our case, the size of the subgraph is

$$|E(\overline{\mathcal{M}}_{st})| \leq 2[A + (l-1)A^2] + N^2.$$

With constant A and the fact that the number of levels l is logarithmic in the number of vertices in G_c, we obtain that the size of the subgraph $\overline{\mathcal{M}}_{st}$ is in $O(\log n + N^2)$.

Concluding, we consider the speed-up factor $s = m/|E(\overline{\mathcal{M}}_{st})|$ indicating how much faster a shortest-path algorithm applied to the subgraph $\overline{\mathcal{M}}_{st}$ can be compared to the same algorithm applied to G_c. By the latter observation concerning the size of the subgraph, the factor s increases with the number of vertices n in G_c, given that the parameter N is kept at a constant size. This implies that the speed-up of a shortest-path algorithm can become arbitrarily high for component-induced random graphs that are sufficiently large by using the multi-level approach (e.g., for fixed parameters c, N, M, and A, increasing the number of levels l yields arbitrarily large graphs, where the other parameters remain constant).

5 Experimental Analysis

First, we introduce the graph classes considered in our experiments, investigate the `min-overlay` procedure to compute minimal shortest-path overlay graphs, and present the results of prestudies regarding betweenness approximation and the planarization technique. The main results concern the basic and extended variants of the multi-level graph approach as speed-up technique for Dijkstra's algorithm. The code is written in `C++` based on the LEDA library [17].

5.1 Graph Classes With our experiments we take into account four types of graphs, two randomly generated and two taken from real world. All graphs are connected and bidirected, i.e., each (undirected) edge is replaced with two directed edges, one in either direction.

Component-Induced Random Graphs (`ci`·). The random graphs introduced in Section 4.3 are of regular hierarchical structure and the multi-level graph approach theoretically works well for such graphs. Edge lengths are chosen at random. We computed a layout for the `ci` graphs using a spring embedder from the LEDA library [17].

Planar Delaunay Graphs (`del`·). Planar Delaunay graphs are graphs with a given number of vertices randomly spread over a unit square for which the Delaunay triangulation is computed and edges are deleted from it at random until a given number of edges has been reached. Edge lengths are the Euclidian distances of the vertices.

Road Graphs (`road`·). By road graphs we denote subgraphs of the German road network[2]. The length of an edge is the length of the corresponding road section (i.e., not the straight-line distance; granularity: 10 meters).

Railway Graph (`rail`·). Railway graphs are graphs condensed of networks reflecting train connections[2] (cf. [20]): vertices stand for railway stations, and there exists an edge between two vertices in a railway graph if there is a non-stop connection between the respective vertices in the underlying net-

[2] We are grateful to the companies HaCon, Hannover, and PTV AG, Karlsruhe, for providing us with railway and road data.

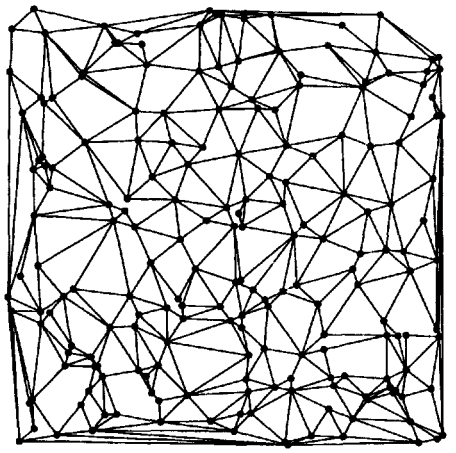


Figure 9: Sample planar Delaunay graph.

	n	unit lengths opt	unit lengths blow-up	application lengths opt	application lengths blow-up
ci	2000	7.0	1.209	6.9	1
del	10000	31.1	2.953	35.4	1
road	19463	12.6	1.264	15.8	1.014
rail	6848	5.9	1.371	8.8	1

Table 1: The optimal density (opt) is defined to be the number of edges of the minimal shortest-path overlay graph divided by the number of its vertices. The blow-up factor is the number of edges of the worst-case overlay graph divided by the number of edges of the minimal shortest-path overlay graph. We distinguish the case of unit edge lengths (left column) and edge lengths from the corresponding application (right column).

work. Graphs representing both long-distance traffic (`lrail`·) of several European countries and local, short-distance traffic (`srail`·) of several German regions are used. With this type of graph, the length of an edge is assigned the average travel time of all trains that contribute to this edge.

Sample instances of the named graph classes can be found in Figures 8, 9, and 14, respectively. A graph generator [3] for the `ci`· and `del`· graphs is available.

5.2 Shortest-Path Overlay Graphs In Theorem 2.1 we proved that `min-overlay` yields shortest-path overlay graphs that are minimal. We computed for each of the four graph classes under consideration a set of selected vertices by the recursive decomposition strategy using PLS. Then we computed minimal shortest-path overlay graphs by our new procedure `min-overlay` (cf. Section 2.1) on the one hand, and shortest-path overlay graphs computed by the procedure suggested in [21] in the worst case[3] on the other hand. Comparing these two shortest-path overlay graphs shows how much can be gained by using the new—minimal—procedure compared to the previous method.

We computed the overlay graphs with unit edge lengths as well as with real edge lengths from their application. We show the density (i.e., the number of edges divided by the number of vertices) of the minimal shortest-path overlay graph and the blow-up of the worst-case procedure (i.e., the non-optimal number of edges divided by the optimal number of edges). The outcome is depicted in Table 1. With unit edge lengths, there are many paths of the same length, and the blow-up is almost 3 for `del` graphs. For application edge lengths the picture looks different, only for `road` graphs a slight blow-up can be observed.

The missing blow-up for the `ci`, `del` and `rail` graphs in the case of application edge lengths can be explained by the fact that edge lengths are double-values representing Euclidean lengths or mean travel times. Hence, it is very unlikely that there are two paths between a given pair of vertices with the same length.

The case of unit edge lengths constitutes the other extreme, where same path lengths are highly probable. With small integer edge lengths (e.g., when actual travel times are used instead of mean travel times in the case of `rail` graphs), the blow-up factors will be somewhere between 1 and the unit edge length case.

5.3 Prestudies Concerning Selection Criteria Most of the selection criteria described in Section 3.3.1 are uniquely determined. However, for BAP and PLS we carried out prestudies investigating applicability to our scenario.

5.3.1 Betweenness Approximation In order to explore the quality of betweenness approximation with different choices of ε and to empirically determine a value for practical application, we investigated the multi-level approach for shortest-path queries with `road` graphs. We answered shortest-path queries with the extended multi-level graph approach, with exact betweenness values on the one hand, and with approximated betweenness values on the other hand, for ε between 0 and 2.

We evaluated the number of visited edges, which

[3] The procedure in [21] applies Dijkstra's algorithm. When two shortest paths of the same length are determined during a run of Dijkstra's algorithm, one of them is selected arbitrarily. This arbitrary choice can result in different shortest-path overlay graphs, and by "worst case" we mean that we computed the one resulting in a maximum number of edges.

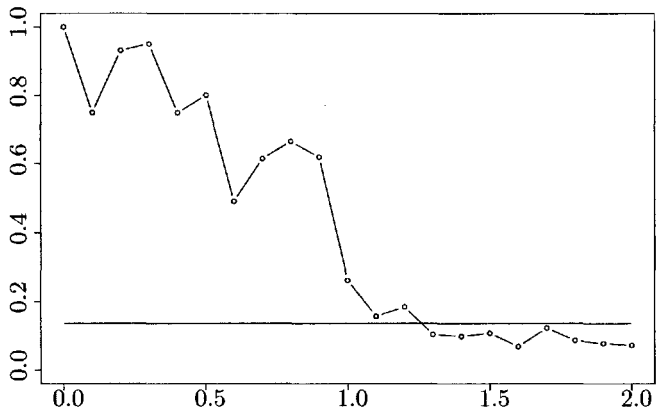


Figure 10: The average number of edges visited by the multi-level graph approach with BET divided by the average number of edges visited by the multi-level graph approach with BAP, applied to a `road` graph with 49 625 vertices. The abscissa denotes ε. The horizontal line indicates the same ratio comparing BET with the standard Dijkstra's algorithm.

	n	m	n^*	$\lvert S^*\rvert$	$\lvert S\rvert$	$\lvert S_{\text{opt}}\rvert$
`lrail`	1650	4574	645	22	38	16
	2239	6452	1830	68	107	55
	2348	8458	3458	154	132	58
	4553	15866	11447	605	412	164
	6848	19276	3169	183	399	164
`road`	19463	49692	479	165	196	177
	49625	125018	1060	336	410	382
	99529	252390	2591	773	941	855
	199739	501948	3754	2176	2636	2315
	299790	771418	8025	3399	4104	3628

Table 2: Applying the planar-separator algorithm to non-planar graphs (`lrail` and `road` graphs): number of vertices (n) and of edges (m); number of crossings yielding new vertices (n^*); separator size of the planarized graph ($|S^*|$), size of 'retranslated' separator ($|S|$) and optimized separator size ($|S_{\text{opt}}|$).

represents the size of the search space of the shortest-path algorithm and its running time. Figure 10 shows the number of visited edges with exact betweenness divided by the number of visited edges with approximated betweenness values for a `road` graph with 49 625 vertices. Hence, the smaller this ratio, the worse the multi-level graph approach works. Under $\varepsilon = 0$ we show the result for the exact betweenness values, so the ratio equals 1.

We observed that, with increasing ε, the multi-level approach with approximation gets worse only very slowly but with $\varepsilon \geq 1$, the ratio drops drastically, and the multi-level approach does not speed up the shortest-path search anymore. Due to this experiment, we chose $\varepsilon = 0.2$ for the following ones. This choice of ε leads to a drastically reduced preprocessing time of only about 0.5% of that needed to compute the exact betweenness values. Further tests with other `road` graphs confirm that the approximated betweenness values with $\varepsilon = 0.2$ are almost as good as the exact ones for the multi-level approach.

5.3.2 Planar-Separator Algorithm The main questions regarding application of our planar separator algorithm to non-planar graphs are: (i) how many crossings induce auxiliary vertices during the planarization and (ii) whether the transformation from the planarized auxiliary graph to the original graph does not enlarge the set of separator vertices too much. We planarized `rail` and `road` graphs, applied our planar-separator algorithm to obtain connected components of maximum size 500 (`rail`) and 1000 (`road`), and transformed the separator of the planarized graph back to the original one. Finally, a straight-forward heuristic was applied to remove redundant separator vertices.

The results are depicted in Table 2 and show that for both real-world graph classes the decomposition works very well: the resulting separators for the `road` graphs consist of roughly 1 percent of the total number of vertices (the number of crossings is between 1.8 and 2.7 percent). Concerning the `rail` graphs, the separators are slightly larger (between 1 and 3.6 percent), but these numbers are still small taking into account the large number of crossings (between 39 and 251 percent of the total number of vertices in the original `rail` graphs).

An alternative to the planar-separator algorithm is the graph partitioning tool MeTiS [14]. In [10] our experiments revealed that separators obtained by MeTiS are of almost the same quality (with respect to size and balance) as the ones obtained by our planar-separator algorithm. Preliminary experiments with MeTiS indicate that this observation translates also to the multi-level approach: MeTiS is indeed applicable, but not as good as PLS. We observed, for example, a speed-up of 13.3 with the `road` graph with 99 529 vertices (cf. the results for PLS below).

5.4 Multi-Level Graph Approach Concerning the multi-level graph approach the most important measurement is *speed-up*, i.e., the fraction of the number edges visited by Dijkstra's algorithm and the number

of edges visited by the corresponding multi-level graph approach. This parameter is independent of implementation and machine, and it turned out, for our experiments, to be the parameter related most closely to CPU time.

5.4.1 Selection Criteria In this section, we focus on the extended multi-level graph approach, the basic multi-level graph approach is discussed in Sections 5.4.2 and 5.4.3.

Small Graphs with One Additional Level. First, we investigate the numerous combinations of selection criterion, number of selected vertices or maximal component size, respectively, and graph type, thus providing a systematic overview of the parameters playing a crucial role in our multi-level graph model.

We took into account one graph of each type with about 1000 vertices: a component-induced graph (`ci1000`) with $N = 50$, $C = 63$, and $A = 3$ and a planar Delaunay graph (`del1000`) with 2500 edges as generated graphs; as real-world graphs, connected subgraphs of our road and railway networks with approximately 1000 vertices, `road1000` and `rail1000`.

With global decomposition, we chose for the number of selected vertices 3, 5, 8, and 10 percent of the number of vertices in the graph and with recursive decomposition, the maximum component size was set to 3, 5, 10, and 20 percent, which in some preliminary runs have turned out to be representative values.

Figure 11 shows in the form of standard boxplots the average (over 1000 runs) speed-up (cf. the definition of speed-up at the beginning of Section 5.4). The range of all absolute values for one selection criterion and one graph are shown; the horizontal line within a box denotes the median, the cross marks the mean value.

The criterion yielding maximum overall speed-up with both global selection and recursive decomposition is BET/BAP (there is practically no difference between these two criteria; cf. Section 5.3.1), which work pretty well for all graphs but `del1000`: best speed-up of almost 20 was achieved with `road1000`. The second-best criterion, of similar quality, turns out to be PLS, followed by RCH. Some of the other criteria work only slightly better than a selection by RND. In general, with recursive decomposition higher speed-ups are attainable: this holds true especially for the centrality index criteria.

For `del` graphs large sets of separator vertices are required to decompose the graph, which yields rather poor performance with the multi-level graph approach; in contrast, the approach works quite well for the real-world graph classes `road` and `rail`, provided a suitable selection criterion is used. For the `ci` graph BET/BAP

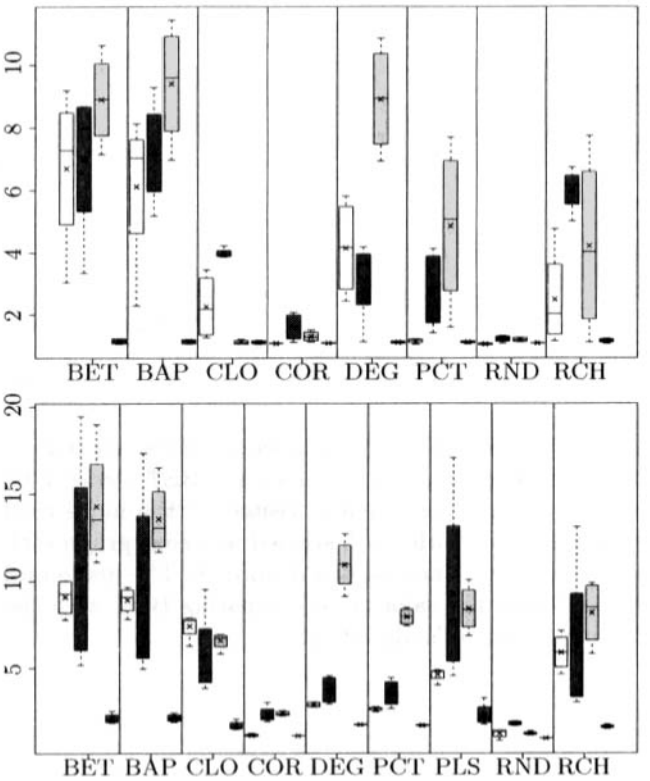


Figure 11: Average speed-up in terms of visited edges with global (top) and recursive (bottom) decomposition; graphs considered: `ci1000` (white), `road1000` (dark-gray), `rail1000` (light-gray), and `del1000` (brown).

with recursive decomposition yields the best speed-up of around 10. The vertex selection from the construction of the `ci` graph leads to practically the same speed-up factor. This shows once again the suitability of the BET/BAP criterion.

Larger Graphs. Concerning `road` graphs, we now consider the two best selection criteria identified in the previous paragraph for this graph class, namely BAP (not BET because of the high preprocessing time) and PLS, with recursive decomposition strategy. Figure 12 shows the speed-up for graphs with 10 000 to 100 000 vertices for different maximum component sizes: 1, 10, and 20 percent of the number of vertices in the graph.

For these graphs, 10 percent turned out always to be the best choice (with one exception). Interestingly, the PLS criterion is clearly superior to BAP, in contrast to the results with the smaller graphs. Another noticeable observation is that, with increasing size of the graphs, the speed-up increases—up to 31 with PLS and maximum component size 10 percent.

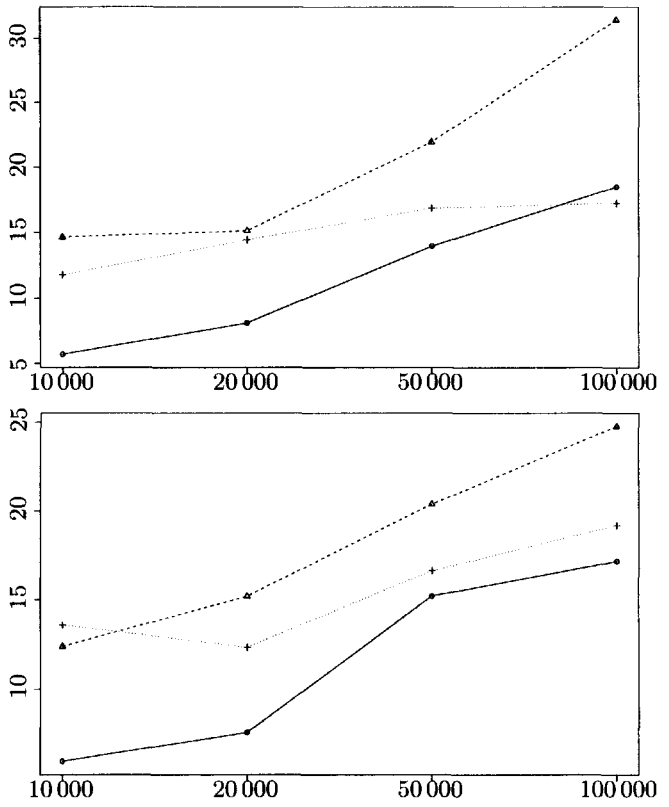


Figure 12: Speed-up with PLS (top) and BAP (bottom) criteria for larger `road` graphs, with maximum component size percentages of 1 (solid), 10 (dashed), and 20 (dotted) percent.

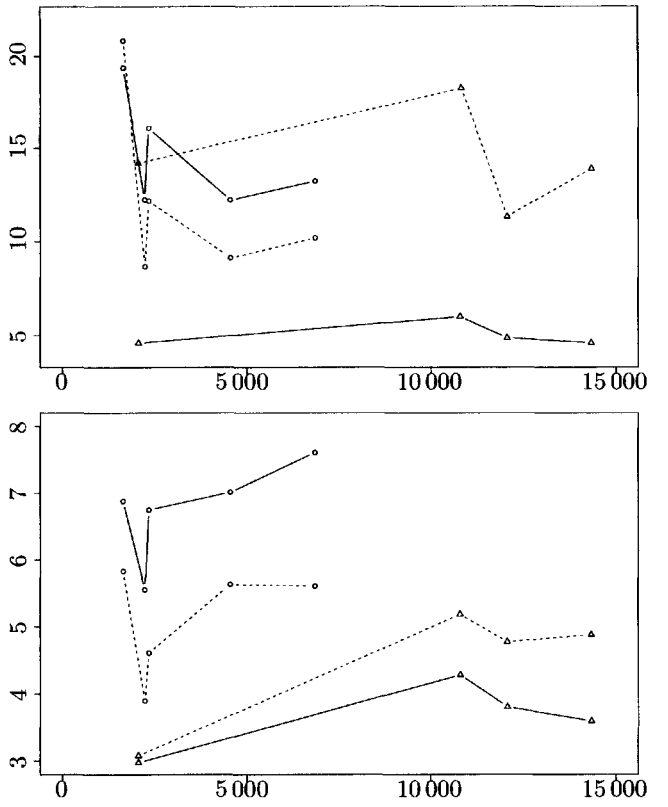


Figure 13: Speed-up with extended (top) and basic (bottom) multi-level graphs for `rail` graphs. The abscissa denotes the number of vertices in the graph; the maximum component percentages are 10 percent for the extended and 1 percent for the basic multi-level graphs. Criteria used: DEG (solid) and PLS (dashed); graph classes: `lrail` (circle) and `srail` (triangle).

Also for `rail` graphs, the PLS criterion is well-suited (cf. Figure 13). We compare it with the DEG criterion, which proved to work well for the `rail1000` graph. An interesting observation is the difference between long-distance and local-traffic networks: The DEG criterion works well (and somewhat better than PLS) only for long-distance traffic; for local traffic, the speed-up with DEG is not that distinct, while PLS still achieves speed-up factors of between 12 and 19 with extended multi-level graphs.

More detailed results for the larger `road` and `rail` graphs can be found in Table 3 and visualizations of some decompositions, in Figure 14.

5.4.2 Basic Multi-Level Graphs The difference of the basic multi-level graph approach compared to the extended one is that no upward and downward edges are maintained (cf. Section 2.2). Hence, one can expect less speed-up but at the same time less additional edges in the multi-level graph. This intuition is approved by Figure 15 depicting the quotient of speed-up (in terms of visited edges) and graph expansion (in terms of edges): the best quotient observed with the extended version is about 2.3, and roughly 12 with the basic version.

In fact, the graph expansion almost equals 1 for the basic version (cf. column $|L|/m$ in Table 3), so the lower diagram in Figure 15 shows basically the speed-up and can be compared to Figure 12 showing the speed-up with the extended version. The best speed-up with the basic version is obtained by a maximum component size of 1 percent, in contrast to the extended version, where the best speed-up is obtained with 10 percent. Hence, the basic version works better with smaller components, compared to the extended one.

Concluding the comparison of basic and extended multi-level graphs, the results shown in Figures 12,13, 15, and Table 3 confirm that the speed-up with the basic version is roughly half the speed-up achievable with the extended version, sometimes less than half. Concerning the additional number of edges, however, the basic version is clearly superior with a graph expansion of

	n	crit	$\lvert S\rvert$	$\lvert L\rvert/m$	$\frac{m'}{m}$	$\frac{\overline{ve}}{ve'}$	$\frac{ve}{ve'}$
srail	2070	PLS	55	5.2%	3.9	14.2	3.1
		DEG	198	19.2%	3.6	4.6	3.0
	10795	PLS	138	3.9%	6.2	18.3	5.2
		DEG	706	14.8%	4.0	6.0	4.3
	12070	PLS	163	6.2%	8.6	11.3	4.8
		DEG	1029	19.1%	4.1	4.9	3.8
	14335	PLS	194	5.6%	8.4	13.9	4.9
		DEG	1347	20.2%	3.8	4.6	3.6
lrail	1650	PLS	39	2.9%	2.9	20.9	5.8
		DEG	34	3.5%	2.8	19.4	6.9
	2239	PLS	72	8.0%	4.0	8.6	3.9
		DEG	74	5.9%	3.0	12.2	5.6
	2348	PLS	74	6.0%	3.5	12.2	4.6
		DEG	66	4.2%	2.6	16.1	6.7
	4553	PLS	181	7.8%	4.0	9.1	5.6
		DEG	144	6.2%	2.9	12.2	7.0
	6848	PLS	191	7.7%	6.0	10.2	5.6
		DEG	203	6.0%	3.3	13.2	7.6
road	9968	PLS	88	4.7%	7.7	14.6	4.9
		BAP	261	6.1%	5.2	12.4	5.2
	19463	PLS	118	4.4%	9.9	15.1	6.7
		BAP	278	4.3%	7.7	15.2	6.8
	49625	PLS	158	2.8%	12.1	21.9	11.1
		BAP	507	2.8%	8.0	20.4	11.2
	99529	PLS	206	2.0%	15.6	31.4	13.3
		BAP	913	2.7%	11.0	24.8	13.0

Table 3: Detailed results for larger `rail` and `road` graphs with recursive decomposition strategy and maximum component size of 10 percent of the total number of vertices n: selection criterion (crit), number of selected vertices ($|S|$), relative number of level edges ($|L|/m$; this equals the relative number of edges in the basic multi-level graph), graph expansion of extended multi-level graph (m'/m), and speed-up in terms of visited edges (extended variant: $\overline{ve}/ve'$; basic variant: ve/ve').

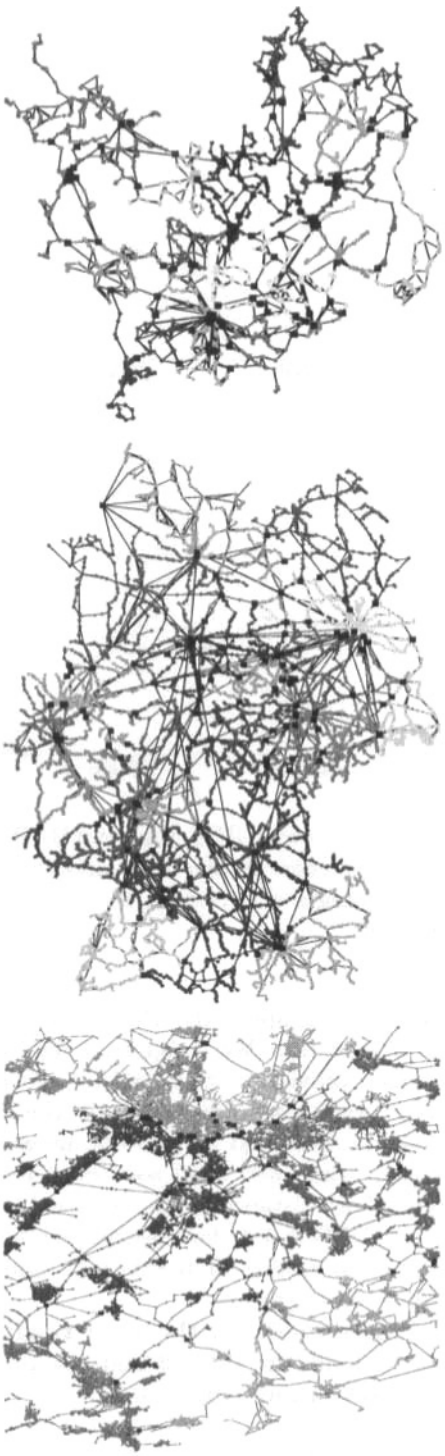


Figure 14: Recursive decompositions of `srail2070`, `lrail6848`, and `road19463` by PLS (cf. Table 3).

less than 1.2.

5.4.3 Multiple Levels We considered basic and extended multi-level graphs with two levels (cf. Figure 16). For the `road` graph with 99 529 vertices, with the extended version the speed-up could be increased from 31.4 with one level to well over 50 and the basic version could be improved from about 13 to 22. We observed a similar behavior also with other `road` graphs with up to 500 000 vertices. Not only the speed-up can be improved with multiple levels: While with one level the speed-up of 31.4 required a graph expansion of 15.6 (cf. the column m'/m in Table 3), the graph expansion with maximum component size 15 000 for level 1 and 300 for level 2 was only 6.1. The graph expansion for

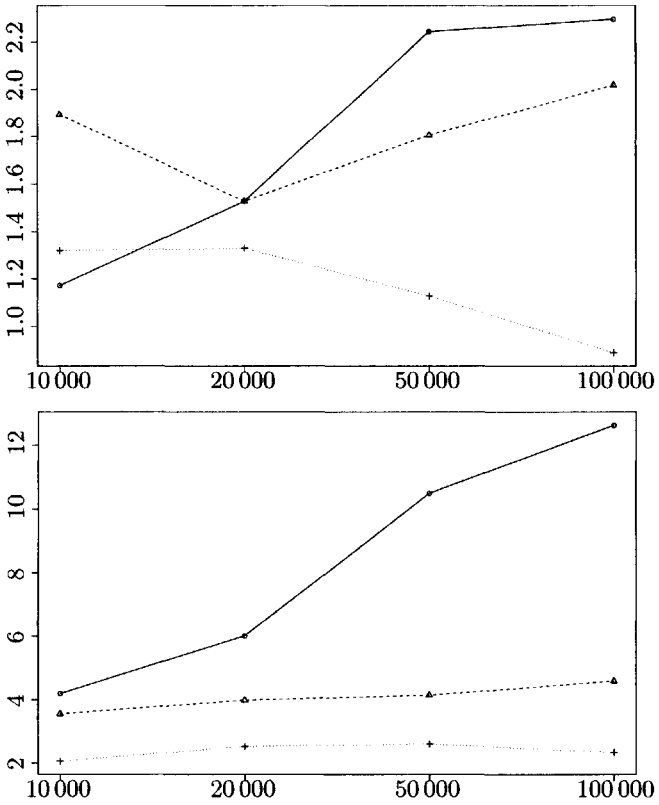


Figure 15: *Speed-up relative to graph expansion* with extended (top) and basic (bottom) multi-level graphs for `road` graphs and PLS criterion. Maximum component percentages: 1 (solid), 10 (dashed), and 20 (dotted) percent.

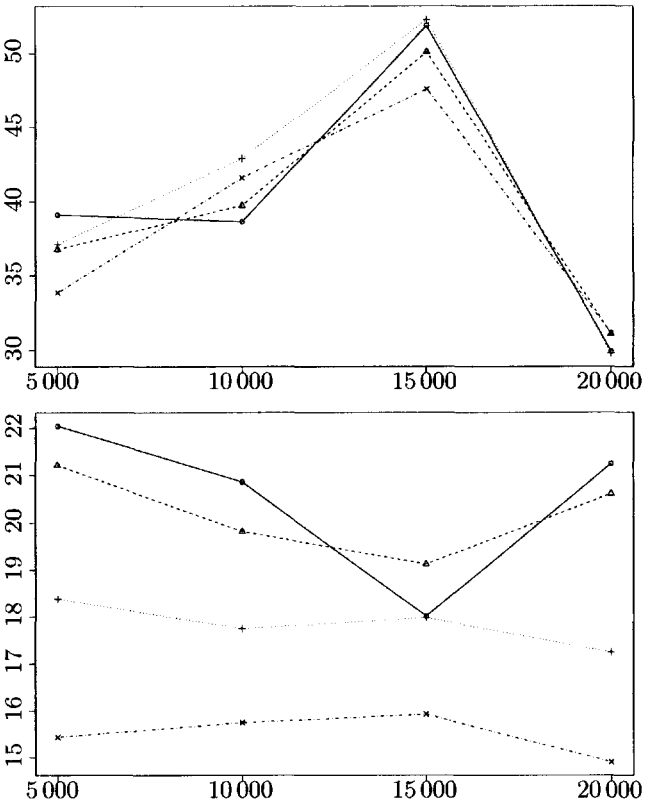


Figure 16: Speed-up with extended (top) and basic (bottom) multi-level graphs and *two levels* for the `road100000` graph and PLS criterion. The abscissa shows the maximum component size for level 1; maximum component sizes for level 2: 300 (solid), 500 (dashed), 1000 (dotted) and 1500 (dotted-dashed).

basic multi-level graphs with two levels is still negligible.

Further, we generated component-induced graphs with up to 100 000 vertices and considered belonging multi-level graphs with up to five additional levels. In this case, the vertex selections from the construction of the graphs have been used for the extended multi-level graph approach. These vertex sets are small and guarantee a regular graph decomposition. The main outcome from experiments with component-induced graphs of increasing size is that speed-up scales, up to a factor of approximately 1000 for a graph with roughly 100 000 vertices and 5 levels. These results confirm the intuition and the theoretical results from Section 4 that a regular (hierarchical) decomposition of the graph is essential for the success of the multi-level graph approach. For more details on the experiments with component-induced graphs we refer the reader to [9].

6 Conclusions

We defined minimal shortest-path overlay graphs, showed how to compute them and proved their minimality. Such graphs can be used to improve speed-up techniques for shortest-path computations that maintain precomputed shortest-path information between selected vertices in a hierarchical fashion. In particular, we investigated two variants of the multi-level graph approach for speeding up shortest-path computation.

In an experimental study, we explored several vertex selection criteria along with two general strategies to decompose the input graph. We used these selections to obtain multi-level overlay graphs and evaluated their performance when used for shortest-path computation.

The recursive strategy turned out to yield better performance than the global one. Concerning the criteria, planar separator and betweenness centrality clearly outperform the others. Experiments showed that betweenness can be approximated very efficiently and yields as good performance as the exact values. The degree criterion applied to `rail` graphs works well, especially for long-distance instances. For the other graph classes the degree criterion cannot compete with

planar separator and betweenness.

Comparing the two multi-level graph variants, the speed-up achieved with extended multi-level graphs is clearly superior to that achieved with basic ones. However, taking into account the sizes of the multi-level graphs, the ratio between speed-up and graph expansion is much more in favor of the basic variant.

Acknowledgments

The authors would like to thank Imen Borgi, Sebastian Knopp, and Andrea Schumm, for their support in parts of the implementation work.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[2] B. Bollobás, *Random Graphs*, London Academic Press, 1985.

[3] I. Borgi, J. Graf, M. Holzer, F. Schulz, and T. Willhalm, *A graph generator.* `http://i11www.ira.uka.de/resources/graphgenerator.php`.

[4] U. Brandes and T. Erlebach, eds., *Network Analysis*, vol. 3418 of LNCS, Springer, 2005.

[5] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[6] A. Goldberg and C. Harrelson, *Computing the shortest path: A* search meets graph theory*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), SIAM, 2005.

[7] A. Goldberg, H. Kaplan, and R. Werneck, *Reach for A*: Efficient point-to-point shortest path algorithms*, in Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX06), SIAM, 2006. To appear in the same volume.

[8] R. Gutman, *Reach-based routing: A new approach to shortest path algorithms optimized for road networks*, in Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM, 2004, pp. 100–111.

[9] M. Holzer, *Hierarchical speed-up techniques for shortest-path algorithms*, Master's thesis, Universität Konstanz, Fachbereich Informatik und Informationswissenschaft, 2003. `http://www.ub.uni-konstanz.de/kops/volltexte/2003/1038/`.

[10] M. Holzer, G. Prasinos, F. Schulz, D. Wagner, and C. Zaroliagis, *Engineering planar separator algorithms*, in Proceedings 13th European Symposium on Algorithms (ESA), vol. 3669 of LNCS, Springer, 2005, pp. 628–639.

[11] M. Holzer, F. Schulz, and T. Willhalm, *Combining speed-up techniques for shortest-path computations*, in Proceedings Third International Workshop on Experimental and Efficient Algorithms (WEA 2004), vol. 3059 of LNCS, Springer, 2004, pp. 269–284.

[12] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, *Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation*, IEEE Trans. Knowledge and Data Engineering, 10 (1998).

[13] S. Jung and S. Pramanik, *An efficient path computation model for hierarchically structured topographical road maps*, IEEE Trans. Knowledge and Data Engineering, 14 (2002).

[14] G. Karypis, *MeTiS.* `http://www-users.cs.umn.edu/~karypis/metis`.

[15] U. Lauther, *An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background*, in Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, vol. 22, IfGI prints, Institut für Geoinformatik, Münster, 2004, pp. 219–230.

[16] R. H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm, *Partitioning graphs to speed up Dijkstra's algorithm.*, in WEA, S. E. Nikoletseas, ed., vol. 3503 of Lecture Notes in Computer Science, Springer, 2005, pp. 189–202.

[17] S. Näher and K. Mehlhorn, *The LEDA Platform of Combinatorial and Geometric Computing*, Cambridge University Press, 1999. (http://www.algorithmic-solutions.com).

[18] P. Sanders and D. Schultes, *Highway hierarchies hasten exact shortest path queries*, in Proceedings 17th European Symposium on Algorithms (ESA), 2005.

[19] F. Schulz, *Timetable Information and Shortest Paths*, PhD thesis, Universität Karlsruhe (TH), Fakultät Informatik, 2005.

[20] F. Schulz, D. Wagner, and K. Weihe, *Dijkstra's algorithm on-line: An empirical case study from public railroad transport*, J. Experimental Algorithmics, 5 (2000).

[21] F. Schulz, D. Wagner, and C. Zaroliagis, *Using multi-level graphs for timetable information in railway systems*, in Proceedings 4th Workshop on Algorithm Engineering and Experiments (ALENEX), vol. 2409 of LNCS, Springer, 2002, pp. 43–59.

[22] D. Wagner and T. Willhalm, *Geometric speed-up techniques for finding shortest paths in large sparse graphs*, in Proc. 11th European Symposium on Algorithms (ESA), vol. 2832 of LNCS, Springer, 2003, pp. 776–787.

[23] T. Willhalm, *Engineering Shortest Paths and Layout Algorithms for Large Graphs*, PhD thesis, Universität Karlsruhe (TH), Fakultät Informatik, 2005.

[24] T. Willhalm and D. Wagner, *Shortest path speedup techniques*, in Algorithmic Methods for Railway Optimization, LNCS, Springer, 2006. To appear, see also [23].

Optimal Incremental Sorting *

Rodrigo Paredes † Gonzalo Navarro †

Abstract

Let A be a set of size m. Obtaining the first $k \leq m$ elements of A in ascending order can be done in optimal $O(m+k \log k)$ time. We present an algorithm (online on k) which incrementally gives the next smallest element of the set, so that the first k elements are obtained in optimal time for any k. We also give a practical version of the algorithm, with the same complexity on average, which performs better in practice than the best existing online algorithm. As a direct application, we use our technique to implement Kruskal's Minimum Spanning Tree algorithm, where our solution is competitive with the best current implementations. We finally show that our technique can be applied to several other problems, such as obtaining an interval of the sorted sequence and implementing heaps.

1 Introduction

There are cases where we need to obtain the smallest elements from a fixed set without knowing how many elements we will end up needing. Prominent examples are Kruskal's Minimum Spanning Tree (MST) algorithm [16] or ranking by Web search engines [1]. Given a graph, Kruskal's MST algorithm processes the edges one by one, from smallest to largest, until it forms the MST. At this point, remaining edges are not considered. Web search engines display a very small sorted subset of the most relevant documents among all those satisfying the query. Later, if the user wants more results, the search engine displays the next group of most relevant documents, and so on. In both cases, we could first sort the whole set and later return the desired objects, but obviously this is more work than necessary.

This problem can be called *Incremental Sorting*. It can be stated as follows: Given a set A of m numbers, output the elements of A from smallest to largest, so that the process can be stopped after k elements have been output, for any k that is unknown to the algorithm. Therefore, *Incremental Sorting* is the online version of an instance of the *Partial Sorting* problem: Given a set A of m numbers and an integer $k \leq m$, output the smallest k elements of A in ascending order.

In 1971, J. Chambers introduced the general notion of Partial Sorting [3]: given an array A of m numbers, and a fixed, sorted set of indices $I = i_0 < i_1 < \ldots < i_{k-1}$ of size $k \leq m$, arrange in place the elements of A so that $A[0, i_0 - 1] \leq A[i_0] \leq A[i_0 + 1, i_1 - 1] \leq A[i_1] \leq \ldots \leq A[i_{k-2}+1, i_{k-1}-1] \leq A[i_{k-1}] \leq A[i_{k-1}+1, m-1]$. This property is equivalent to the statement that $A[i]$ is the i-th order statistic of A for all $i \in I$.

We are interested in the particular case of finding the first k order statistics of a given set A of size $m > k$. This can be easily solved by first finding p, the k-th smallest element of A, using $O(m)$ time SELECT algorithm [2], and then collecting and sorting the elements smaller than p. We call this algorithm SELECTSORT. Its complexity, $O(m + k \log k)$, is optimal under the comparison model, as there are $m^{\underline{k}} = m!/(m-k)!$ possible answers and $\log(m^{\underline{k}}) = \Omega(m + k \log k)$.

A practical version of the above, QUICKSELECTSORT (**QSS**), uses QUICKSELECT [10] and QUICKSORT [11] as the selection and sorting algorithms, obtaining $O(m+k \log k)$ average complexity. Recently, it has been shown that selection and sorting can be interleaved. The result, PARTIALQUICKSORT (**PQS**), has the same average complexity but smaller constant terms [17].

To solve the online problem, we must select the smallest element, then the second one, and so on until the process finishes at some unknown value $k \in [0, m-1]$. One can do this by using SELECT to find each of the first k elements, which costs $O(km)$ overall. We can improve this by transforming A into a min-heap in time $O(m)$ [6], and then performing k extractions. This has $O(m + k \log m)$ worst-case complexity. Note that $m + k \log m = O(m + k \log k)$, as they can differ only if $k = o(m^c)$ for any $c > 0$, and then m dominates $k \log m$. However, according to experiments this scheme is much slower than the offline practical algorithm **PQS** if a classical heap is used.

In [22], P. Sanders proposes *sequence heaps*, a cache-aware priority queue to solve the online problem, which is optimized to insert and extract *all the elements* in the priority queue at small amortized cost. Even though the total CPU time used for this algorithm in the whole

*Supported in part by the Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.

†Center for Web Research, Dept. of Computer Science, University of Chile. Blanco Encalada 2120, Santiago, Chile. {raparede,gnavarro}@dcc.uchile.cl

process of inserting and extracting all the m elements is pretty close to the time of running QUICKSORT, our experiments show that this scheme is not so efficient when we want to sort just a small fraction of the set. Then the quest for a practical online algorithm for partial sorting is raised.

In this paper we present INCREMENTALSELECT (**IS**), which is yet another algorithm that solves the online problem in optimal $O(m + k \log k)$ time. But our main contribution is INCREMENTALQUICKSELECT (**IQS**), a practical variant of **IS**, which is $O(m + k \log k)$ time on average. Our experimental results show that **IQS** is almost as efficient as its offline version **PQS**, and is faster in practice than alternative solutions.

As an application, we show how to use our algorithm to boost the performance of Kruskal's MST algorithm [16]. Given a graph $G(V, E)$, we compute its MST in $O(|E| + |V| \log^2 |V|)$ average time, which is optimal in medium or high density graphs. In practice, by using **IQS** we obtain an efficient MST implementation. Our implementation is much faster than any other Kruskal's implementation we could program or find for any graph density. As a matter of fact, our Kruskal's version is faster than Prim's algorithm [20], even as optimized by B. Moret and H. Shapiro [18], and also competitive with the best alternative implementations we could find [14, 15].

We finally show that our algorithm can be used to solve other basic problems, such as obtaining an incremental segment of the sorted sequence, and implementing a priority queue. The algorithm can obviously be used to find the largest elements instead of the smallest.

2 Incremental sorting

In this section we describe **IQS** algorithm. At the end we show how it can be converted into its worst-case version **IS**. Essentially, to output the k smallest elements, **IQS** calls QUICKSELECT to find the smallest element on arrays $A[0, m-1]$, $A[1, m-1]$, ..., $A[k-1, m-1]$. This naturally leaves the k smallest elements sorted in $A[0, k-1]$. **IQS** avoids the $O(km)$ complexity by reusing the work among calls to QUICKSELECT.

Let us recall how QUICKSELECT works. Given an integer k, QUICKSELECT aims to find the k-th smallest element from a set A of m numbers. For this sake it chooses an object p (the pivot), and partitions A so that the elements lower than p are allocated to the left-side partition, and the others to the right side. After the partitioning, p is placed in its correct position i_p. So, if $i_p = k$, QUICKSELECT returns p and finishes. Otherwise, if $k < i_p$ it recursively processes the left partition, else the right partition (with a new $k \leftarrow k - i_p - 1$).

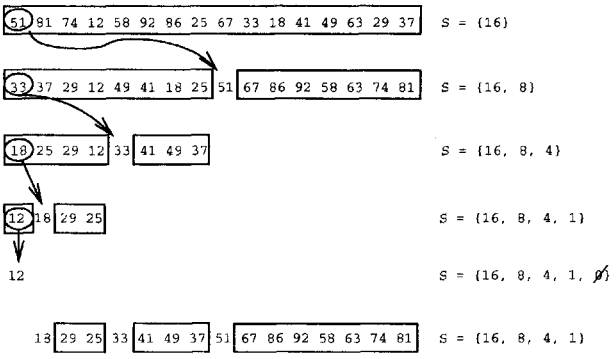


Figure 1: Example of how **IQS** finds the first element of an array. Each line corresponds to a new partition of a sub-array. Note that all the pivot positions are stored in stack S. In the example we use the first element as the pivot but it could be any other element. The bottom line shows the array with the three partitions generated by the first call to **IQS**, and the pivot positions stored in S.

Note that it is possible to reuse the work made by previous calls to QUICKSELECT. When we call QUICKSELECT on $A[1, m-1]$, a decreasing sequence of pivots has already been used to partially sort A since the previous invocation on $A[0, m-1]$. **IQS** manages this sequence of pivots to reuse previous work. Specifically, it uses a stack S of decreasing pivot positions that are relevant for the next calls to QUICKSELECT.

Fig. 1 shows how **IQS** searches for the smallest element of an array by using a stack initialized with a single value $m = 16$. To find the next minimum, we first check whether p, the top value in S, is the index of the element sought, in which case we pop and return it. Otherwise, because of previous partitionings, it holds that elements in $A[1, p-1]$ are smaller than all the rest, so we run QUICKSELECT on that portion of the array, pushing new pivots into S.

As can be seen in Fig. 1, the second minimum is the pivot on the top of S, so we pop and return it. Fig. 2 shows how **IQS** finds the third minimum using the pivot information stored in S. Notice that **IQS** *just works on the current first chunk* ($\{29, 25\}$), where it adds one pivot position to S and returns the third element in the next recursive call.

Now, retrieving the fourth and fifth elements is easy since both of them are pivots. Fig. 3 shows how **IQS** finds the sixth minimum. The current first chunk contains three elements: $\{41, 49, 37\}$. So, **IQS** obtains the next minimum by selecting 41 as pivot, partitioning its chunk and returning the element 37. The incremental sorting process will continue as long as needed, and it can be stopped in any time.

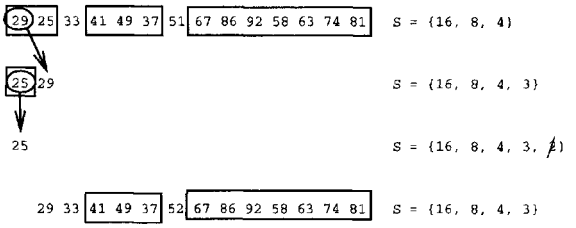


Figure 2: Example of how **IQS** finds the third element of the array. Since it starts with pivot information stored in S, it just works on the current first chunk ($\{29, 25\}$).

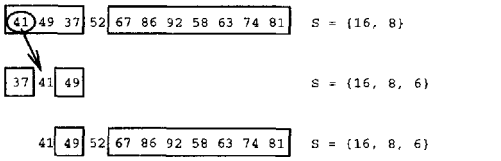


Figure 3: Example of how **IQS** finds the sixth element of an array. Since it starts with pivot information stored in S, it just works on the current first chunk ($\{41, 49, 37\}$). We omit the line where element 37 becomes a pivot and is popped from S.

The algorithm is given in Fig. 4. Stack S is initialized as $S = \{|A|\}$. **IQS** receives the set A, the index idx[1] of the element sought (that is, we seek the smallest element in $A[idx, m-1]$), and the current stack S (with former pivot positions). First it checks whether the top element of S is the desired index idx, in which case it pops idx and returns $A[idx]$. Otherwise it chooses a random pivot index $pidx$ from $[idx, S.\text{top}()-1]$. Pivot $A[pidx]$ is used to partition $A[idx, S.\text{top}()-1]$. After the partitioning, the pivot has reached its final position $pidx'$, which is pushed in S. Finally, a recursive invocation continues the work on the left hand of the partition.

Recall that **partition**(A, $A[pidx]$, i, j) rearranges $A[i, j]$ and returns the new position $pidx'$ of the original element in $A[pidx]$, so that, in the rearranged array, all the elements smaller/larger than $A[pidx']$ appear before/after $pidx'$. Thus, pivot $A[pidx']$ is left at the correct position it would have in the sorted array $A[i, j]$. The next lemma shows that it is correct to search for the minimum just within $A[i, S.\text{top}() - 1]$, from which the correctness of **IQS** immediately follows.

LEMMA 2.1. *After i minima have been obtained in $A[0, i-1]$, (1) the pivot indices in S are decreasing bottom to top, (2) for each pivot position $p \neq m$ in S, $A[p]$ is not smaller than any element in $A[i, p-1]$ and not larger than any element in $A[p+1, m-1]$.*

Proof. Initially this holds since $i = 0$ and $S = \{m\}$. Assume this is valid before pushing p, when p' was the top of the stack. Since the pivot was chosen from $A[i, p'-1]$ and left at some position $i \leq p \leq p'-1$ after partitioning, property (1) is guaranteed. As for property (2), after the partitioning it still holds for any pivot other than p, as the partitioning rearranged elements at the left of it. With respect to p, the partitioning ensures that elements smaller than p are left at $A[i, p-1]$, while larger elements are left at $A[p+1, p'-1]$. Since $A[p]$ was already not larger than elements in $A[p', m-1]$, the lemma holds. It obviously remains true after removing elements from S. ■

[1] Since we start counting array positions from 0, the place of the k-th element is $k-1$, so $idx = k-1$.

IQS (Set A, Int idx, Stack S)

// Precondition: $idx \leq S.top()$
1. **If** $idx = S.\text{top}()$ **Then** $S.\text{pop}()$, **Return** $A[idx]$
2. $pidx \leftarrow$ **random**$[idx, S.\text{top}()-1]$
3. $pidx' \leftarrow$ **partition**$(A, A[pidx], idx, S.\text{top}()-1)$
   // Invariant: $A[0] \leq \ldots \leq A[idx-1]$
   // $\leq A[idx, pidx'-1] \leq A[pidx']$
   // $\leq A[pidx'+1, S.\text{top}()-1] \leq A[S.\text{top}(), m-1]$
4. $S.\text{push}(pidx')$
5. **Return IQS**(A, idx, S)

Figure 4: INCREMENTALQUICKSELECT (**IQS**) algorithm. Stack S is initialized as $S \leftarrow \{|A|\}$. Both S and A are modified and rearranged during the algorithm. Note that the search range is limited to the array segment $A[idx, S.\text{top}()-1]$. Procedure **partition** returns the position of pivot $A[pidx]$ after the partition completes. Note that the tail recursion can be easily removed.

The worst-case complexity of **IQS** is $O(m^2)$, but it is easy to derive worst-case optimal **IS** from it. The only change is in line 2 of Fig. 4, where the random selection of the next pivot position must be changed to choosing the median of $A[idx, S.\text{top}() - 1]$, using the linear-time selection algorithm [2]. Section 3 analyzes the worst-case of **IS** and Section 4 considers the average-case of **IQS**, both of which are $O(m + k \log k)$.

3 IS worst-case complexity

In this section we analyze **IS**, which is not as efficient in practice as **IQS**, but has good worst-case performance. In particular, the analysis serves as a basis for the average-case analysis of **IQS** in Section 4. In **IS**, the

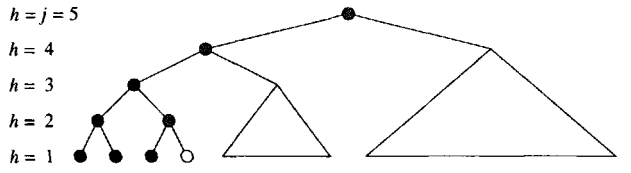


Figure 5: **IS** partition tree for incremental sorting when $k = 5$, $m = 31$, $j = 5$.

partition is perfectly balanced since each pivot position is chosen as the median of its array segment.

In this analysis we assume that m is of the form $2^j - 1$. We recall that array indices are in the range $[0, m-1]$. Fig. 5 illustrates the incremental sorting process when $k = 5$ in a perfect balanced tree of $m = 31$ elements, $j = 5$. Black nodes are the elements already reported, grey nodes are the pivots that remain in stack S, and white nodes and trees are the other elements of A.

The pivot at the tree root is the first to be obtained (the median of A), at cost linear in m (both to obtain the median and to partition the array). The two root children are the medians of $A[0, \frac{m-3}{2}]$ and $A[\frac{m+1}{2}, m-1]$. Obtaining those pivots and partitioning with them will cost time linear in $m/2$. The left child of the root will actually be the second pivot to be processed. The right child, on the other hand, will be processed only if $k > \frac{m-1}{2}$, that is, at the moment we ask **IS** to output the $\frac{m+1}{2}$-th minimum. In general, processing the pivots at level h will cost $O(2^h)$, but only some of these will be required for a given k. The maximum level is $j = \log_2(m+1)$.

It is not hard to see that, in order to obtain the k smallest elements of A, we will require $\lceil \frac{k}{2^h} \rceil$ pivots of level h. Adding up their processing cost we get Eq. (3.1), where we split the sum into the cases $\lceil \frac{k}{2^h} \rceil > 1$ and $\lceil \frac{k}{2^h} \rceil = 1$. Only then, in Eq. (3.3), we use $k + 2^h$ to bound the terms of the first sum, and redistribute terms to obtain that **IS** is $O(m + k \log k)$ worst-case time. The extra space used by **IS** is $O(\log m)$ pivot positions.

$$(3.1)\quad T(m,k) = \sum_{h=1}^{\log_2(m+1)} \left\lceil \frac{k}{2^h} \right\rceil 2^h$$

$$(3.2)\quad = \sum_{h=1}^{\lfloor \log_2 k \rfloor} \left\lceil \frac{k}{2^h} \right\rceil 2^h + \sum_{h=\lfloor \log_2 k \rfloor + 1}^{\log_2(m+1)} 2^h$$

$$(3.3)\quad \leq \sum_{h=1}^{\lfloor \log_2 k \rfloor} k + \sum_{h=1}^{\log_2(m+1)} 2^h$$

$$(3.4)\quad T(m,k) = k\lfloor \log_2 k \rfloor + 2m + 1$$

4 IQS average-case complexity

In this case the final pivot position p after the partitioning of $A[0, m-1]$ distributes uniformly in $[0, m-1]$. Consider again Fig. 5, where the root is not anymore the middle of A but a random position. We call $T(m,k)$ the average number of key comparisons needed to obtain the k smallest elements of $A[0, m-1]$. After the $m-1$ comparisons used in the partitioning, there are three cases depending on p: (1) $k \leq p$, in which case the right subtree will never be expanded, and the total extra cost will be $T(p,k)$ to solve $A[0, p-1]$; (2) $k = p+1$, in which case the left subtree will be fully expanded to obtain its p elements at cost $T(p,p)$; and (3) $k > p+1$, in which case we pay $T(p,p)$ on the left subtree, whereas the right subtree, of size $m-1-p$, will be expanded so as to obtain the remaining $k-p-1$ elements.

Thus **IQS** average cost follows Eq. (4.5), which is rearranged as Eq. (4.6). It is easy to check that this is exactly the same as Eq. (3.1) in [17], which shows that **IQS** makes exactly the same number of comparisons of the offline version, **PQS**. This is $2m + 2(m+1)H_m - 2(m+3-k)H_{m+1-k} - 6k + 6$. That analysis [17] is rather sophisticated, resorting to bivariate generating functions. In which follows we give a simple development arriving at a solution of the form $O(m + k \log k)$.

$$(4.5)\quad T(m,k) = m - 1 + \frac{1}{m}\left(\sum_{p=k}^{m-1} T(p,k) + T(k-1,k-1) + \sum_{p=0}^{k-2} \Big(T(p,p) + T(m-1-p, k-p-1) \Big) \right)$$

$$(4.6)\quad = m - 1 + \frac{1}{m}\left(\sum_{p=0}^{k-1} T(p,p) + \sum_{p=0}^{k-2} T(m-1-p, k-p-1) + \sum_{p=k}^{m-1} T(p,k) \right)$$

Eq. (4.6) simplifies to Eq. (4.7) by noticing that $T(p,p)$ behaves exactly like QUICKSORT, $2(p+1)H_p - 4p$ [9] (this can also be seen by writing down $T(p) = T(p,p)$ and noting that the very same QUICKSORT recurrence is obtained), so that $\sum_{p=0}^{k-1} T(p,p) = k(k+1)H_k - \frac{k}{2}(5k-1)$. We also write p' for $k-p-1$ and rewrite the second sum as $\sum_{p'=1}^{k-1} T(m-k+p', p')$.

$$(4.7)\quad T(m,k) = m - 1 + \frac{1}{m}\left(k(k+1)H_k - \frac{k}{2}(5k-1) + \sum_{p=1}^{k-1} T(m-k+p, p) + \sum_{p=k}^{m-1} T(p,k) \right)$$

We make some pessimistic simplifications now. The first sum governs the dependence on k of the recurrence. To avoid such dependence, we bound the second argument to k and the first to m, as $T(m,k)$ is monotonic on both its arguments. The new recurrence, Eq. (4.8), depends only on parameter m and treats k as a constant.

$$T(m) = m - 1 + \frac{1}{m}\left(k(k+1)H_k - \frac{k}{2}(5k-1) + (k-1)T(m) + \sum_{p=k}^{m-1} T(p)\right) \tag{4.8}$$

We subtract $mT(m) - (m-1)T(m-1)$ using Eq. (4.8), to obtain Eq. (4.9) and Eq. (4.10). Since $T(k)$ is actually $T(k,k)$, we use again QUICKSORT formula in Eq. (4.11). We bound the first part by $2m + 2kH_{m-k}$ and the second part by $2kH_k$ to obtain Eq. (4.12).

$$T(m) = 2\frac{m-1}{m-k+1} + T(m-1) \tag{4.9}$$

$$= 2\sum_{i=k+1}^{m}\left(1 + \frac{k-2}{i-k+1}\right) + T(k) \tag{4.10}$$

$$= 2(m-k) + 2(k-2)(H_{m-k+1} - 1) + (2(k+1)H_k - 4k) \tag{4.11}$$

$$< 2(m + kH_{m-k} + kH_k) \tag{4.12}$$

This result does not yet look good enough, but we plug it again into Eq. (4.7). In this case, we bound the sum $\sum_{p=1}^{k-1} T(m-k+p,p)$ with $\sum_{p=1}^{k-1} 2(m-k+p+pH_{m-k}+pH_p) = (k-1)\left(2m + k\left(H_{m-k} + H_k - \frac{3}{2}\right)\right)$. Upper bounding again and multiplying by m we get a new recurrence in Eq. (4.13). Note that this recurrence only depends on m.

$$mT(m) = m(m-1) + k(k+1)H_k - \frac{k}{2}(5k-1) + (k-1)\left(2m + k\left(H_{m-k} + H_k - \frac{3}{2}\right)\right) + \sum_{p=k}^{m-1} T(p) \tag{4.13}$$

Subtracting again $mT(m) - (m-1)T(m-1)$ we get Eq. (4.14). Noting that $\frac{(k-1)k}{(m-k)m} = (k-1)\left(\frac{1}{m-k} - \frac{1}{m}\right)$, we get Eq. (4.15), which is solved in Eq. (4.16).

$$T(m) = 2\frac{m+k-2}{m} + \frac{(k-1)k}{(m-k)m} + T(m-1) \tag{4.14}$$

$$< \sum_{i=k+1}^{m}\left(2 + 2\frac{k-2}{i} + (k-1)\left(\frac{1}{i-k} - \frac{1}{i}\right)\right) + T(k) \tag{4.15}$$

$$= 2(m-k) + 2(k-2)(H_m - H_k) + (k-1)(H_{m-k} - H_m + H_k) + (2(k+1)H_k - 4k) \tag{4.16}$$

Note that $H_m - H_k < \frac{m-k}{k+1}$ and thus $(k-2)(H_m - H_k) < m-k$. Also, $H_{m-k} \leq H_m$, so collecting terms we obtain Eq. (4.17). Therefore, **IQS** is also $O(m + k\log k)$ in the average-case when we choose pivots at random.

$$T(m,k) < 4m - 8k + (3k+1)H_k < 4m + 3kH_k \tag{4.17}$$

As a final remark, note that when we use **QSS** a portion of the QUICKSORT partitioning work repeats the work made in the previous QUICKSELECT calling. Fig. 6 illustrates this, showing that upon finding the k-th element, the QUICKSELECT stage has produced partitions A_1 and A_2, however the QUICKSORT that follows processes the left partition as a whole ($[A_1 p_1 A_2]$), ignoring the previous partitioning work done over it. On the other hand, **IQS** sorts the left segment by processing each partition independently, because it knows their limits (as they are stored in the stack S). This also applies to **PQS** and it explains the finding of C. Martínez that **PQS**, and thus **IQS**, makes $2k - 4H_k + 2$ less comparisons than **QSS** [17].

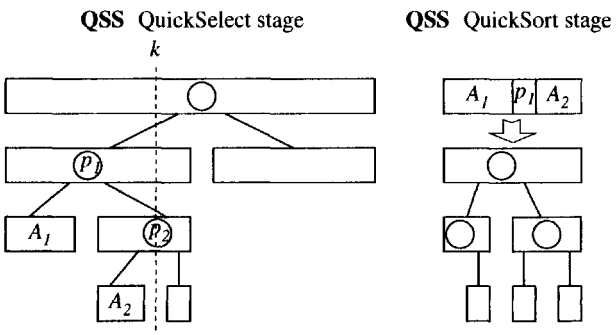


Figure 6: Partition work performed by **QSS**. First, **QSS** uses QUICKSELECT for finding the k-th element (left). Then it uses QUICKSORT on the left array segment as a whole ($[A_1\ p_1\ A_2]$) neglecting the previous partitioning work (right).

5 IQS and the minimum spanning tree

In this section we explore a practical application of **IQS**: improving the performance of Kruskal's Minimum Spanning Tree (MST) algorithm.

Let us recall the MST problem. Let $G(V, E)$ be a connected graph with a nonnegative cost function $d(e)$ assigned to its edges $e \in E$. A minimum spanning tree mst of the graph $G(V, E)$ is a tree composed of edges of E that connect all the vertices of V at the lowest total cost $\sum_{e \in mst} d(e)$. Note that, given a graph, its MST is not necessarily unique.

Let $n = |V|$, $m = |E|$. The most popular algorithms to solve the MST problem are Kruskal's [16] and Prim's [20], whose basic versions have complexity $O(m \log m)$ and $O(n^2)$, respectively. We call these basic versions **Kruskal1** and **Prim1**, respectively. In sparse graphs, with $|E| = O(n)$, it is recommended to use **Kruskal1**, whereas in dense graphs, with $|E| = O(n^2)$, **Prim1** is recommended [5, 25]. Alternatively, Prim can be implemented using Fibonacci Heaps [8] to obtain $O(m + n \log n)$ complexity.

There are other MST algorithms compiled by Tarjan [23]. Recently, B. Chazelle [4] gave an $O(m\alpha(m, n))$ algorithm, where $\alpha \in \omega(1)$ is the very slowly-growing inverse Ackermann's function. Later, S. Pettie and V. Ramachandran [19] proposed an algorithm that runs in optimal time $O(T^*(m, n))$, where $T^*(m, n)$ is the minimum number of edge-weight comparisons needed to determine the MST of any graph $G(V, E)$ with m edges and n vertices. The best known upper bound of this algorithm is also $O(m\alpha(m, n))$. These algorithms almost reach the lower bound $\Omega(m)$, yet they are so complicated that their interest is mainly theoretical. Furthermore, there is a randomized algorithm [13] that finds the MST in $O(m)$ time with high probability in the restricted RAM model, but it is also considered impractical as it is complicated to implement and the $O(m)$ complexity hides a big constant factor.

Experimental studies on MST are given in [18, 14, 15]. In [18], they compare several versions of Kruskal's, Prim's and Tarjan's algorithms, concluding that the best in practice (albeit not in theory) is Prim using pairing heaps [7]. We call this algorithm **Prim2**. Their experiments show that neither Cheriton and Tarjan's [23] nor Fredman and Tarjan's algorithm [8] ever approach the speed of **Prim2**. On the other hand, they show that **Kruskal1** can run very fast when it uses an array of edges that can be overwritten during sorting, instead of an adjacency list. Moreover, they show that it is possible to use heaps to improve Kruskal's algorithm. They call this variant *Kruskal's with demand sorting*, and we will refer to it as **Kruskal2**. The result is a rather efficient MST version with complexity $O(m + k \log m)$, being $k \leq m$ the number of edges reviewed by Kruskal technique.

```
Kruskal1 (Graph G(V, E))
1.  UnionFind C ← {v ∈ V, {v}}
    // the set of all connected components
2.  mst ← ∅ // the growing minimum spanning tree
3.  ascendingSort(E), k ← 0
4.  While |C| > 1 Do
        // select an edge in ascending order
5.      (e = {u, v}) ← E[k], k ← k + 1
6.      If C.find(u) ≠ C.find(v) Then
7.          mst ← mst ∪ {e}, C.union(u, v)
8.  Return mst
```

Figure 7: The basic version of Kruskal's MST algorithm (**Kruskal1**). To carry out the heap-based optimization (**Kruskal2**), we change line 3 to **heapify**(E) and line 5 to $(e = \{u, v\}) \leftarrow E$.**extractMin**().

In [14, 15], they give an algorithm whose complexity is $O(m + n \log n)$. It generates a subgraph G' by selecting $\sqrt{mn}$ edges from G at random. Then, it builds the minimum spanning forest T' of G'. Then, it filters each edge $e \in E$ using the cycle property: discard e if it is the heaviest edge on a cycle in $T' \cup \{e\}$. Finally, it builds the MST of the subgraph that contains the edges of T' and the edges that were not filtered out. We call this algorithm **iMax**.

5.1 Kruskal's MST algorithm. Kruskal's algorithm starts with n single-node components, and it merges them until it produces a sole connected component. To do this, **Kruskal1** begins by setting the mst to $(V, \emptyset)$, that is, n single-node trees. Later, in each iteration, it adds to the mst the cheapest edge of E that does not produce a cycle on the mst, that is, it only adds edges whose vertices belong to different connected components. Once the edge is added, both components are merged. The process ends when the mst becomes a single connected component. At this point the mst is a minimum spanning tree of $G(V, E)$.

To manage the component operations, we use the *Union-Find* data structure C with path compression, see [5, 25] for a comprehensive explanation. Given two vertices u and v, we use the **find**(u) operation to compute which component u belongs to, and use **union**(u, v) to merge the components of u and v. The amortized cost of **find**(u) is $O(\alpha(m, n))$ and the cost of **union**(u, v) is constant.

Fig. 7 depicts the basic Kruskal's MST algorithm.

We need $O(n)$ time to initialize both C and mst, and $O(m \log m)$ time to sort the edge set E. Then we make at most $m\,O(\alpha(m,n))$-time iterations of the **While** loop. Therefore, **Kruskal1** complexity is $O(m \log m)$.

Assuming we are using either full or random graphs whose cost edges are assigned at random independently of the rest (using any continuous distribution), the subgraph composed by V with the edges reviewed by the algorithm is a random graph [12]. Therefore, based on [12, pp. 349], we expect to finish the MST construction upon reviewing $\frac{1}{2}n \ln n + \frac{1}{2}\gamma n + \frac{1}{4} + O(\frac{1}{n})$ edges, which can be much lower than m. So, it is not necessary to sort the whole set E, but it is enough to select and extract one by one the minimum-cost edges until we complete the MST. The common solution of this type consists in min-heapifying the set E, and later performing as many min-extractions of the lowest cost edge as needed (in [18], they do this in their Kruskal's demand sorting version). This is an implementation of *Incremental Sort*. For this sake we modify lines 3 and 5 of Fig. 7: line 3 changes to **heapify**(E) and line 5 to $(e = \{u, v\}) \leftarrow E$.**extractMin**().

Kruskal2 needs $O(n)$ time to initialize both C and mst, and $O(m)$ time to heapify E. We expect to review $\frac{1}{2}n \ln n + O(n)$ edges in the **While** loop. For each of these edges, we use $O(\log m)$ time to select and extract the minimum element of the heap, and $O(\alpha(m,n))$ time to perform the **union** and **find** operations. Therefore, **Kruskal2** average complexity is $O(m + n \log n \log m)$. As $n - 1 \leq m \leq n^2$, **Kruskal2** average complexity can also be written as $O(m + n \log^2 n)$.

5.2 IQS-based implementation of the Kruskal's MST algorithm. We can use **IQS** in order to incrementally sort E. After initializing C and mst, we create the stack S, and push m into S. Later, inside the **While** loop, we call **IQS** in order to obtain the k-th edge of E incrementally. Fig. 8 shows our Kruskal's MST variant, that we call **Kruskal3**. Note that the expected number of pivoting edges that we store in S is $O(\log m)$.

We need $O(n)$ time to initialize both C and mst, and constant time for S. We expect to review $\frac{1}{2}n \ln n + O(n)$ edges within the **While** loop, thus we need $O(m + n \log^2 n)$ overall expected time for IQS and $O(n\alpha(m,n) \log n)$ time for all the **union** and **find** operations. Therefore, **Kruskal3** average complexity is $O(m + n \log^2 n)$, just as **Kruskal2**.

6 Experimental results

We ran two experimental series with **IQS**. In the first series we compare **IQS** against other alternatives. In the second we evaluate our **Kruskal3** algorithm. The experiments were run on an Intel Pentium 4 of 3 GHz, 4 GB of RAM and local disk. For each experimental condition we show averages computed over 50 repetitions, for all competing implementations. The weighted least square fittings were performed with R [21]. In order to illustrate the precision of our fittings, we also show the average percent error of residuals with respect to real values ($|\frac{y-\hat{y}}{y}|100\%$) for fittings belonging around to the 45% of the largest values[2].

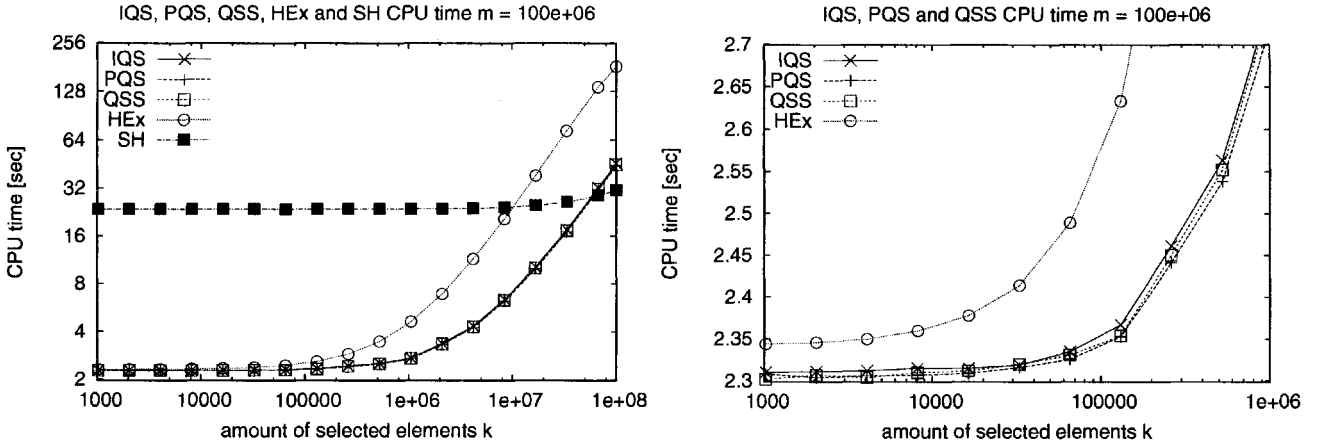
Kruskal3 (Graph $G(V, E)$)
1. UnionFind $C \leftarrow \{v \in V, \{v\}\}$
   // the set of all connected components
2. $mst \leftarrow \emptyset$ // the growing minimum spanning tree
3. Stack S, S.**push**($|E|$), $k \leftarrow 0$
4. **While** $|C| > 1$ **Do**
   // select the lowest edge incrementally
5. $(e = \{u, v\}) \leftarrow$ **IQS**(E, k, S), $k \leftarrow k + 1$
6. **If** C.**find**(u) $\neq C$.**find**(v) **Then**
7. $mst \leftarrow mst \cup \{e\}$, C.**union**(u, v)
8. **Return** mst

Figure 8: Our Kruskal's MST variant (**Kruskal3**). Note the changes in lines 3 and 5 with respect to **Kruskal1**.

6.1 Evaluating IQS. We compared **IQS** against **PQS**, **QSS**, and two online approach: the first based on classical heaps [26] (called **HEx**), and the second based on sequence heaps [22] (called **SH**, obtained from `www.mpi-inf.mpg.de/~sanders/programs/spq/`). The idea is to verify that **IQS** is in practice a competitive algorithm for the *Partial Sorting* problem for finding the smallest elements in ascending order. For this sake, we use random permutations in $[0, m-1]$, for $m \in [10^5, 10^8]$, and we select the k first elements with $k = 2^j < m$, for $j \geq 10$. The selection is incremental for **IQS**, **HEx** and **SH**, and in one shot for **PQS** and **QSS**. We measure CPU time and the number of key comparisons, except for **SH** where we only measure CPU time.

As it turned out to be more efficient, we implement **HEx** by using the *bottom up heuristic* [24] for **extractMin**: when the minimum is extracted, we lift up ele-

[2] Our fittings are too pessimistic for small permutations or edge sets, so we intend to show that they are asymptotically good. In the first series we compute the percent error for permutations of length $m \in [10^7, 10^8]$ for all the k values, approximately 45.4% of the measures. In the second series we compute the percent error for edge density in [16%, 100%] for all values of $|V|$, approximately 44.4% of the measures. Both turn out to be around 45%.



(a) CPU time for the five algorithms.

(b) Detail of CPU time for **IQS**, **PQS**, **QSS** and **HEx**.

Figure 9: Performance comparison between **IQS**, **PQS**, **QSS**, **HEx** and **SH** as a function of the amount of searched elements k for different values of set size m. Note the logscales in the plots.

	Fitting	Error
$\mathbf{PQS}_{cpu}$	$25.79m + 16.87k \log_2 k$	6.77%
$\mathbf{PQS}_{cmp}$	$2.138m + 1.232k \log_2 k$	5.54%
$\mathbf{IQS}_{cpu}$	$25.81m + 17.44k \log_2 k$	6.82%
$\mathbf{PQS}_{cmp}$	$2.138m + 1.232k \log_2 k$	5.54%
$\mathbf{QSS}_{cpu}$	$25.82m + 17.20k \log_2 k$	6.81%
$\mathbf{QSS}_{cmp}$	$2.140m + 1.292k \log_2 k$	5.53%
$\mathbf{HEx}_{cpu}$	$23.85m + 67.89k \log_2 m$	6.11%
$\mathbf{HEx}_{cmp}$	$1.904m + 0.967k \log_2 m$	1.20%
$\mathbf{SH}_{cpu}$	$9.165m \log_2 m + 66.16k$	2.20%

Table 1: **IQS**, **PQS**, **QSS**, **HEx** and **SH** weighted least square fittings. For **SH** we only compute the CPU time fitting. CPU time is measured in nanoseconds.

ments on a min-path from the root to a leaf in the bottom level. Then, we place the rightmost element (the last of the heap) into the free leaf, and bubble it up to restore the min-heap condition. Using this heuristic we perform only $\log_2 m + O(1)$ key comparisons for each extraction on average (saving up to half of the comparisons used by a straightforward implementation taken from textbooks [5, 25]).

We summarize the experimental results in Figs. 9, 10 and 11, and Table 1. As can be seen from the least square fittings of Table 1, **IQS** CPU time performance is 2.99% slower than that of its offline version **PQS**. The number of key comparisons is exactly the same, as we expected from Section 4. This is an extremely small price for permitting incremental sorting without knowing in advance how many elements we wish to retrieve, and shows that **IQS** is practical. Moreover, as the pivots in the stack help us reuse the partitioning work, our online **IQS** is 1.33% slower in CPU time and uses 4.20% less key comparisons than the offline **QSS**.

On the other hand, we obtain large improvements with respect to online alternatives. According to the insertion and deletion strategy of sequence heaps, we compute its CPU time least square fitting by noticing that we can split the experiment in two stages. The first inserts m random elements into the priority queue, and the second extracts the smallest k elements from it. Then, we obtain a simplified $O(m \log m + k)$ complexity model that shows that most of the work performed by **SH** comes from the insertion process. Note that, if we want a small fraction of the sorted sequence, we prefer to pay a lower insertion and a higher extraction cost (just like **IQS**) than to perform most of the work in the insertions and a little in the extractions. Finally, even when the online **HEx** with the bottom-up heuristic uses at most $2m$ key comparisons to heapify the array, and $\log m + O(1)$ key comparisons on average to extract elements, numerous cache faults slow down its performance. As a matter of fact, **HEx** takes 3.88 times more CPU time and 18.76% less key comparisons than **IQS**.

Fig. 9(a) compares the five algorithms. As can be seen, even though **SH** is the best implementation to sort the whole set, it is not so efficient to sort just a small fraction of it. We suspect that this is because **SH** is cleverly optimized for the whole process of insertions and extractions, but not for a small fraction. As we have already said, its CPU time depends only mildly on the number of extracted elements, as most of the work performed by **SH** comes from the insertion process.

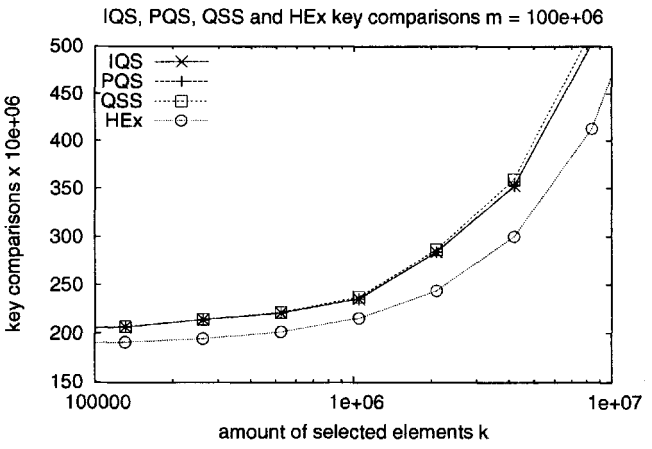


Figure 10: Detail of key comparisons for **IQS**, **PQS**, **QSS** and **HEx** for $m = 10^8$ varying k. Note the logscale in the plot.

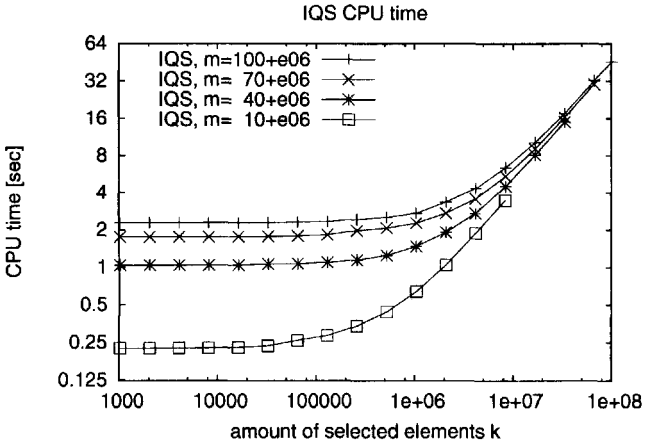


Figure 11: **IQS** CPU time as a function of k and m. Note the logscale in the plot.

HEx has the second worst CPU performance for $k \leq 0.1m$ and the worst for $k \in [0.1m, m]$, despite that it makes less key comparisons than others when extracting few objects, see Fig. 10. The reason is that classic heaps (even with the bottom-up heuristic) do not take advantage of the cache because of their poor locality of reference, which slows down the performance of **HEx**. Fig. 9(b) shows that **PQS** is the fastest algorithm when sorting a small fraction of the set, but **IQS** and **QSS** have rather similar behavior, and **HEx** follow them by far, confirming the results of our fittings of Table 1.

Finally, Fig. 11 shows that, as k grows, **IQS** behavior changes as follows. When $k \leq 0.01m$, there is no difference in the first k element incremental sorting, namely, the term m dominates the cost. When $0.01m < k \leq 0.04m$, there is a slight increase of both CPU time and key comparisons, that is, both terms m and $k \log k$ take part in the cost. Finally, when $0.04m < k \leq m$, term $k \log k$ leads the cost.

	Fitting	Error
Sorted edges	$0.532n \ln n$	1.69%
$\mathbf{Kruskal1}_{cpu}$	$12.85m \log_2 m$	1.74%
$\mathbf{Kruskal2}_{cpu}$	$39.99m + 46.30n \log_2 n \log_2 m$	6.96%
$\mathbf{Kruskal3}_{cpu}$	$19.26m + 10.93n \log_2^2 n$	4.17%

Table 2: Weighted least square fittings for Kruskal's MST versions ($n = |V|$, $m = |E|$). CPU time is measured in nanoseconds.

6.2 Evaluating Kruskal3. We now evaluate how **IQS** improves Kruskal's MST algorithm, so we compare its three versions against state of the art alternatives. We use synthetic graphs with edges chosen at random, and with edge costs uniformly distributed in $[0, 1]$. We consider graphs with $|V| \in [2000, 26000]$, and graph edge densities $\rho \in [0.5\%, 100\%]$, where $\rho = \frac{2m}{n(n-1)} 100\%$.

According to the experiments of Section 6.1, we preferred classical heaps using the bottom-up heuristic (**HEx**) over sequence heaps (**SE**) to implement **Kruskal2** in these experiments (as we expect to extract $\frac{1}{2} n \ln n + O(n) \ll m$ edges). We also show results for **iMax** and **Prim2** implementations from [14], as well as **Prim1** in complete graphs[3]. We obtained both the **iMax** and the optimized **Prim2** implementations from `www.mpi-sb.mpg.de/~sanders/dfg/iMax.tgz`.

For Kruskal's versions we measure the CPU time, memory requirements and the size of the edge subset reviewed during the MST construction. Note that those edges are the ones we incrementally sort. As the three versions run over the same graphs, they review the same subset of edges and use almost the same memory. For **Prim1** we only measure CPU time. For **Prim2** and **iMax** we measure CPU time and memory requirements.

We summarize the experimental results in Figs. 12, 14 and 13, and Table 2. Table 2 shows our least squares fittings for the MST experiments. First of all, we compute the fitting for the number of lowest-cost edges Kruskal's MST algorithm reviews to build the tree. We obtain $0.532\ |V| \ln |V|$, which is very close to the theoretically expected value $\frac{1}{2}|V| \ln |V|$. Second, we compute fittings for the CPU cost of the three versions of Kruskal's using their theoretical complexity

[3] Note that we use the plain **Prim1**, that is, without priority queues. This is the best choice to implement Prim in complete graphs.

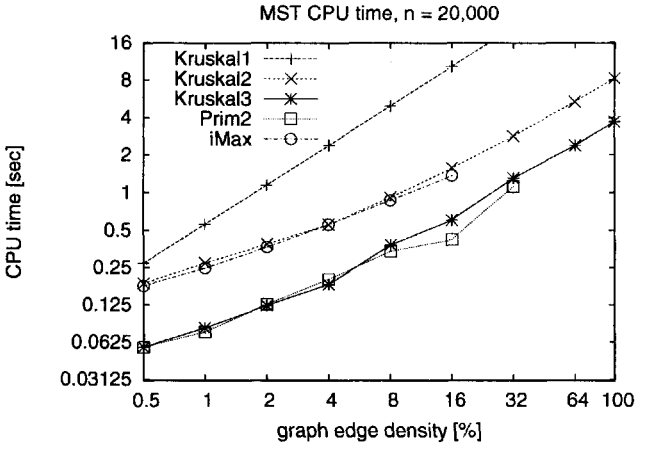


Figure 12: Evaluating **Kruskal3**. MST CPU time, dependence on ρ. For $\rho = 100\%$ **Kruskal1** reaches 70.1 seconds. Note the logscale.

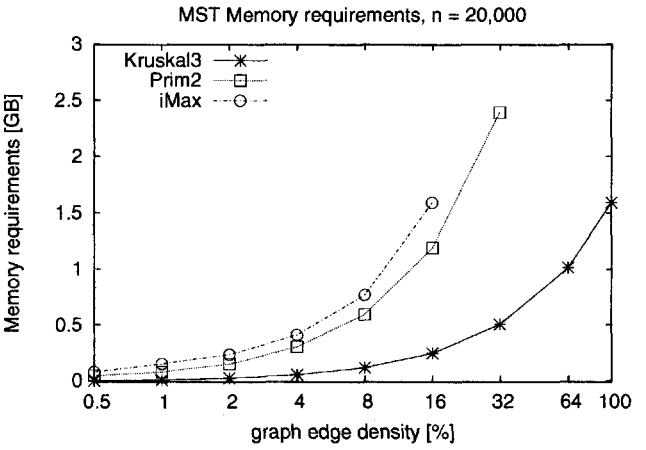


Figure 13: Memory used by **Kruskal3**, **iMax** and **Prim2** for $|V| = 20000$ nodes, dependence on ρ. As can be seen, **iMax** and **Prim2** exhausts the memory for $\rho \geq 16\%$ and $\rho \geq 32\%$, respectively.

models. Note that, in terms of CPU time, **Kruskal1** is 18.27 times and **Kruskal2** is 2.14 times slower than **Kruskal3**.

Fig. 12 compares the Kruskal's versions, **Prim2** and **iMax** for $n = 20000$ and graph edge density $\rho \in [0.5\%, 100\%]$. As can be seen, **Kruskal1** is, by far, the slowest version, and, **Kruskal3** is systematically the best for all ρ. We also notice that, as ρ increases, the advantage of our MST variant is more remarkable against basic Kruskal's MST algorithm. We could not complete the series for **Prim2** and **iMax**, as their structures require too much space. For 20000 vertices and $\rho \geq 32\%$ these algorithms reach the 3 GB out-of-memory threshold of our machine.

Fig. 13 shows the memory requirements of **Kruskal3**, **iMax** and **Prim2** for $n = 20000$. Since our Kruskal's implementation sort edges in place, we require a bit of extra memory to manage the edge incremental sorting. On the other hand, the additional structures of **Prim2** and **iMax** increase heavily the memory consumption of the process. We suspect that these high memory requirements trigger many cache faults and slow down the CPU performance. As a result, for large graphs, **Prim2** and **iMax** become slower than **Kruskal3**, despite their better complexity.

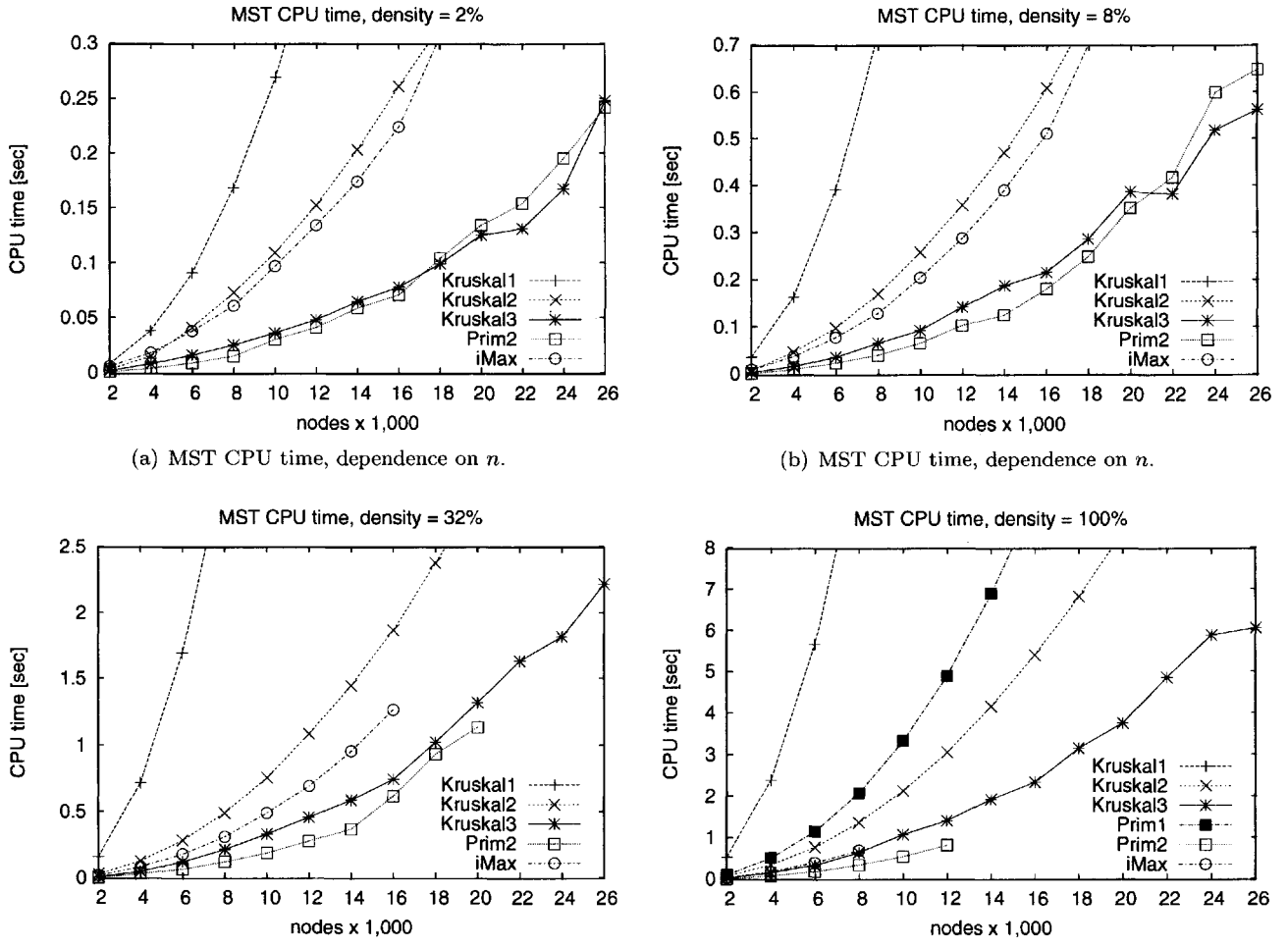
Figs. 14(a), 14(b), 14(c) and 14(d) show the comparison for four edge densities $\rho = 2\%$, 8%, 32% and 100%, respectively. In the four plots **Kruskal3** is always the best Kruskal's version for all sizes of set V and all edge densities ρ. Moreover, Fig. 14(d) shows that **Kruskal3** is also better than **Prim1**, even in complete graphs. On the other hand, **Kruskal3** is better than **iMax** in the four plots, and very competitive against **Prim2**, beating **Prim2** in some cases (for $|V| \geq 18000$ and 22000 vertices in $\rho = 2\%$ and 8%, respectively). We suspect that this is due to the high memory usage of **Prim2**, which affects cache efficiency. Note that with $\rho = 32\%$ and 100% we could not finish the series with **Prim2** and **iMax** because of their memory requirements.

7 Conclusions

We have presented IncrementalQuickSelect (**IQS**), an algorithm to incrementally retrieve the next smallest element from a set. **IQS** has the same average complexity as existing solutions, but it is considerably faster in practice. It is nearly as fast as the best algorithm that knows beforehand the number of elements to retrieve. We have applied **IQS** to solve the graph MST problem, showing that the **IQS**-based Kruskal's version is competitive against the best state-of-the-art alternatives.

One trend of further work considers studying the behaviour of our **IQS**-based Kruskal on different graph classes, and also research variants tuned for secondary memory. Another trend is to look for other applications of **IQS**. We finish by detailing two of them.

The first is that we can use the **IQS** stack-of-pivots underlying idea to partially sort in increasing/decreasing order starting from any place of the array. For instance, if we want to perform an incremental sorting in increasing order with a stack initialized as the set size, we first use QuickSelect to find the first element we want, storing in the stack all the pivots larger than the first element, and later we use **IQS**



(a) MST CPU time, dependence on n.

(b) MST CPU time, dependence on n.

(c) MST CPU time, dependence on n.

(d) MST CPU time, dependence on n.

Figure 14: Evaluating **Kruskal3**. MST CPU time, dependence on $n = |V|$ in (a), (b), (c) and (d) for $\rho = 2\%$, 8%, 32% and 100%, respectively. For $n = 26000$, in (a) **Kruskal1**, **Kruskal2** and **iMax** reach 2.67, 0.76 and 0.62 seconds; in (b) **Kruskal1**, **Kruskal2** and **iMax** reach 9.08, 1.56 and 1.53 seconds; in (c) **Kruskal1** and **Kruskal2** reach 37.02 and 4.82 seconds; in (d) **Kruskal1**, **Kruskal2** and **Prim1** reach 121.14, 13.84 and 25.96 seconds, respectively.

with the stack to search for the next elements (the other pivots would be useful for decreasing order, initializing the stack with -1). Moreover, with two stacks we can make centered searching, namely, finding the k-th element, the $(k+1)$-th and $(k-1)$-th, the $(k+2)$-th and $(k-2)$-th, and so on.

The second remarkable application is that we can use **IQS** as an underlying implementation of the HEAP data structure [5, 25]. (Naturally, this allow us to speed up HEAPSORT [26].) In this application, we heapify the set A by using **IQS** to search for the first element, paying on average $O(m)$ CPU time, and then we extract elements by using **IQS** incrementally, paying average amortized time $O(\log k)$ for the k-th extraction. To insert a new element x, we need to know which is the array segment that corresponds to x (see Fig. 1). To do this it is enough with reviewing the pivot stack S. Assume $S = \{|A|, p_1, p_2, ..., p_j\}$. From Lemma 2.1, we know that $A[p_1] > A[p_2] > \ldots > A[p_j]$. So, to insert x we need to find the first pivot p_i such that $A[p_i] < x$, so as to place x at $A[p_{i-1}]$. Then, we put $A[p_{i-1}]$ at position $p_{i-1} + 1$ (and increment p_{i-1} in S). Then, we move the old $A[p_{i-1} + 1]$ value to $A[p_{i-2}]$, and so on. Note that, as pivot closer to the bottom cover exponentially larger areas, the insertion takes $O(1)$ time on average. With this **IQS**-based heap we can reach the $O(m+n \log n)$ performance of Fibonacci-heap-based Prim's algorithm [8], yet using a rather simple heap.

Acknowledgment

We wish to thank the very valuable comments from the anonymous referees.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[2] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[3] J. M. Chambers. Algorithm 410 (PARTIAL SORTING). *Communications of the ACM*, 14(5):357–358, 1971.

[4] B. Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.

[6] R. W. Floyd. Algorithm 245 (TREESORT). *Communications of the ACM*, 7:701, 1964.

[7] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. The pairing heap: a new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986.

[8] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[9] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures.* Addison-Wesley, 2nd edition, 1991.

[10] C. A. R. Hoare. Algorithm 65 (FIND). *Communications of the ACM*, 4(7):321–322, 1961.

[11] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[12] S. Janson, D. Knuth, T. Luczak, and B. Pittel. The birth of the giant component. *Random Structures & Algorithms*, 4(3):233–358, 1993.

[13] D. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995.

[14] I. Katriel, P. Sanders, and J. Träff. A practical minimum spanning tree algorithm using the cycle property. Research Report MPI-I-2002-1-003, Max-Planck-Institut für Informatik, October 2002.

[15] I. Katriel, P. Sanders, and J. Träff. A practical minimum spanning tree algorithm using the cycle property. In *11th European Symposium on Algorithms (ESA'03)*, LNCS 2832, pages 679–690, 2003.

[16] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[17] C. Martínez. Partial quicksort. In *Proc. 6th ACM-SIAM Workshop on Algorithm Engineering and Experiments and 1st ACM-SIAM Workshop on Analytic Algorithmics and Combinatorics*, pages 224–228, 2004.

[18] B. Moret and H. Shapiro. An empirical analysis of algorithms for constructing a minimum spanning tree. In *Proc. 2nd Workshop Algorithms and Data Structures (WADS'91)*, LNCS 519, pages 400–411, 1991.

[19] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1):16–34, 2002.

[20] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[21] R Development Core Team. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria, 2004.

[22] P. Sanders. Fast priority queues for cached memory. *J. Exp. Algorithmics*, 5:7, 2000.

[23] R. E. Tarjan. *Data structures and network algorithms.* Society for Industrial and Applied Mathematics, 1983.

[24] I. Wegener. BOTTOM-UP-HEAPSORT, a new variant of HEAPSORT beating, on an average, QUICKSORT (if n is not very small). *Theoretical Computer Science*, 118(1):81–98, 1993.

[25] M. Weiss. *Data structures & algorithm analysis in Java™.* Addison-Wesley, 1999.

[26] J. Williams. Algorithm 232 (HEAPSORT). *Communications of the ACM*, 7(6):347–348, 1964.

Workshop on Analytic Algorithmics and Combinatorics

This page intentionally left blank

Deterministic Random Walks*

Joshua Cooper† Benjamin Doerr‡ Joel Spencer§ Garbor Tardos¶

Abstract

Jim Propp's P-machine, also known as 'rotor router model' is a simple deterministic process that simulates a random walk on a graph. Instead of distributing chips to randomly chosen neighbors, it serves the neighbors in a fixed order.

We investigate how well this process simulates a random walk. For the graph being the infinite path, we show that, independent of the starting configuration, at each time and on each vertex, the number of chips on this vertex deviates from the expected number of chips in the random walk model by at most a constant c_1, which is approximately 2.29. For intervals of length L, this improves to a difference of $O(\log L)$ (instead of $2.29L$), for the L_2 average of a contiguous set of intervals even to $O(\sqrt{\log L})$. It seems plausible that similar results hold for higher-dimensional grids $\mathbb{Z}^d$ instead of the path $\mathbb{Z}$.

1 The Propp Machine

The following deterministic process was suggested by Jim Propp as an attempt to derandomize random walks on infinite grids $\mathbb{Z}^d$:

> **Rules of the Propp machine:** Each vertex $x \in \mathbb{Z}^d$ is associated with a 'rotor' and a cyclic permutation of the $2d$ cardinal directions of $\mathbb{Z}^d$. Each vertex may hold an arbitrary number of 'chips'. In each time step, each vertex sends out all its chips to neighboring vertices in the following manner: The first chip is sent into the direction the rotor is pointing, then the rotor direction is updated to the next direction in the cyclic ordering. The second chip is sent in this direction, the rotor is updated, and so on. In consequence, the chips are distributed highly evenly among the neighbors.

This process found considerable attention recently. It turns out that the Propp machine in several respects behaves highly similar to a random walk. Used to simulate internal diffusion limited aggregation (repeatedly, a single chip is inserted at the origin, performs a rotor router walk until it reaches an unoccupied position and occupies it), it was shown by Levine and Peres [LP] that this derandomization produces results that are extremely close to what a random walk would have produced. See also Kleber's paper [Kle05], which adds an interesting experimental results: Having inserted three million chips, the closest unoccupied site is at distance 976.45, the farthest occupied site is at distance 978.06. Hence the occupied sites almost form a perfect circle!

In [CS05, CS], the authors consider the following question: Start with an arbitrary initial position (that is, chips on vertices and rotor directions), run the Propp machine for some time and compare the number of chips on a vertex with the expected number of chips a random walk (run for the same amount of time) would have gotten to that vertex. Apart from a technicality, which we defer to the end of Section 2, the answer is astonishing: For any grid $\mathbb{Z}^d$, this difference (discrepancy) can be bounded by a constant, independent of the number of chips, the run-time, the initial rotor positions and the cyclic permutations of the cardinal directions.

In this paper, we continue this work. We mainly regard the one-dimensional case, but as will be visible from the proofs, our methods can be extended to higher dimensions as well. Besides making the constant precise (approximately 2.29), we show that the differences become even better for larger intervals (both in space and time). We also present a fairly general method to prove lower bounds (the 'arrow forcing theorem'). This shows that all our upper bounds actually are sharp (including the constant of about 2.29).

Instead of talking about the expected number of chips the random walk produces on a vertex, we find it more convenient to think of the following 'linear'

*The authors enjoyed the hospitality, generosity and the strong coffee of the Rényi Institute (Budapest) while doing this research. Spencer's research was partially supported by EU Project Finite Structures 003006; Doerr's by EU Research Training Network COMBSTRU; Cooper's by an NSF Postdoctoral Fellowship (USA, NSF Grant DMS-0303272); and Tardos was a member of the Rényi Institute.

†Courant Institute of Mathematical Sciences, New York, U.S.A. Research supported by NSF Grant DMS-0303272.

‡Max-Planck-Institut für Informatik, Saarbrücken, Germany.

§Courant Institute of Mathematical Sciences, New York, U.S.A.

¶Rényi Institute of the Hungarian Academy of Sciences, Budapest, Hungary.

machine. Here, in each time step each vertex sends out exactly the same number (possibly non-integral) of chips to each neighbor. Hence for a given starting configuration, after t time-steps the number of chips in the linear model is exactly the expected number of chips in the random walk model.

2 Our Results

We obtain the following results (again, see the end of the section for a slight technical restriction): Fix any starting configuration, that is, the number of chips on each vertex and the position of the rotor on each vertex. Now run both the Propp machine and the linear machine for a fixed number of time-steps. Looking at the resulting chip configurations, we have the following:

- On each vertex, the number of chips in both models deviates by at most a constant $c_1 \approx 2.29$. We may interpret this as that the Propp machine simulates a random walk extremely well. In some sense, it is even better than the random walk. Recall that in a random walk a vertex holding n chips only in expectation sends $n/2$ chips to the left and the right. With high probability, the actual numbers deviate from this by $\Omega(n^{1/2-\varepsilon})$.

- In each interval of length L, the number of chips that are in this interval in the Propp model deviates from that in the linear model by only $O(\log L)$ (instead of, e.g., $2.29L$).

- If we average this over all length L intervals in some larger interval of $\mathbb{Z}$, things become even better. The average squared discrepancy in the length L intervals also is only $O(\log L)$.

We may as well average over time. In the setting just fixed, denote by $f(x,T)$ the sum of the numbers of chips on vertex x in the last T time steps in the Propp model, and by $E(x,T)$ the corresponding number for the linear model. Then we have the following discrepancy bounds:

- The discrepancy on a single vertex over a time interval of length T is at most $|f(x,T)-E(x,T)| = O(T^{1/2})$. Hence a vertex cannot have too few or too many chips for a long time (it may, however, alternate having too few and too many chips and thus have an average $\Omega(1)$ discrepancy over time).

- We may extend this to discrepancies in intervals in space and time: Let I be some interval in $\mathbb{Z}$ having length L. Then the discrepancy in I over a time interval of length T satisfies

$$\left|\sum_{x\in I} f(x,T) - \sum_{x\in I} E(x,T)\right| = \begin{cases} O(LT^{1/2}) & \text{if } L \le T^{1/2}, \\ O(T\log(LT^{-1/2})) & \text{otherwise.} \end{cases}$$

  Hence if L is small compared to $T^{1/2}$, we get L times the single vertex discrepancy in a time interval of length T (no significant cancellation in space); if L is of larger order than $T^{1/2}$, we get T times the $O(\log L)$ bound for intervals of length L (no cancellation in time, the discrepancy cannot leave the large interval in short time).

All bounds stated above are sharp, that is, for each bound there is a starting configuration such that after suitable run-time of the machines we find the claimed discrepancy on a suitable vertex, in a suitable interval, etc.

A technicality: There is one limitation, which we only briefly mentioned, but without which our results are not valid. Note that since $\mathbb{Z}^d$ is a bipartite graphs, the chips that start on even vertices ("even chips" for short) never mix with those which start on odd positions ("odd chips"). It looks as if we would play two games in one. This is not true, however. The even chips and the odd ones do interfere with each other through the rotors.

In fact, there are initial positions such that the odd chips suitably reset the arrows and thus mess up the even chips. Note that the odd chips are not visible if we look at an even position after an even run-time. Hence in this general setting, no useful bounds on discrepancies on a single vertex can exist.

We therefore assume that *the starting configuration has chips only on even positions* ("even starting configuration"). Of course, nothing would change if we only allowed odd chips.

An alternative, in fact equivalent, solution would be to have two rotors on each vertex, one for even and one for odd chips.

3 The Basic Method

For numbers a and b set $[a..b] = \{z \in \mathbb{Z} \mid a \le z \le b\}$ and $[b] = [1..b]$. For integers m and n, we write $m \sim n$ if m and n have the same parity, that is, if $m-n$ is even.

For a fixed starting configuration, we use $f(x,t)$ to denote the number of chips at time t at position x and $\mathrm{ARR}(x,t)$ to denote the value of the arrow at time t and position x, i.e., $+1$ if it points to the right, and -1

if it points to the left. We have:

$$\begin{aligned} & f(x,t+1) \\ &= f(x-1,t)/2 + f(x+1,t)/2 \\ &\quad + \mathrm{ARR}(x-1,t)(f(x-1,t) \bmod 2)/2 \\ &\quad - \mathrm{ARR}(x+1,t)(f(x+1,t) \bmod 2)/2, \\ & \mathrm{ARR}(x,t+1) \\ &= (-1)^{f(x,t)}\mathrm{ARR}(x,t). \end{aligned}$$

Note that after an even starting configuration if $x \sim t$ does not hold, then we have $f(x,t) = 0$ and $\mathrm{ARR}(x,t+1) = \mathrm{ARR}(x,t)$.

We consider the machine to be started at time $t = 0$. Being a deterministic process, the initial configuration (i.e., the values $f(x,0)$ and $\mathrm{ARR}(x,0)$, $x \in \mathbb{Z}$) determines the configuration at any time $t > 0$ (i.e., the values $f(x,t)$ and $\mathrm{ARR}(x,t)$, $x \in \mathbb{Z}$). The totality of all configurations for $t > 0$ we term a *game*. We call a configuration *even* if no chip is at an odd position. Similarly, a position is *odd* if no chip is at an even position. Clearly, an even position is always followed by an odd position and vice versa.

By $E(x,t)$ we denote the expected number of chips on a vertex x after running a random walk for t steps (from the implicitly given starting configuration). As described earlier, this is equal to the number of chips on x after running the linear machine for t time-steps.

In the proofs, we need the following mixed notation. Let $E(x,t_1,t_2)$ be the expected number of chips at location x and time t_2 if a simple random walk were performed beginning from the Propp machine's configuration at time t_1. In other words, this is the number of chips on vertex x after t_1 Propp and $t_2 - t_1$ linear steps.

Let $H(x,t)$ denote the probability that a chip arrives at location x at time t in a simple random walk begun from the origin, i.e., $H(x,t) = 2^{-t}\binom{t}{(t+x)/2}$, if $t \sim x$, and $H(x,t) = 0$ otherwise. Let $\mathrm{INF}(y,t)$ denote the "influence" of a Propp step of a single chip at distance y with t linear steps remaining (compared to a linear step). More precisely, we compare the two probabilities that a chip on position y reaches 0 if (a) it is first sent to the right (by a single Propp step) and then does a random walk for the remaining $t-1$ time steps, or (b) it just does t random walk steps starting from y. Hence,

$$\mathrm{INF}(y,t) := H(y+1,t-1) - H(y,t).$$

A simple calculation yields

$$\mathrm{INF}(y,t) = -\tfrac{y}{t}H(y,t). \tag{3.1}$$

This shows in particular, that $\mathrm{INF}(y,t) \le 0$ for $y \ge 0$ and $\mathrm{INF}(y,t) \ge 0$ for $y \le 0$. We have $\mathrm{INF}(0,t) = 0$.

Note that

$$\mathrm{INF}(y,t) = \tfrac{1}{2}H(y+1,t-1) - \tfrac{1}{2}H(y-1,t-1). \tag{3.2}$$

Therefore, the first Propp step with arrow pointing to the left has an influence of $-\mathrm{INF}(y,t)$.

Using this notation, we can conveniently express the (signed) discrepancy $f(x,t) - E(x,t)$ on a vertex x using information about when "odd splits" occurred. It suffices to prove the result for the vertex $x = 0$. Clearly, $E(0,t,t) = f(0,t)$ and $E(0,0,t) = E(0,t)$, so that

$$f(0,t) - E(0,t) = \sum_{s=0}^{t-1} \left(E(0,s+1,t) - E(0,s,t)\right). \tag{3.3}$$

In comparing $E(0,s+1,t)$ and $E(0,s,t)$, note that whenever there are two chips on some vertex at time s, then these chips can be assumed to behave identically no matter whether the next step is a linear or a Propp step. Denote by ODD_s the set of locations which are occupied by an odd number of chips at time s. Then

$$\begin{aligned} & E(0,s+1,t) - E(0,s,t) \\ &= \sum_{y \in \mathrm{ODD}_s} \left(H(y + \mathrm{ARR}(y,s), t-s-1) - H(y,t-s)\right) \\ &= \sum_{y \in \mathrm{ODD}_s} \mathrm{ARR}(y,s)\,\mathrm{INF}(y,t-s). \end{aligned}$$

Therefore, appealing to (3.3),

$$f(0,t) - E(0,t) = \sum_{s=0}^{t-1} \sum_{y \in \mathrm{ODD}_s} \mathrm{ARR}(y,s)\,\mathrm{INF}(y,t-s).$$

Let $s_i(y)$ be the i^{th} time (up to time t) that y is occupied by an odd number of chips, beginning with $i = 0$. Switching the order of summation and noting that the arrows flip each time there is an odd number of chips on a vertex, we have

$$\begin{aligned} & f(0,t) - E(0,t) \\ &= \sum_{y \in \mathbb{Z}} \sum_{i \ge 0} \mathrm{ARR}(y, s_i(y))\,\mathrm{INF}(y, t - s_i(y)) \\ &= \sum_{y \in \mathbb{Z}} \mathrm{ARR}(y,0) \sum_{i \ge 0} (-1)^i\,\mathrm{INF}(y, t - s_i(y)). \end{aligned} \tag{3.4}$$

This equation will be crucial in the remainder of the paper. It shows that the discrepancy on a vertex only depends on the initial arrow positions and the set of location-time pairs holding an odd number of chips.

In the remainder, we show that we can construct starting configurations with arbitrary initial arrow positions and odd number of chips at arbitrary sets of location-time pairs. This will be the heart of our lower bound proofs in the following sections. Here $\mathbb{N}_0$ denotes the set of non-negative integers.

THEOREM 3.1. (PARITY-FORCING THEOREM) *For any initial position of the arrows and any $\pi : \mathbb{Z} \times \mathbb{N}_0 \to \{0,1\}$, there is an initial even configuration of the chips such that for all $x \in \mathbb{Z}$, $t \in \mathbb{N}_0$ such that $x \sim t$, $f(x,t)$ and $\pi(x,t)$ have identical parity.*

Since rotors change their direction if and only if the vertex has an odd number of chips, the parity-forcing theorem is a consequence of the following arrow-forcing statement.

THEOREM 3.2. (ARROW-FORCING THEOREM) *Let $\rho(x,t) \in \{-1,+1\}$ be arbitrarily defined for $t \geq 0$ integer and $x \sim t$. Then there exists an even initial configuration that results in a game with $\mathrm{ARR}(x,t) = \rho(x,t)$ for all such x and t. Similarly, if $\rho(x,t)$ is defined for $x \sim t+1$ a suitable odd initial configuration can be found.*

Proof. By symmetry, it is enough to prove the first statement.

Assume the functions f and ARR describe the game following an even initial configuration, and for some $T \geq 0$, we have $\mathrm{ARR}(x,t) = \rho(x,t)$ for all $0 \leq t \leq T+1$ and $x \sim t$. We modify the initial position by defining $f'(x,0) = f(x,0) + \epsilon_x 2^T$ for even x, while we have $f'(x,0) = 0$ for odd x and $\mathrm{ARR}'(x,0) = \mathrm{ARR}(x,0)$ for all x. Here, $\epsilon_x \in \{0,1\}$ are to be determined.

Observe that a pile of 2^T chips will split evenly T times so that the arrows at time $t \leq T$ remain the same. Our goal is to choose the values ϵ_x so that $\mathrm{ARR}'(x,t) = \rho(x,t)$ for $0 \leq t \leq T+2$ and $x \sim t$. As stated above this holds automatically for $t \leq T$ as $\mathrm{ARR}'(x,t) = \mathrm{ARR}(x,t) = \rho(x,t)$ in this case. For $t = T+1$ and $x-T-1$ even we have $\mathrm{ARR}'(x,T+1) = \mathrm{ARR}'(x,T) = \mathrm{ARR}(x,T) = \mathrm{ARR}(x,T+1) = \rho(x,T+1)$ since we start with an even configuration. To make sure the equality also holds for $t = T+2$ we need to ensure that the parities of the piles $f'(x,T)$ are right. Observe that $\mathrm{ARR}'(x,T+2) = \mathrm{ARR}'(x,T)$ if $f'(x,T)$ is even, otherwise $\mathrm{ARR}'(x,T+2) = -\mathrm{ARR}'(x,T)$. So for $x-T$ even we must make $f'(x,T)$ even if and only if $\rho(x,T+2) = \rho(x,T)$. At time T the "extra" groups of 2^T chips have spread as in Pascal's Triangle and we have

$$f'(x,T) = f(x,T) + \sum_y \epsilon_y \binom{T}{\frac{T+x-y}{2}}$$

where $x \sim T$ and the sum is over the even values of y with $|y-x| \leq T$. As $f(x,T)$ are already given it suffices to set the parity of the sum arbitrarily. For $T = 0$ the sum is ϵ_x so this is possible. For $T > 0$ we express

$$\sum_y \epsilon_y \binom{T}{\frac{T+x-y}{2}} = \epsilon_{x+T} + h + \epsilon_{x-T}$$

where h depends only on ϵ_y with $x-T < y < x+T$. We now determine the ϵ_y sequentially. We initialize by setting $\epsilon_y = 0$ for $-T < y \leq T$. The values ϵ_y for $y > T$ are set in increasing order. The value of ϵ_y is set so that the sum at $x = y - T$ (and thus $f'(y-T,T)$) will have the correct parity. Similarly, the values ϵ_y for $y \leq -T$ are set in decreasing order. The value of ϵ_y is set so that the sum at $x = y+T$ (and thus the $f'(y+T,T)$) will have the correct parity.

Note that the above procedure changes an even initial configuration that matches the prescription in ρ for times $0 \leq t \leq T+1$ into another even initial configuration that matches the prescription in ρ for times $0 \leq t \leq T+2$. We start by defining $f(x,0) = 0$ for all x (no chips anywhere) and $\mathrm{ARR}(x,0) = \rho(x,0)$ for even x, while $\mathrm{ARR}(x,0) = \rho(x,1)$ for odd x. We now have $\mathrm{ARR}(x,t) = \rho(x,t)$ for $0 \leq t \leq 1$ and $x \sim t$. We can apply the above procedure repeatedly to get an even initial configuration that satisfies the prescription in ρ for an ever increasing (but always finite) time period $0 \leq t < T$. Notice however, that in the procedure we do not change the initial configuration of arrows $\mathrm{ARR}(x,0)$ at all, and we change the initial number of chips $f(x,0)$ at position x only if $|x| \geq T$. Thus at any given position x the initial number of chips will be constant after the first $|x|$ iterations. This means that the process converges to an (even) initial configuration. It is simple to check that this limit configuration satisfies the statement of the theorem.

4 Discrepancy on a Single Vertex

THEOREM 4.1. *There exists a constant $c_1 \approx 2.29$, independent of the initial (even) configuration, the time t, or the location x, so that*

$$|f(x,t) - E(x,t)| \leq c_1.$$

The proof needs the following elementary fact. Let $X \subseteq \mathbb{R}$. We call a mapping $f : X \to \mathbb{R}$ *unimodal*, if there is an $m \in X$ such that f is monotonically increasing in $\{x \in X \mid x \leq m\}$ and f is monotonically decreasing in $\{x \in X \mid x \geq m\}$.

LEMMA 4.1. *Let $f : X \to \mathbb{R}$ be non-negative and unimodal. Let $t_1, \ldots, t_n \in X$ such that $t_1 < \ldots < t_n$. Then*

$$\left| \sum_{i=1}^{n} (-1)^i f(t_i) \right| \leq \max_{x \in X} f(x).$$

It suffices to prove Theorem 4.1 for $x = 0$. In case t is even we start with an even configuration, if t is odd, then with an odd configuration (otherwise both $f(0,t)$ and $E(0,t)$ would be zero with no discrepancy).

First we show that $\mathrm{INF}(y,u)$ with a fixed $y<0$ is a non-negative unimodal function of u if restricted to the values $u \sim y$. We have already seen that it is non-negative. For the unimodality let $y<0$ and $u>2$, $u \sim y$. We have

$$\begin{aligned}&\mathrm{INF}(y,u)-\mathrm{INF}(y,u+2)\\&=-\frac{y}{u}H(y,u)+\frac{y}{u+2}H(y,u+2)\\&=\frac{4+3u-y^2}{(u+2-y)(u+2+y)}\mathrm{INF}(y,u),\end{aligned}$$

whenever $u \geq y$. Hence the difference is non-negative if $u \geq (y^2-4)/3$ and it is non-positive if $u \leq (y^2-4)/3$. Thus we have unimodality, with $\mathrm{INF}(y,u)$ taking its maximum at the largest value of u not exceeding $(y^2-4)/3$ with $u \sim y$. Let $t_{\max}(y) := \lfloor (y^2-4)/3 \rfloor$. It is easy to check that $t_{\max}(y) \sim y$ always holds, so we have that $\mathrm{INF}(y,u)$ takes its maximum for fixed $y \leq -4$ at $u = t_{\max}(y)$. For $1 \leq |y| \leq 3$ we have $\min_t |\mathrm{INF}(y,t)| = |\mathrm{INF}(y,|y|)|$, for $y=0$, $\min_t |\mathrm{INF}(y,t)| = 0$, and, for $y \geq 4$ we have, by symmetry, that $-\mathrm{INF}(y,t)$ is non-negative and unimodal (again, restricted to $u \sim y$), with maximum taken at $u = t_{\max}(y)$. Therefore, define $t'_{\max}(y) = t_{\max}(y)$ when $|y|>3$, otherwise $t_{\max}(y) = |y|$.

LEMMA 4.2. *For $y \in \mathbb{Z}$, the function $|\mathrm{INF}(y,t)|$ is maximized over all nonnegative integers t with $t \sim y$ at $t = \max\{|y|, \lfloor (y^2-4)/3 \rfloor\}$.*

To bound $|f(0,t)-E(0,t)|$ we use the formula (3.4) where the inner sums are alternating sums, for which we can apply Lemma 4.1, as $y \sim t - s_i(y)$ holds by our even or odd starting position assumption. We get

$$\begin{aligned}|f(0,t)-E(0,t)| &\leq \sum_{y\in\mathbb{Z}} \left| \sum_{i\geq 0} (-1)^i \mathrm{INF}(y, t-s_i(y)) \right| \\ &\leq \sum_{y\in\mathbb{Z}} \max_u |\mathrm{INF}(y,u)| \\ &= 2\sum_{y=1}^{\infty} |\mathrm{INF}(y, t'_{\max}(y))|. \qquad (4.5)\end{aligned}$$

Note that

$$\begin{aligned}&|\mathrm{INF}(y, t'_{\max}(y))| \\ &= \frac{y}{t'_{\max}(y)} 2^{-t'_{\max}(y)} \binom{t'_{\max}(y)}{(t'_{\max}(y)+y)/2} \\ &= O(y/(t_{\max}(y))^{3/2}) = O(y^{-2}).\end{aligned}$$

Therefore, (4.5) implies that $|f(0,t)-E(0,t)|$ is bounded. With

$$c_1 := 2\sum_{y=1}^{\infty} |\mathrm{INF}(y, t'_{\max}(y))| \approx 2.29$$

we have therefore

$$|f(0,t)-E(0,t)| < c_1.$$

Amazingly, the constant c_1 defined above is best possible. Indeed, let $y>0$ be arbitrary and even and let $t_0 = t_{\max}(y)$. We apply the Arrow-forcing Theorem to find an even starting position that makes $\mathrm{ARR}(x,t) = -1$ if $x>0$ and $t \leq t_0 - t_{\max}(x)$ or $x<0$ and $t > t_0 - t_{\max}(x)$ and makes $\mathrm{ARR}(x,t) = -1$ otherwise. It is easy to verify that in this case at a position $|x| \leq y$, $x \neq 0$ we have an odd number of chips exactly once at time $t_0 - t_{\max}(x)$ and the formula (3.4) gives

$$f(0,t_0) - E(0,t_0) = 2\sum_{x=1}^{y} |\mathrm{INF}(x, t_{\max}(x))|.$$

5 Intervals in Space

In this section, we regard the discrepancy in intervals in $\mathbb{Z}$. For an arbitrary finite subset X of $\mathbb{Z}$ set

$$\begin{aligned}f(X,t) &:= \sum_{x\in X} f(x,t),\\ E(X,t) &:= \sum_{x\in X} E(x,t).\end{aligned}$$

We show that the discrepancy in an interval of length L is $O(\log L)$, and this is sharp. We need the following facts about H.

LEMMA 5.1. *For all $x \in \mathbb{Z}$,*

$$H(x,\cdot) : \{t \in \mathbb{N}_0 \mid x \sim t\} \to \mathbb{R}; t \mapsto H(x,t)$$

is unimodal. $H(x,t)$ is maximal for $t = x^2-2$ and $t = x^2$. We have $H(x,x^2-2) = H(x,x^2) = \Theta(|x|^{-1})$.

Proof. Since $H(x,t-2) - H(x,t) = \frac{t-x^2}{t^2-t} H(x,t)$, we conclude that $H(x,t)$ is unimodal and it has exactly two maxima, namely $t = x^2-2$ and $t = x^2$.

THEOREM 5.1. *For any even initial configuration, any time t and any interval X of length L,*

$$|f(X,t) - E(X,t)| = O(\log L).$$

There is an even initial configuration, a time t and an interval X of length L such that

$$|f(X,t) - E(X,t)| = \Omega(\log L).$$

Proof. Without loss of generality, let $X = [-L+1..0]$. Fix any even initial configuration. By (3.4), we have

$$\begin{aligned}&f(X,t) - E(X,t)\\ &= \sum_{y\in\mathbb{Z}} \mathrm{ARR}(y,0) \sum_{x\in X} \sum_{i\geq 0} (-1)^i \mathrm{INF}(y-x, t-s_i(y)).\end{aligned}$$

Let us call

$$\mathrm{CON}(y) := \mathrm{ARR}(y,0)\sum_{x\in X}\sum_{i\geq 0}(-1)^i \mathrm{INF}(y-x, t-s_i(y))$$

the contribution of the vertex y to the discrepancy in the interval X. We first regard vertices y that lie outside X. By symmetry, we may assume $y \in \mathbb{N}$. The contribution of a vertex depends on its distance from the interval X. If y is $\Omega(L)$ away from X, its influences on the various vertices of X are roughly equal, and all such influences are quite small. In this case we bound its influence by L times the one we computed in Theorem 4.1:

Let $y > L$. By Lemma 4.1 and 4.2,

$$\begin{aligned}
|\mathrm{CON}(y)| &= \Big|\sum_{x\in X}\sum_{i\geq 0}(-1)^i \mathrm{INF}(y-x,t-s_i(y))\Big| \\
&\leq \sum_{x\in X}\Big|\sum_{i\geq 0}(-1)^i \mathrm{INF}(y-x,t-s_i(y))\Big| \\
&\leq \sum_{x\in X}\max_t |\mathrm{INF}(y-x,t)| \\
&\leq O\Big(\sum_{x\in X}(y-x)^{-2}\Big) = O(Ly^{-2}).
\end{aligned}$$

Hence the total contribution of these vertices is

$$\sum_{y>L}|\mathrm{CON}(y)| = O\Big(\sum_{y>L} Ly^{-2}\Big) = O(1).$$

We now turn to vertices $y \in [L]$. Here mainly those vertices of X that are close to y contribute to $\mathrm{CON}(y)$. Hence, the approach above is too coarse.

Let $\delta_i, \varepsilon_i \in \{0,1\}$ such that $y+L-\delta_i$ and $y-1+\varepsilon_i$ have the same parity as $t - s_i(y) - 1$. Then by (3.2), Lemma 4.1 and 5.1,

$$\begin{aligned}
&|\mathrm{CON}(y)| \\
&= \Big|\sum_{i\geq 0}(-1)^i \sum_{x\in X}\mathrm{INF}(y-x,t-s_i(y))\Big| \\
&= \Big|\tfrac{1}{2}\sum_{i\geq 0}(-1)^i \sum_{x\in X}(H(y-x+1,t-s_i(y)-1) \\
&\quad -H(y-x-1,t-s_i(y)-1))\Big| \\
&= \Big|\tfrac{1}{2}\sum_{i\geq 0}(-1)^i (H(y+L-\delta_i,t-s_i(y)-1) \\
&\quad -H(y-1+\varepsilon_i,t-s_i(y)-1))\Big| \\
&\leq \Big|\tfrac{1}{2}\sum_{i\geq 0}(-1)^i H(y+L-\delta_i,t-s_i(y)-1)\Big| \\
&\quad +\Big|\tfrac{1}{2}\sum_{i\geq 0}(-1)^i H(y-1+\varepsilon_i,t-s_i(y)-1)\Big| \\
&\leq \tfrac{1}{2}\max_{s\in\mathbb{N},\delta\in\{0,1\}} H(y+L-\delta,s) \\
&\quad +\tfrac{1}{2}\max_{s\in\mathbb{N},\varepsilon\in\{0,1\}} H(y-1+\varepsilon,s) \\
&= O(y^{-1}).
\end{aligned}$$

Thus the vertices in $[L]$ contribute $\sum_{y\in[L]}|\mathrm{CON}(y)| = O(\sum_{y=1}^{L} y^{-1}) = O(\log L)$.

For $y \in X$, note that exploiting symmetries, we can employ the same reasoning as above: Denote the upper bound for $|\mathrm{CON}(y)|$ computed above by $b_{y,L}$. Note that it is non-decreasing in terms of L.

We have $\mathrm{INF}(x,t) = -\mathrm{INF}(-x,t)$ for all $x \in \mathbb{Z}$ and $\mathrm{INF}(0,t) = 0$. Taking this into account, we see that for each $y \in X$, there is a subinterval $X_y \subseteq X$ such that $y \notin X_y$ and

$$\mathrm{CON}(y) = \mathrm{ARR}(y,0)\sum_{i\geq 0}(-1)^i \sum_{x\in X_y}\mathrm{INF}(y-x,t-s_i(y)).$$

Denote by d_y the distance from y to X_y, that is, $d_y = \min_{x\in X_y}|y-x|$. Then the above yields $\mathrm{CON}(y) \leq b_{d_y,|X_y|} \leq b_{d_y,L}$. Note further, that at most two different y yield the same d_y. Hence $\sum_{y\in X}|\mathrm{CON}(y)| \leq 2\sum_{y\in\mathbb{N}}|\mathrm{CON}(y)|$.

Combining all cases, we have

$$|f(X,t) - E(X,t)| \leq \sum_{y\in\mathbb{Z}}|\mathrm{CON}(y)| = O(\log L).$$

For the lower bound, we just have to place the chips in a way the one of the logarithmic terms above actually occurs. Without loss of generality, let L be odd.

Consider the following initial configuration (its existence is ensured by the parity forcing theorem): All arrows point towards the interval X (arrows of vertices in X may point anywhere). Let $t = L^2$. Choose an initial configuration of the chips such that $f(y,s)$ is odd if and only if $y \in [L]$ is even and $t - s = y^2$.

Now by construction, $\mathrm{CON}(y) = 0$ for all $y \in \mathbb{Z}\backslash[L]$. For $y \in [L]$, we have

$$\begin{aligned}\mathrm{CON}(y) &= -\sum_{x\in X} \mathrm{INF}(y-x, y^2)\\ &= \tfrac{1}{2}H(y,y^2) - \tfrac{1}{2}H(y+L+1,y^2)\\ &\geq \tfrac{1}{2}H(y,y^2) - \tfrac{1}{2}H(y+L+1,(y+L+1)^2)\\ &= \Omega(y^{-1}).\end{aligned}$$

Hence for this initial configuration,

$$\begin{aligned}f(X,t) - E(X,t) &= \sum_{y\in\mathbb{Z}} \mathrm{CON}(y)\\ &= \sum_{y\in[L],y\sim 2} O(y^{-1}) = \Omega(\log L).\end{aligned}$$

6 Intervals in Time

In this section, we regard the discrepancy in time-intervals. For $x \in \mathbb{Z}$ and finite $S \subseteq \mathbb{N}_0$, set

$$\begin{aligned}f(x,S) &:= \sum_{t\in S} f(x,t),\\ E(x,S) &:= \sum_{t\in S} E(x,t).\end{aligned}$$

We show that the discrepancy of a single vertex in a time-interval of length T is $O(\sqrt{T})$, and this is sharp.

THEOREM 6.1. *The maximal discrepancy $|f(x,S) - E(x,S)|$ of a single vertex x in a time interval S of length T is $\Theta(T^{1/2})$.*

In the proof, we need the following fact that "rolling sums" of unimodal functions are unimodal again.

LEMMA 6.1. (UNIMODALITY OF ROLLING SUMS) *Let $f : \mathbb{Z} \to \mathbb{R}$ be unimodal. Let $k \in \mathbb{N}$. Define $F : \mathbb{Z} \to \mathbb{R}$ by $F(z) = \sum_{i=0}^{k-1} f(z+i)$. Then F is unimodal.*

Proof. If f is non-decreasing (resp. non-increasing), then clearly so is F and thus it is unimodal. Hence let f and $m \in \mathbb{Z}$ be such that f is non-decreasing in $\mathbb{Z}_{\leq m}$ and non-increasing in $\mathbb{Z}_{\geq m}$. We show that for some $M \in \mathbb{Z}$, $G(x) := F(x+1) - F(x)$ is nonnegative in $\mathbb{Z}_{\leq M}$ and not positive in $\mathbb{Z}_{\geq M}$. This implies that F is unimodal.

Since $G(x) = f(x+k) - f(x)$ for all $x \in \mathbb{Z}$, G is non-negative in $\mathbb{Z}_{\leq m-k}$ and not positive in $\mathbb{Z}_{\geq m}$. Let $x \in \mathbb{Z} \cap [m-k, m-1]$. Then $G(x+1) - G(x) = (f(x+k+1) - f(x+k)) - (f(x+1) - f(x)) \leq 0$, that is, G is non-increasing in $\mathbb{Z} \cap [m-k, m]$. Hence M as claimed exists (and satisfies $m - k \leq M \leq m$).

Of course, analoguous statements hold for functions defined only on even or odd integers.

The following result says that a single odd split has an influence of exactly one on another vertex over infinite time.

LEMMA 6.2. *For all $x \in \mathbb{Z} \setminus \{0\}$, $\sum_{t\in\mathbb{N}} |\mathrm{INF}(x,t)| = 1$.*

Proof. W.l.o.g., let $x \in \mathbb{N}$. Then $|\mathrm{INF}(x,t)| = \frac{1}{2}H(x-1,t-1) - \frac{1}{2}H(x+1,t-1)$. Consider a random walk of a single chip started at zero. Let $X_{y,t}$ be the indicator random variable for the event that the chip is on vertex y at time t. Let $Y_{y,t}$ be the indicator random variable for the event that the chip is on vertex y at time t and that it has not visited vertex x so far. Let T denote the first time the chip arrives at x.

For any $t > s > 0$ we have by symmetry that $\Pr(X_{x-1,t-1} = 1 | T = s) = \Pr(X_{x+1,t-1} = 1 | T = s)$. Clearly, for $t \leq T$, $X_{x+1,t-1} = 0$, and for $t > T$, $Y_{x-1,t-1} = 0$. Thus

$$\begin{aligned}&\sum_{t\in\mathbb{N}} |\mathrm{INF}(x,t)|\\ &= \tfrac{1}{2}\sum_{t\in\mathbb{N}} (E(X_{x-1,t-1}) - E(X_{x+1,t-1}))\\ &= \tfrac{1}{2}\sum_{s\in\mathbb{N}} \Pr(T=s) \sum_{t\in\mathbb{N}} E((X_{x-1,t-1} - X_{x+1,t-1}) \,|\, T=s)\\ &= \tfrac{1}{2}\sum_{s\in\mathbb{N}} \Pr(T=s) \sum_{t\in[s]} E(X_{x-1,t-1} \,|\, T=s)\\ &= \tfrac{1}{2}\sum_{s\in\mathbb{N}} \Pr(T=s) E\Big(\sum_{t\in[s]} X_{x-1,t-1} \,|\, T=s\Big)\\ &= \tfrac{1}{2}\sum_{s\in\mathbb{N}} \Pr(T=s) E\Big(\sum_{t\in\mathbb{N}} Y_{x-1,t-1} \,|\, T=s\Big)\\ &= \tfrac{1}{2} E\Big(\sum_{t\in\mathbb{N}} Y_{x-1,t-1}\Big).\end{aligned}$$

Note that $E(\sum_{t\in\mathbb{N}} Y_{x-1,t-1})$ is just the expected number of visits to $x-1$ before visiting x. This number of visits is exactly k if and only if the chip moves left after each of its first $k-1$ visits and right after the kth visit. This happens with probability 2^{-k}. Hence $E(\sum_{t\in\mathbb{N}} Y_{x-1,t-1}) = \sum_{i\in\mathbb{N}} i2^{-i} = 2$.

Proof. [Proof of Theorem 6.1] Fix any even initial configuration. Let $t_0 \in \mathbb{N}_0$ and $S = [t_0 .. t_0 + T - 1]$.

Without loss, let $x = 0$. By (3.4), we have

$$\begin{aligned} & f(0,S) - E(0,S) \\ &= \sum_{t\in S}(f(0,t) - E(0,t)) \\ &= \sum_{y\in\mathbb{Z}} \mathrm{ARR}(y,0) \sum_{i\geq 0}(-1)^i \sum_{t\in S} \mathrm{INF}(y, t - s_i(y)). \end{aligned}$$

By unimodality of rolling sums (Lemma 6.1),

$$|f(0,S) - E(0,S)| \leq \sum_{y\in\mathbb{Z}} \max_{s\in\mathbb{N}} \left| \sum_{t\in S} \mathrm{INF}(y, t-s) \right|.$$

We estimate the term $\max_{s\in\mathbb{N}} |\sum_{t\in S} \mathrm{INF}(y,t-s)|$ for all y. For $1 \leq |y| \leq T^{1/2}$, we use Lemma 6.2 and simply estimate

$$(6.6) \quad \max_{s\in\mathbb{N}} \left| \sum_{t\in S} \mathrm{INF}(y, t-s) \right| \leq \sum_{t\in\mathbb{N}} |\mathrm{INF}(y,t)| = 1.$$

For $|y| > T^{1/2}$, $\max_{s\in\mathbb{N}} \left| \sum_{t\in S} \mathrm{INF}(y,t-s) \right| \leq T \max_{t\in\mathbb{N}} |\mathrm{INF}(y,t)| = TO(y^{-2})$ by Lemma 4.2. Hence

$$\begin{aligned} |f(0,S) - E(0,S)| &\leq \sum_{1\leq|y|\leq T^{1/2}} 1 + T \sum_{|y|>T^{1/2}} O(y^{-2}) \\ &= O(T^{1/2}). \end{aligned}$$

For the lower bound, we invoke the parity forcing theorem again. By this, there is an even initial configuration such that all arrows point towards zero, and such that there is an odd number of chips on vertex $x \in \mathbb{Z}$ at time $t \in \mathbb{N}_0$ if and only if $x \in X := [\sqrt{T} .. 2\sqrt{T}]$ and $t = 4T - x^2$. For this initial configuration and $S = [4T+1 .. 5T]$, we compute

$$\begin{aligned} & |f(0,S) - E(0,S)| \\ &= \sum_{t\in S} \sum_{y\in X} |\mathrm{INF}(y, t - 4T + y^2)| \\ &\geq (1 - o(1))T^{3/2} \\ &\quad \min\left\{ |\mathrm{INF}(y,t)| \,\middle|\, y \in X, t \in [T+1 .. 5T], y \sim t \right\} \\ &= \Omega(T^{1/2}). \end{aligned}$$

7 Space-Time-Intervals

We now regard the discrepancy in space-time-intervals. Extending the previous notation, for finite $X \subseteq \mathbb{Z}$ and finite $S \subseteq \mathbb{N}_0$ set

$$\begin{aligned} f(X,S) &:= \sum_{x\in X} \sum_{t\in S} f(x,t), \\ E(X,S) &:= \sum_{x\in X} \sum_{t\in S} E(x,t). \end{aligned}$$

THEOREM 7.1. *Let $X \subseteq \mathbb{Z}$ and $S \subseteq \mathbb{N}_0$ be finite intervals of lengths L and T, respectively. Then the maximal discrepancy $|f(X,S) - E(X,S)|$ (taken over all odd or even initial configurations) is $\Theta(T \log(LT^{-1/2}))$, if $L \geq T^{1/2}$, and $\Theta(LT^{1/2})$ otherwise.*

Proof. Fix an even initial configuration. Without loss of generality, let $X = [-L+1..0]$. Let $t_0 \in \mathbb{N}_0$ and $S = [t_0 .. t_0 + T - 1]$. As in previous proofs, by (3.4) we have $f(X,S) - E(X,S) = \sum_{y\in\mathbb{Z}} \mathrm{CON}(y)$ with

$$\begin{aligned} & \mathrm{CON}(y) := \\ & \mathrm{ARR}(y,0) \sum_{i\geq 0}(-1)^i \sum_{x\in X} \sum_{t\in S} \mathrm{INF}(y - x, t - s_i(y)). \end{aligned}$$

Case 1: Let $L \geq T^{1/2}$.

We analyze first the contribution of vertices $y \in \mathbb{N}$. Let $y \geq L$. Then

$$\begin{aligned} & |\mathrm{CON}(y)| \\ &= \left| \sum_{i\geq 0}(-1)^i \sum_{x\in X} \sum_{t=t_0}^{t_0+T-1} \mathrm{INF}(y - x, t - s_i(y)) \right| \\ &\leq \sum_{j=0}^{L-1} \left| \sum_{i\geq 0}(-1)^i \sum_{t=t_0}^{t_0+T-1} \mathrm{INF}(y + j, t - s_i(y)) \right| \\ &\leq \sum_{j=0}^{L-1} \max_s \left| \sum_{t=t_0}^{t_0+T-1} \mathrm{INF}(y + j, t - s) \right| \\ &\leq LT \max_t \max_{j\in[0..L-1]} |\mathrm{INF}(y + j, t)| \\ &= LTO(y^{-2}) \end{aligned}$$

by unimodality of rolling sums, Lemma 4.1 and 4.2.

Now let $T^{1/2} \leq y \leq L$. Let $H^-(x,t) := \max\{H(x-1,t), H(x,t)\}$. In other words, $H^-(x,t) = H(x,t)$, if $x \sim t$, and $H^-(x,t) = H(x-1,t)$ otherwise. Similarly, let $H^+(x,t) := \max\{H(x,t), H(x+1,t)\}$. Then, by (3.2), total unimodularity of rolling sums, Lemma 4.1

and 5.1,

$$
\begin{aligned}
&|\mathrm{CON}(y)| \\
&= \Big| \sum_{i\geq 0} (-1)^i \sum_{x\in X} \sum_{t=t_0}^{t_0+T-1} \mathrm{INF}(y-x, t-s_i(y)) \Big| \\
&= \Big| \tfrac{1}{2} \sum_{i\geq 0} (-1)^i \sum_{t=t_0}^{t_0+T-1} \sum_{x\in X} \big(H(y-x+1, t-s_i(y)-1) \\
&\quad - H(y-x-1, t-s_i(y)-1) \big) \Big| \\
&= \Big| \tfrac{1}{2} \sum_{i\geq 0} (-1)^i \sum_{t=t_0}^{t_0+T-1} (H^-(y+L, t-s_i(y)-1) \\
&\quad - H^+(y-1, t-s_i(y)-1)) \Big| \\
&\leq \Big| \tfrac{1}{2} \sum_{i\geq 0} (-1)^i \sum_{t=t_0}^{t_0+T-1} H^-(y+L, t-s_i(y)-1) \Big| \\
&\quad + \Big| \tfrac{1}{2} \sum_{i\geq 0} (-1)^i \sum_{t=t_0}^{t_0+T-1} H^+(y-1, t-s_i(y)-1) \Big| \\
&\leq \tfrac{1}{2} \max_s \sum_{t=t_0}^{t_0+T-1} H^-(y+L, t-s) \\
&\quad + \tfrac{1}{2} \max_s \sum_{t=t_0}^{t_0+T-1} H^+(y-1, t-s) \\
&\leq \tfrac{1}{2} T \max_t H^-(y+L, t) + \tfrac{1}{2} T \max_t H^+(y-1, t) \\
&= TO(y^{-1}).
\end{aligned}
$$

Finally let $y \leq T^{1/2}$. Now starting as above

$$
\begin{aligned}
&|\mathrm{CON}(y)| \\
&\leq \tfrac{1}{2} \max_s \sum_{t=t_0}^{t_0+T-1} H^-(y+L, t-s) \\
&\quad + \tfrac{1}{2} \max_s \sum_{t=t_0}^{t_0+T-1} H^+(y-1, t-s) \\
&\leq \max_s \sum_{t=t_0}^{t_0+T-1} H(0, t-s) \\
&\leq \sum_{t=0}^{T-1} H(0,t) \\
&= O(T^{1/2}).
\end{aligned}
$$

Putting all this together, we obtain

$$
\begin{aligned}
&\Big| \sum_{y\in\mathbb{N}} \mathrm{CON}(y) \Big| \\
&\leq \sum_{y\leq T^{1/2}} |\mathrm{CON}(y)| + \sum_{T^{1/2}<y\leq L} |\mathrm{CON}(y)| + \sum_{L<y} |\mathrm{CON}(y)| \\
&= \sum_{y\leq T^{1/2}} O(T^{1/2}) + \sum_{T^{1/2}<y\leq L} TO(y^{-1}) + \sum_{L<y} LTO(y^{-2}) \\
&= O(T) + O(T\log(LT^{-1/2})) + O(T) \\
&= O(T\log(LT^{-1/2})).
\end{aligned}
$$

By symmetry, we have the same bounds for $\mathrm{CON}(y)$ if $y \leq -L$. Using the same resoning as in the proof of Theorem 5.1, vertices $y \in X$ contribute at most as much as those not in X.

Altogether, we have

$$
\begin{aligned}
\sum_{y\in\mathbb{Z}} |\mathrm{CON}(y)| &= O\Big(\sum_{y\in\mathbb{N}} |\mathrm{CON}(y)| \Big) \\
&= O(T\log(LT^{-1/2})).
\end{aligned}
$$

We now prove the corresponding lower bound. Set $Y = [T^{1/2}..L]$. Choose an even initial configuration such that $f(x,t)$ is odd if and only if $x \in Y$ and $t = L^2 - x^2$. Direct all arrows towards zero. Let $S = [L^2..L^2+T-1]$. Then for $y \in Y$, with $\delta_t, \varepsilon_t \in \{0,1\}$ appropriately chosen as before,

$$
\begin{aligned}
&\mathrm{CON}(y) \\
&= \sum_{x\in X} \sum_{t=L^2}^{L^2+T-1} |\mathrm{INF}(y-x, t-(L^2-y^2))| \\
&\geq \tfrac{1}{2} \sum_{t=y^2}^{y^2+T-1} (H(y-1+\varepsilon_t, t-1) - H(y+L-1-\delta_t, t-1)) \\
&\geq \Omega\Big(\sum_{t=y^2}^{y^2+T-1} H(y-1+\varepsilon_t, t-1) \Big) \\
&= \Omega(Ty^{-1}).
\end{aligned}
$$

For $y \notin Y$, $\mathrm{CON}(y) = 0$. Hence the discrepancy in this setting is at least $\sum_{y\in Y} \mathrm{CON}(y) = \sum_{y=T^{1/2}}^{L} O(Ty^{-1}) = O(T\log(LT^{-1/2}))$.

Case 2: Let $L \leq T^{1/2}$.

For the upper bound, we simply apply Theorem 6.1 as follows:

$$
\begin{aligned}
|f(X,S) - E(X,S)| &\leq \sum_{x\in X} |f(x,S) - E(x,S)| \\
&\leq LO(T^{1/2}).
\end{aligned}
$$

For the lower bound, the setting of Theorem 6.1 works as well. Choose an initial configuration such that $f(x,t)$ is odd if and only if $x \in X := [T^{1/2}..2T^{1/2}]$ and $t = 4T - y^2$. Then

$$\begin{aligned}
&\mathrm{CON}(y)\\
&= \sum_{x\in X}\sum_{t=4T}^{5T-1} |\mathrm{INF}(y-x, t-(4T-y^2))|\\
&\geq LT \min\{|\mathrm{INF}(y,t)| \mid y \in \mathbb{Z}\cap[T^{1/2}..3T^{1/2}],\\
&\quad t\in\mathbb{Z}\cap[T..5T], y\sim t\}\\
&= \Omega(L)
\end{aligned}$$

for all $y \in Y$. Again, $\mathrm{CON}(y) = 0$ for $y \notin Y$. Hence $\sum_{y\in\mathbb{Z}} \mathrm{CON}(y) = \Omega(LT^{1/2})$.

8 Intervals in Space, Revisited

We stated in Theorem 5.1 that the discrepancy in an interval of length L is $O(\log L)$. Here we state that intervals of length L with about $\log L$ discrepancy are very rare, the quadratic average of the discrepancies of a long contiguous set of intervals of length L is only $O(\sqrt{\log L})$, and this bound is tight.

For a set X of vertices we denote by $\mathrm{DISC}(X,t)$ the discrepancy of the set X at time t, i.e., we set $\mathrm{DISC}(X,t) = f(X,t) - E(X,t)$.

THEOREM 8.1. *Let X be an interval of length L. For M sufficiently large,*

$$\frac{1}{M}\sum_{k=1}^{M} \mathrm{DISC}^2(X+k,t) = O(\log L).$$

Furthermore, for a given L and M there exists an even initial configuration, and a time t and an interval X of length L such that

$$\frac{1}{M}\sum_{k=1}^{M} \mathrm{DISC}^2(X+k,t) = \Omega(\log L).$$

Proof. For the first statement we need to prove an $O(\sqrt{\log L})$ bound on the quadratic average of the discrepancies $\mathrm{DISC}(X+k,t)$ with $k = 1,\ldots,M$. First note that changing the individual discrepancies by a bounded amount we change the quadratic average by at most the same amount. We use this observation to freely neglect $O(1)$ terms in the discrepancy of the intervals. In particular we can change the intervals themselves by adding or deleting a bounded number of vertices. We use this to make a few simplifying assumptions. As in Section 7 we assume that (i) the starting configuration is odd, (ii) the interval X is $X = [-L'..L']$ with $L' \sim t$, and (iii) M is even and we only consider even values of k, i.e., we consider the average of $\mathrm{DISC}^2(X+k,t)$ for $2 \leq k \leq M$, k even (this can be justified by considering $X+k+1$ instead of $X+k$ for odd k).

First we show that discrepancies caused by odd piles at time $t - L^2$ or before can be neglected. We start with (3.4) for the individual discrepancies $\mathrm{DISC}(x,t)$.

$$\begin{aligned}
&\mathrm{DISC}(x,t)\\
&= \sum_{y\in\mathbb{Z}} \mathrm{ARR}(y,0)\sum_{i\geq 0}(-1)^i \mathrm{INF}(y-x, t-s_i(y))\\
&= \mathrm{DISC}_1(x,t) + \mathrm{DISC}_2(x,t);\\
&\mathrm{DISC}_1(x,t)\\
&= \sum_{y\in\mathbb{Z}} \mathrm{ARR}(y,0) \sum_{s_i(y)>t-L^2} (-1)^i \mathrm{INF}(y-x, t-s_i(y));\\
&|\mathrm{DISC}_2(x,t)|\\
&= |\sum_{y\in\mathbb{Z}} \mathrm{ARR}(y,0) \sum_{s_i(y)\leq t-L^2} (-1)^i \mathrm{INF}(y-x, t-s_i(y))|\\
&\leq \sum_{y\in\mathbb{Z}} \max_{u\geq L^2} |\mathrm{INF}(y-x,u)|.
\end{aligned}$$

We have seen that $|\mathrm{INF}(z,u)|$ is unimodal for fixed z and its maximum is at $u = \lceil z^2/3\rceil$, so we have

$$\begin{aligned}
&|\mathrm{DISC}_2(x,t)|\\
&\leq 2\sum_{z=1}^{L} |\mathrm{INF}(z,L^2)| + 2\sum_{z>L} |\mathrm{INF}(z,\lceil z^2/3\rceil)|\\
&\leq 2\sum_{z=1}^{L} \frac{z}{L^2} H(z,L^2) + 2\sum_{z>L} O(z^{-2})\\
&\leq 2\sum_{z=1}^{L} H(z,L^2)/L + O(1/L) = O(1/L).
\end{aligned}$$

Therefore the total contribution of DISC_2 to the discrepancy of an interval $X+k$ is small. For

$$\mathrm{DISC}_1(X+k,t) := \sum_{x\in X+k} \mathrm{DISC}_1(x,t)$$

we have

$$\begin{aligned}
&|\mathrm{DISC}_1(X+k,t) - \mathrm{DISC}(X+k,t)|\\
&= |\sum_{x\in X+k} \mathrm{DISC}_2(x,t)|\\
&= O(1).
\end{aligned}$$

We continue as in Section 7 collapsing a sum using $\mathrm{INF}(z,u) = \frac{1}{2}H(z+1,u-1) - \frac{1}{2}H(z-1,u-1)$. We also use that $\mathrm{DISC}(x,t) = \mathrm{DISC}_1(x,t) = 0$ for $x \sim t$ as

the starting configuration is odd.

$$\begin{aligned}
&\text{DISC}_1(X+k,t)\\
&= \sum_{\substack{x\in X+k\\ x\sim t+1}} \sum_{y\in\mathbb{Z}} \text{ARR}(y,0)\\
&\quad \sum_{s_i(y)>t-L^2} (-1)^i \text{INF}(y-x,t-s_i(y))\\
&= \sum_{y\in\mathbb{Z}} \text{ARR}(y,0) \sum_{s_i(y)>t-L^2} (-1)^i\\
&\quad \sum_x (\tfrac{1}{2}H(y-x+1,t-s_i(y)-1)\\
&\quad -\tfrac{1}{2}H(y-x-1,t-s_i(y)-1))\\
&= \tfrac{1}{2}\sum_{y\in\mathbb{Z}} \text{ARR}(y,0) \sum_{s_i(y)>t-L^2} (-1)^i\\
&\quad (H(y-k+L',t-s_i(y)-1)\\
&\quad -H(y-k-L',t-s_i(y)-1)).
\end{aligned}$$

We separate the two terms in this last expression. With

$$\begin{aligned}
D(m) &:= 2\sum_{y\in\mathbb{Z}} \text{ARR}(y,0)\\
&\quad \sum_{s_i(y)>t-L^2} (-1)^i H(y-m,t-s_i(y)-1)
\end{aligned}$$

we have

$$\text{DISC}_1(X+k,t) = \frac{1}{4}D(k-L') - \frac{1}{4}D(k+L').$$

Our original goal was to prove an $O(\sqrt{\log L})$ bound on the quadratic average of $\text{DISC}(X+k,t)$. As $\text{DISC}_1(X+k,t)$ differs from $\text{DISC}(X+k,t)$ by $O(1)$ it is clearly enough to prove the same bound for the quadratic average of $\text{DISC}_1(X+k,t)$. By the last displayed formula it is enough to prove the $O(\sqrt{\log L})$ bound on the two parts $D(k-L')$ and $D(k+L')$ separately, both for $0<k\le M$ even. It is therefore enough to bound the quadratic average of $D(m)$ for an arbitrary interval I of length M. Here we consider only values $m\sim t$, for other values of m we have $D(m)=0$.

Let $t_0=\max(0,t-L^2+1)$ be the first time-step considered. For $y\in\mathbb{Z}$ and $u\sim y+1$ we have an odd pile at y if and only if $\text{ARR}(y,u)\neq\text{ARR}(y,u+2)$ and in this case $\text{ARR}(y,u)=(-1)^i\text{ARR}(y,0)$ for the index i with $s_i(y)=u$. For any $m\sim t$ we have

$$\begin{aligned}
&2\text{ARR}(y,0) \sum_{s_i(y)>t-L^2} (-1)^i H(y-m,t-s_i(y)-1)\\
&= \sum_{\substack{t_0\le u<t\\ u\sim y+1}} (\text{ARR}(y,u)-\text{ARR}(y,u+2))\\
&\quad H(y-m,t-u-1)\\
&= \sum_{\substack{t_0+2\le u<t\\ u\sim y+1}} \text{ARR}(y,u)H(y-m,t-u-1)\\
&\quad -\text{ARR}(y,u)H(y-m,t-u+1)\\
&\quad +\text{ARR}(y,t_1)H(y-m,t-t_1-1)\\
&\quad -\text{ARR}(y,t_2)H(y-m,t-t_2+1),
\end{aligned}$$

where $t_1=t_1(y)$ is either t_0 or t_0+1, whichever makes $t_1\sim y+1$ and similarly $t_2=t_2(y)$ is either t or $t+1$, so that $t_2\sim y+1$. With

$$\begin{aligned}
D'(m) :=& \sum_{y\in\mathbb{Z}} \sum_{\substack{t_0+2\le u<t-2\\ u\sim y+1}} \text{ARR}(y,u)\\
&(H(y-m,t-u-1)-H(y-m,t-u+1))
\end{aligned}$$

we have

$$\begin{aligned}
&|D(m)-D'(m)|\\
&= |\sum_{y\in\mathbb{Z}} (\text{ARR}(y,t_1(y))H(y-m,t-t_1(y)-1)\\
&\quad -\text{ARR}(y,t_2(y))H(y-m,t-t_2(y)+1))|\\
&\le \sum_{u\in\{0,1,t-t_0-2,t-t_0-1\}} \sum_{y\in\mathbb{Z}} H(y-m,u) \le 4.
\end{aligned}$$

As before, we ignore the small difference and will prove the $O(\sqrt{\log L})$ bound on the quadratic average of $D'(m)$ instead of $D(m)$. Computing the square and summing for m we get the following. The summations are taken for $m\in I$, $m\sim t$, for $y,y_2\in\mathbb{Z}$, and for $u_1,u_2\in[t_0+2,t-3]$, $u_1\sim u_2\sim y+1$, respectively.

$$\begin{aligned}
&\sum_m D'^2(m)\\
&= \sum_{y_1,y_2} \sum_{u_1,u_2} \text{ARR}(y_1,u_1)\text{ARR}(y_2,u_2)\\
&\quad \cdot\sum_m (H(y_1-m,t-u_1-1)-H(y_1-m,t-u_1+1))\\
&\quad \cdot(H(y_2-m,t-u_2-1)-H(y_2-m,t-u_2+1)).
\end{aligned}$$

Let us estimate the contribution to this sum coming from a fixed y_1, u_1, and u_2. Disregarding the signs and extending the summation for all m (even outside I) the contribution of each of the four terms we get from the multiplication is exactly 1. As u_1 and u_2 can take at

most $L^2/2$ values each, the total contribution coming from a single value of y_1 is at most L^4.

Let us obtain the intervals I' and I'' from I by extending or shortening it at both ends by L^2, i.e., if $I = [a, b]$, then $I' = [a - L^2, b + L^2]$, $I'' = [a + L^2, b - L^2]$. If y_1 is outside I' we have $H(y_1 - m, t - u_1 - 1) = H(y_1 - m, t - u_1 + 1) = 0$ for all $m \in I$, therefore such y_1 has zero contribution to $\sum_m D'^2(m)$. The contribution for fixed y_1, y_2, u_1, and u_2 can usually be written in closed form using the identity

$$\sum_m H(y_1 - m, v_1)H(y_2 - m, v_2) = H(y_1 - y_2, v_1 + v_2).$$

This identity works if we sum for all possible values of m, but for $y_1 \in I''$ the contribution of the values $m \notin I$ is zero. Therefore the contribution to $\sum_m D'^2(m)$ of the fixed terms $y_1 \in I''$, y_2, u_1, and u_2 is

$$\begin{aligned}
&\mathrm{ARR}(y_1, u_1)\mathrm{ARR}(y_2, u_2) \\
&\sum_m (H(y_1 - m, t - u_1 - 1) - H(y_1 - m, t - u_1 + 1)) \\
&\cdot (H(y_2 - m, t - u_2 - 1) - H(y_2 - m, t - u_2 + 1)) \\
=&\mathrm{ARR}(y_1, u_1)\mathrm{ARR}(y_2, u_2) \\
&(H(y, v - 2) - 2H(y, v) + H(y, v + 2)),
\end{aligned}$$

where $y = y_1 - y_2$ and $v = 2t - u_1 - u_2$.

To estimate these contributions we first calculate

$$\begin{aligned}
&H(y, v - 2) - 2H(y, v) + H(y, v + 2) \\
&= O(y^4/v^4 + 1/v^2)H(y, v + 2).
\end{aligned}$$

The same $y = y_1 - y_2$ value arises exactly once for every $y_1 \in I''$, a total of $M - 2L^2$ possibility. The highest possible value of v is less than $2L^2$ and the same value v can be the result of at most v pairs u_1, u_2. There are $4L^2$ possible values of y_1 outside I'' but inside I' contributing at most $4L^6$. Summing for all these contributions we estimate

$$\begin{aligned}
&\sum_m D'^2(m) \\
&\leq 4L^6 + O\left(\sum_{v=1}^{2L^2} Mv \sum_{y \in \mathbb{Z}} (y^4/v^4 + 1/v^2)H(y, v + 2)\right) \\
&= 4L^6 + O\left(M \sum_{v=1}^{2L^2} \sum_{y \in \mathbb{Z}} (y^4/v^3 + 1/v)H(y, v + 2)\right) \\
&= 4L^6 + O\left(M \sum_{v=1}^{2L^2} 1/v\right) = O(L^6 + M \log L).
\end{aligned}$$

Here we used the estimate on the fourth moment of the random walk:

$$\sum_{y \in \mathbb{Z}} y^4 H(y, v + 2) = O((v + 2)^2) = O(v^2).$$

To finish the proof we set $M > L^6$ for our threshold for large enough M. We did not make an effort to optimize for this threshold. This ensures that $\sum_m D'^2(m) = O(M \log L)$, so the quadratic average of $D'(m)$ (and therefore of $\mathrm{DISC}(X + k, t)$) is $O(\sqrt{\log L})$ as claimed.

It remains to construct a starting configuration where the quadratic average of discrepancies in the intervals of length L is large. For our construction we do not even use the value L. For a given (even) parameter t we define a probability distribution on starting positions, such that for all $L < t$ and all intervals X of length L the expectation of $\mathrm{DISC}^2(X, t) = \Omega(\log L)$.

We let $r(a, b)$ stand for independent random ± 1 variables for all integers a and $b \geq 1$. We look for an even starting configuration (guaranteed by the Arrow-forcing Theorem, such that $\mathrm{ARR}(x, u) = r(a, b)$ for all even x and u satisfying $4^b < u \leq 4^{b+1}$ and $a2^b < x \leq (a + 1)2^b$. For simplicity we set $\mathrm{ARR}(x, u) = 1$ for all u and all odd x and we also set $\mathrm{ARR}(x, u) = 1$ for all x and $u \leq 4$.

Simple calculation similar to the one in Section 7 shows that for an interval $X = [c, d]$ we have

$$\mathrm{DISC}(X, t) = \sum_{a,b} h(a, b)r(a, b),$$

where the coefficients $h(a, b)$ depend on X. Further analysis shows that all coefficients are bounded and $\Theta(\log L)$ of them are above a positive absolute constant for each interval of length L. This implies that the expectation of $\mathrm{DISC}^2(X, t)$ is $\Omega(\log L)$, and therefore the expectation of the average $\frac{1}{M}\sum_{k=1}^{M} \mathrm{DISC}^2(X + k, t)$ is also $\Omega(\log L)$. This proves the second statement of the theorem.

9 Conclusion

In this paper, we elaborated that Jim Propp's rotor router model is a very good simulation for random walks on $\mathbb{Z}$. Not only that there is a constant bound for errors on single vertices, these bounds become stronger in several forms of averages.

We are optimistic that similar results hold for higher-dimensional grids. For errors on single vertices, this was shown by Cooper and Spencer [CS05, CS]. For higher-dimensional intervals, that is, products of intervals, we note that interesting things mainly happen along the boundary. Hence we expect the discrepancy in a product of d intervals of length L to be of order $L^{d-1} \log L$. Note that the arrow-forcing theorem can be extended to higher-dimensional grids, so our lower bound proofs can be extended as well.

From this work a number of open problems arise. One is the following: The arrow/parity-forcing theorem yields a simple, yet general way to produce lower bounds. It is, however, quite wasteful in the number of chips. To obtain a desired behavior for t time steps, we will have $\Theta(2^t)$ chips on many vertices. For actual simulations, this is not desirable. Therefore, it seems to be an interesting problem to look for sparser setups that still produce large discrepancies.

References

[CS] J. Cooper and J. Spencer. Simulating a Random Walk with Constant Error. `arXiv:math.CO/0402323`.

[CS05] J. Cooper and J. Spencer. Simulating a random walk with constant error. *Combinatorics, Probability and Computing*, 2005. To appear.

[Kle05] M. Kleber. Goldbug variations. *Mathematical Intelligencer*, 27, 2005.

[LP] L. Levine and Y. Peres. Spherical Asymptotics for the Rotor-Router Model in $\mathbb{Z}^d$. `arXiv:math.PR/0503251`.

Binary Trees, Left and Right Paths, WKB Expansions, and Painlevé Transcendents*

Charles Knessl† Wojciech Szpankowski‡

Abstract

During the *10th Seminar on Analysis of Algorithms*, MSRI, Berkeley, June 2004, Knuth posed the problem of analyzing the left and the right path length in a random binary trees. In particular, Knuth asked about properties of the generating function of the joint distribution of the left and the right path lengths. In this paper, we mostly focus on the asymptotic properties of the distribution of the difference between the left and the right path lengths. Among other things, we show that the Laplace transform of the appropriately normalized moment generating function of the path difference satisfies the *first Painlevé transcendent*. This is a nonlinear differential equation that has appeared in many modern applications, from nonlinear waves to random matrices. Surprisingly, we find out that the difference between path lengths is of the order $n^{5/4}$ where n is the number of nodes in the binary tree. This was also recently observed by Marckert and Janson. We present precise asymptotics of the distribution's tails and moments. We shall also discuss the joint distribution of the left and right path lengths. Throughout, we use methods of analytic algorithmics such as generating functions and complex asymptotics, as well as methods of applied mathematics such as the WKB method.

1 Introduction

Trees are the most important nonlinear structures that arise in computer science. Applications are in abundance; here we discuss binary unlabeled ordered trees (further called binary trees) and study their asymptotic properties when the number of nodes, n, becomes large. While various interesting questions concerning statistics of randomly generated binary trees were investigated since Euler and Cayley [8, 17, 18, 25, 27, 28], recently novel applications have been surfacing. In 2003 Seroussi [22], when studying universal types for sequences and Lempel-Ziv'78 parsings, asked for the number of binary trees of *given* path length (sum of all paths from the root to all nodes). This was an open problem; partial solutions are reported in [15, 23].

During the *10th Seminar on Analysis of Algorithms*, MSRI, Berkeley, June 2004, Knuth asked to analyze the joint distribution of the left and the right path lengths in random binary trees. This problem received a lot of attention in the community (cf. related papers [11, 20]) and leads to an interesting analysis, that encompasses several other problems studied recently [11, 15, 19, 20, 22, 23, 27]. Here, we mostly focus on the asymptotic properties of the distribution of the difference between the left and the right path lengths. However, we also obtain some results for the joint distribution of the left and the right path lengths in a random binary tree.

In the standard model, that we adopt here, one selects uniformly a tree among all binary unlabeled ordered trees built on n nodes, $\mathcal{T}_n$ (where $|\mathcal{T}_n| = \binom{2n}{n}\frac{1}{n+1}$ =Catalan number). Many deep and interesting results concerning the behavior of binary trees in the standard model were uncovered. For example, Flajolet and Odlyzko [6] and Takacs [27] established the average and the limiting distribution for the height (longest path), while Louchard [19] and Takacs [26, 27, 28] derive the limiting distribution for the path length. As we indicate below, these limiting distributions are expressible in terms of the Airy's function (cf. [2, 3]). Recently, Seroussi [22, 23], and Knessl and Szpankowski [15] analyzed properties of random binary trees when selected uniformly from the set $\mathcal{T}_t$ of all binary trees of given path length t. Among other results, they enumerated the number of trees in $\mathcal{T}_t$ and analyze the number of nodes in a randomly selected tree from $\mathcal{T}_t$.

We now summarize our main results and put them into a bigger perspective. Let $N_n(p,q)$ be the number of binary trees built on n nodes with the right path length equal to p and the left path length equal to q. It is easy

*The work was supported by NSF Grants CCR-0208709 and DMS 05-03745, NIH Grant R01 GM068959-01, and NSA Grant MDA 904-03-1-0036

†Dept. Math., Stat. & Computer Science, University of Illinois at Chicago, Chicago, Illinois 60607-7045, U.S.A

‡Department of Computer Science, Purdue University, West Lafayette, IN 47907-2066 U.S.A.

to see that its generating function $G_n(w,v)$ satisfies

$$G_{n+1}(w,v) = \sum_{i=0}^{n} w^i v^{n-i} G_i(w,v) G_{n-i}(w,v) \tag{1.1}$$

with $G_0(w,v) = 1$. Summing over n we obtain the triple transform $C(w,v,z)$ (cf. also (2.12) below) that satisfies

$$C(w,v,z) = 1 + zC(w,v,wz)C(w,v,vz). \tag{1.2}$$

This is exactly the equation that Knuth asked to analyze.

The above functional equation encompasses many properties of binary trees. Let us first set $w = v$ and define $C(w,z) = C(w,w,z)$. Recurrence (1.2) then becomes

$$C(w,z) = 1 + zC^2(w,zw). \tag{1.3}$$

Observe that this equation is asymmetric with respect to z and w. When enumerating trees in $\mathcal{T}_n$, we set $w = 1$ to get the well known algebraic equation $C(1,z) = 1 + zC^2(1,z)$ that can be explicitly solved as $C(1,z) = \left(1 - \sqrt{1-4z}\right)/(2z)$, leading to the Catalan number. A randomly (uniformly) selected tree from $\mathcal{T}_n$ has path length T_n that is asymptotically distributed as Airy's distribution [26, 27], that is,

$$\Pr\{T_n/\sqrt{2n^3} \le x\} \to W(x)$$

where $W(x)$ is the Airy distribution function defined by its moments [7]. The Airy distribution arises in surprisingly many contexts, such as parking allocations, hashing tables, trees, discrete random walks, area under a Brownian bridge, etc. [7, 19, 26, 27, 28].

Setting $z = 1$ in (1.3) we arrive at

$$C(w,1) = 1 + C^2(w,w)$$

which does not seem to be explicitly solvable, e.g., by using a Taylor expansion in w. Observe that the coefficient of w^t in $C(w,1)$ enumerates binary trees with a *given* path length t. In [15, 23] it was shown that

$$[w^t]C(w,1) = |\mathcal{T}_t| =$$

$$= \frac{1}{(\log_2 t)\sqrt{\pi t}} 2^{\frac{2t}{\log_2 t}\left(1 + c_1 \log^{-2/3} t + c_2 \log^{-1} t + O(\log^{-4/3} t)\right)}$$

for large t, where c_1 and c_2 are explicitly computable constants.

Let us now set $v = 1$ in (1.2). Then

$$C(w,1,z) = 1 + zC(w,1,wz)C(w,1,z)$$

while $G_n(w,1) = [z^n]C(w,1,z)$ satisfies

$$G_{n+1}(w,1) = \sum_{i=1}^{n} w^i G_i(w,1) G_{n-i}(w,1). \tag{1.4}$$

Observe that $G_n(w,1)$ is the generating function of the right path length. Actually, recurrence (1.4) was studied by Takacs in [26] when analyzing the area under a Bernoulli random walk. Also, it appears in the Kleitman-Winston conjecture [13, 29].

It turns out that there is a close link between the path length difference in trees and the center of mass S of the Integrated SuperBrownian Excursion (ISE), which was introduced by Aldous in [1]. The ISE is also related to the Brownian snake, which has various applications in branching processes, random maps and lattice trees (see the survey in [24]). Recent work on this problem, and its connection to the path difference in trees, is due to Marckert [20], Janson [10, 11], and Janson and Chassaing [12].

In particular moments of the path length difference were recently analyzed by Janson [11] using a Galton-Watson branching process approach, and the limiting distribution is implicit in Marckert [20], who applied Brownian analysis. As pointed out by Janson [11] "many different methods are useful and valuable, even for the same types of problems, and should be employed without prejudice".

Finally, we turn attention to results presented in this paper. We first analyze the limiting distribution of the difference $\mathcal{D}_n = \mathcal{R}_n - \mathcal{L}_n$ where $\mathcal{R}_n, \mathcal{L}_n$ are the right and left path lengths, respectively. We observe that the difference $\mathcal{D}_n$ is of order $n^{5/4}$. This was also recently observed by Marckert [20] and Janson [11]. Among other things, we show that the tail of the distribution is thicker than that of the Gaussian distribution.

Next, we analyze moments of $\mathcal{D}_n$. We first observe that odd moments vanish, while the *normalized* even moments satisfy (asymptotically) a certain *nonlinear* recurrence that occurs in various forms in many other problems, that are described by nonlinear functional equations similar to (1.1) (e.g., quicksort, linear hashing, enumeration of trees in $\mathcal{T}_t$). In these cases, usually the limiting distribution can be characterized only by moments. We conjecture that these problems constitute a new class of distributions determined by moments. More precisely, let Z be a (normalized) limiting distribution of such a process. Then for some $a_m \to \infty$ we have $\mathbf{E}[Z^m]/a_m = c_m$ such that in general c_m satisfies

$$c_{m+1} = \alpha_m + \beta_m c_m + \gamma_m \sum_{i=0}^{m} c_i c_{m-i} \tag{1.5}$$

with some initial conditions, and given α_m, β_m and γ_m. In our case (cf. also [11] and [12]) the even normalized moments of $\mathcal{D}_n$ converge as $\mathbf{E}[\mathcal{D}_n^{2m+2}]/n^{5(m+1)/2} \to (2m+2)!\sqrt{\pi}\Delta_m$ for any integer $m \ge 0$, where Δ_m satisfy the recurrence similar to (1.5) (cf. (2.23) and (2.24)

below). Similar recurrences appear in the quicksort [14], linear hashing [7], path length in binary trees [17, 19, 27, 28], area under Bernoulli walk [26], enumeration of trees with given path length [15], and many others [8, 18, 25].

Finally, we analyze the moment generating function of $\mathcal{D}_n$. We shall show that the Laplace transform of an appropriately normalized moment generating function satisfies the *first Painlevé transcendent* nonlinear differential equation [9]. This also appears in many modern applications, including nonlinear waves and random matrices. We shall also discuss the joint distribution of the left and the right path lengths, and this will be the starting point of our analysis.

Throughout, we use methods of analytic algorithmics such as generating functions and complex asymptotics, as well as methods of applied mathematics, such as the WKB method [4]. In this approach we make some assumptions about the forms of asymptotic expansions, for which we provide extensive numerical back-up.

2 Problem Statement and Summary of Results

Here we give a detailed summary of our main results; the technical derivations appear in [16].

Let $N(p,q;n)$ be the number of binary trees with n nodes that have a total right path length p and a total left path length q. We also set

$$N(p,q;n) = \bar{N}(p+q, p-q; n) \tag{2.1}$$

and note the obvious symmetry relation

$$N(p,q;n) = N(q,p;n). \tag{2.2}$$

We shall mostly focus on analyzing the difference between the right and left path length, and this we denote by

$$J = p - q. \tag{2.3}$$

It is well known [8] that the total number of trees with n nodes is the Catalan number

$$C_n = \frac{1}{n+1}\binom{2n}{n}. \tag{2.4}$$

Then we define the probability distribution of the path length difference, $\mathcal{D}_n$, by

$$\begin{aligned} P_-(J;n) &= \text{Prob}[\mathcal{D}_n = J] \\ &= \frac{1}{C_n}\sum_{i=0}^{\binom{n}{2}-|J|} N(i, i+|J|; n), \end{aligned} \tag{2.5}$$

for $J \in \left[-\binom{n}{2}, \binom{n}{2}\right]$. Here we used the fact that the left or right path in a tree with n nodes can be at most $\binom{n}{2}$. We can easily verify that

$$P_-\left(\binom{n}{2} - 1; n\right) = 0, \tag{2.6}$$

$$\begin{aligned} P_-\left(\binom{n}{2}; n\right) &= P_-\left(\binom{n}{2} - 2; n\right) \\ &= P_-\left(\binom{n}{2} - 3; n\right) = 1/C_n, \end{aligned} \tag{2.7}$$

since there are no trees where the path length difference is one below the maximum, and exactly one tree (out of C_n) has this difference either zero or two or three below the maximum value of $\binom{n}{2}$. In view of (2.2) we have $P_-(J;n) = P_-(-J;n)$ so it is sufficient to analyze (2.5) for $J \geq 0$.

The generating function of $N(p,q)$ in (2.1)

$$G_n(w,v) = \sum_p \sum_q N(p,q;n) w^p v^q \tag{2.8}$$

satisfies the recurrence

$$G_{n+1}(w,v) = \sum_{i=0}^{n} w^i v^{n-i} G_i(w,v) G_{n-i}(w,v), \quad n \geq 0, \tag{2.9}$$

subject to the initial condition

$$G_0(w,v) = 1. \tag{2.10}$$

From (2.9) we also obtain the functional equation

$$C(w,v,z) = 1 + zC(w,v,wz)C(w,v,vz) \tag{2.11}$$

for the triple transform

$$C(w,v,z) = \sum_{n=0}^{\infty} G_n(w,v) z^n = \sum_n \sum_p \sum_q N(p,q;n) z^n w^p v^q. \tag{2.12}$$

We note that $G_n(1,1) = C_n$ and from (2.5) we obtain

$$P_-(J;n) = \frac{1}{C_n}[w^J] G_n\left(w, \frac{1}{w}\right). \tag{2.13}$$

We study the limit $n \to \infty$, with an appropriate scaling of p and q. First we consider the path length difference, with J scaled as

$$J = \beta n^{5/4} = O(n^{5/4}). \tag{2.14}$$

For a fixed β we shall obtain

$$P_-(J;n) \sim n^{-5/4} p_-(\beta) \tag{2.15}$$

where $p_-(\beta)$ is a probability density that can be represented as

$$\begin{aligned} p_-(\beta) &= \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} e^{-\beta b}[1+\sqrt{\pi}\bar{H}(b)]db \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-\beta i x}[1+\sqrt{\pi}\bar{H}(ix)]dx \\ &= \frac{1}{\pi}\int_0^{\infty} \cos(\beta x) S(x) dx, \end{aligned} \tag{2.16}$$

where $S(x) = 1 + \sqrt{\pi}\bar{H}(ix)$ and

$$1 + \sqrt{\pi}\bar{H}(b) = \int_{-\infty}^{\infty} e^{\beta b} p_-(\beta) d\beta. \tag{2.17}$$

Thus the left side of (2.17) is the moment generating function of $p_-(\beta)$, which is an *entire* function of b.

While we do not have an explicit formula for $p_-(\beta)$, we have the following asymptotic and numerical values:

$$p_-(\beta) \sim \sqrt{\frac{5}{6}}(5\beta)^{1/3} c_0 \exp\left(-\frac{3}{4}5^{1/3}\beta^{4/3}\right)[1+O(\beta^{-4/3})] \tag{2.18}$$

as $\beta \to \infty$, with

$$c_0 \approx .5513288; \tag{2.19}$$

$$p_-(0) \approx .45727\ ;\ p_-''(0) \approx -.71462. \tag{2.20}$$

We also find that the density has two inflection points, at $\beta = \pm\beta_c$, with

$$\beta_c \approx .75898. \tag{2.21}$$

This is our first main result. We comment that tails with exponent 4/3 are also observed for problems dealing with the Brownian snake (see [5, 12, 21]).

The function $p_-(\beta)$ is graphed in Figure 1 over the range $\beta \in [0,3]$, and the derivatives $p_-'(\beta)$ and $p_-''(\beta)$ are given over the same range in Figure 2. The graph of $p_-(\beta)$ somewhat resembles a Gaussian, but it differs from the Gaussian density in at least three important respects. First, the tail is clearly thicker in view of (2.18). For any Gaussian density, we would have $\beta_c p_-(0) = 1/\sqrt{2\pi} = .39894\ldots$ while for the present density (2.20) and (2.21) yield the value $\beta_c p_-(0) \approx .34705$. Finally, the Gaussian density would be an entire function of β, while we will show that if we view $p_-(\beta)$

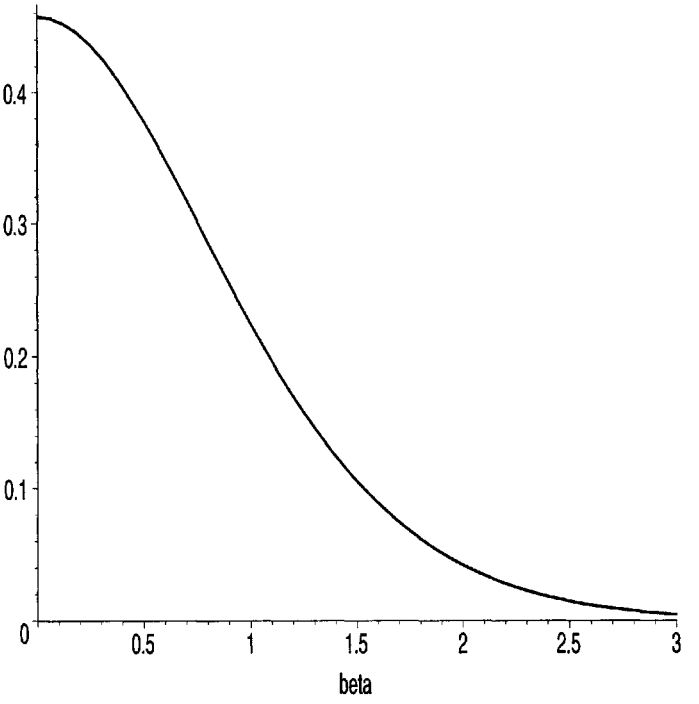


Figure 1: The density $p_-(\beta)$ for $\beta \in [0,3]$.

as a function of the complex variable β, this function has an essential singularity at $\beta = 0$. While for $|\beta|$ small and β real, $p_-(\beta)$ is locally Gaussian, its behavior for $|\beta|$ small and β imaginary is quite different.

The function $\bar{H}$ in (2.16) is an entire function satisfying $\bar{H}(b) = \bar{H}(-b)$ and $\bar{H}(0) = 0$. Denoting its Taylor series as

$$\bar{H}(b) = \sum_{m=0}^{\infty} b^{2m+2}\Delta_m \tag{2.22}$$

and setting

$$\Delta_m = \frac{1}{\Gamma\left(\frac{5}{2}m+2\right)}\tilde{\Delta}_m \tag{2.23}$$

we find that $\tilde{\Delta}_m$ satisfies the nonlinear recurrence

$$\tilde{\Delta}_{m+1} = \frac{(5m+6)(5m+4)}{8}\tilde{\Delta}_m + \frac{1}{4}\sum_{\ell=0}^{m}\tilde{\Delta}_\ell\tilde{\Delta}_{m-\ell},\ m \geq 0 \tag{2.24}$$

with

$$\Delta_0 = \tilde{\Delta}_0 = \frac{1}{4}. \tag{2.25}$$

This recurrence agrees with results in [12]. In view of (2.22) and (2.17) the variance of the limiting density $p_-(\beta)$ in (2.15) is

$$\int_{-\infty}^{\infty}\beta^2 p_-(\beta)d\beta = 2\sqrt{\pi}\Delta_0 = \frac{\sqrt{\pi}}{2}. \tag{2.26}$$

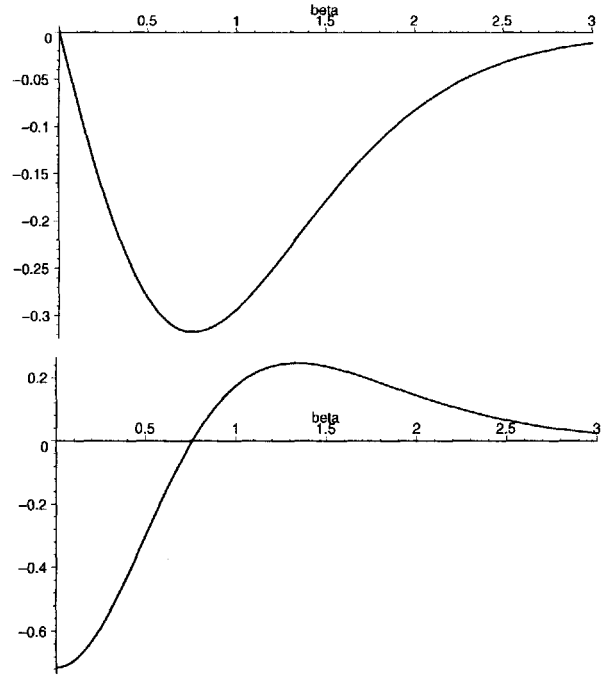


Figure 2: The first derivative $p'_-(\beta)$ and the second derivative $p''_-(\beta)$.

We thus observe that the even moments of the difference converge as follows

$$\frac{\mathbf{E}[\mathcal{D}_n^{2m+2}]}{n^{5(m+1)/2}} \to (2m+2)!\sqrt{\pi}\Delta_m.$$

These results agree with Janson [11].

Furthermore, setting

$$\bar{H}(b) = b^{6/5}\Delta(b^{4/5}) = B^{3/2}\Delta(B), \qquad b = B^{5/4} \tag{2.27}$$

we shall show that for b, $B > 0$ the function $\Delta(B)$ satisfies the nonlinear integral equation

$$0 = \int_0^B \Delta(\xi)\Delta(B-\xi)d\xi + 2B^2\Delta(B) + 2\frac{\sqrt{B}}{\sqrt{\pi}} - \frac{4}{\sqrt{\pi}}\int_0^B \frac{\Delta'(\xi)}{\sqrt{B-\xi}}d\xi. \tag{2.28}$$

We also have $\Delta(B) \sim B/4$ as $B \to 0^+$ and, in view of (2.22),

$$\Delta(B) = \sum_{m=0}^{\infty} B^{1+\frac{5}{2}m}\Delta_m. \tag{2.29}$$

The following asymptotic properties hold:

$$\Delta_m = k'\frac{e^{m/2}}{\sqrt{m}}m^{-m/2}10^{-m/2}\left[1 - \frac{4}{15m} + O(m^{-2})\right], \quad m \to$$

$$\Delta(B) = c_0 B e^{B^5/20}[1 - B^{-5} + O(B^{-10})], \quad B \to \infty,$$

$$\bar{H}(b) = c_0 b^2 e^{b^4/20}[1 - b^{-4} + O(b^{-8})], \quad b \to \infty.$$

Here k' and c_0 are related by

$$c_0 = 2\sqrt{\pi}k', \tag{2.30}$$

so that the numerical value of k' can be obtained from (2.19).

We can also infer the behavior of $\bar{H}(b)$ for purely imaginary values of b. Letting $b = ix$ with $x > 0$ and using (2.22) yields

$$1 + \sqrt{\pi}\bar{H}(ix) \equiv S(x) = \sum_{m=0}^{\infty} \Delta_{m-1}(-1)^m\sqrt{\pi}x^{2m} \tag{2.31}$$

where

$$\Delta_{-1} \equiv \frac{1}{\sqrt{\pi}}. \tag{2.32}$$

Then if

$$\bar{H}(ix) = -y^{3/2}\Lambda(y) = -x^{6/5}\Lambda(x^{4/5}), \qquad y = x^{4/5} \tag{2.33}$$

we find that $\Lambda(y)$ satisfies

$$0 = \int_0^y \Lambda(\xi)\Lambda(y-\xi)d\xi + 2y^2\Lambda(y) - 2\frac{\sqrt{y}}{\sqrt{\pi}} + \frac{4}{\sqrt{\pi}}\int_0^y \frac{\Lambda'(\xi)}{\sqrt{y-\xi}}d\xi. \tag{2.34}$$

Note that this equation differs from (2.28) only slightly, by the signs of the last two terms. However, (2.34) can be analyzed by a Laplace transform whereas (2.28) cannot. Indeed, setting

$$U(\phi) = \int_0^{\infty} e^{-y\phi}\Lambda(y)dy \tag{2.35}$$

we obtain from (2.34)

$$0 = 2U''(\phi) + U^2(\phi) + 4\sqrt{\phi}U(\phi) - \phi^{-3/2}. \tag{2.36}$$

Also, since we know the behavior of $\Lambda(y)$ as $y \to 0^+$ we must have

$$U(\phi) \sim \frac{1}{4\phi^2}, \quad \phi \to +\infty. \tag{2.37}$$

Setting

$$U(\phi) = -2\sqrt{\phi} + U_1(\phi) \tag{2.38}$$

we obtain from (2.36) and (2.37)

$$0 = U_1^2(\phi) + 2U_1''(\phi) - 4\phi, \tag{2.39}$$

with

$$U_1(\phi) = 2\sqrt{\phi} + \frac{1}{4}\phi^{-2}[1 + o(1)], \qquad \text{for} \quad \phi \to \infty.$$

The second order nonlinear ODE in (2.39) is (after a slight rescaling) the *first Painlevè transcendent* [9]. This classic problem has been studied for over 100 years, and modern applications in nonlinear waves and random matrices have been found in recent years. It is well known that each singularity of $U_1(\phi)$ is a double pole, and the Laurent expansion near any singularity at $\phi = -\nu$ has the form

$$U_1(\phi) = \frac{-12}{(\phi + \nu)^2} + O(1), \ \phi \to -\nu. \tag{2.40}$$

Let us denote by ν_* the singularity with the largest real part. Note that to uniquely fix this we need the second term in the expansion of $U_1(\phi)$ as $\phi \to \infty$, as given below (2.39). In view of (2.40), (2.35), and (2.38) we then obtain

$$\Lambda(y) - \frac{1}{\sqrt{\pi}} y^{-3/2} \sim -12 y e^{-\nu_* y}, \ y \to \infty \tag{2.41}$$

and (2.33) then yields

$$\sqrt{\pi}\bar{H}(ix) + 1 = S(x) \sim 12\sqrt{\pi}x^2 \exp(-\nu_* x^{4/5}), \ x \to \infty. \tag{2.42}$$

This yields the behavior of the moment generating function of the density $p_-(\beta)$ along the imaginary axis. The constant ν_* is found numerically as

$$\nu_* = 3.41167\ldots\,. \tag{2.43}$$

Finally, we discuss the joint distribution for the total left and right path lengths. This problem we formulate, but do not analyze. Introducing the scaling

$$p + q = \alpha n^{3/2}, \ p - q = \beta n^{5/4} \tag{2.44}$$

and define $P(p, q; n) =$ Prob [right path $= p$, left path $= q \mid \#$ nodes $= n$]. Then we obtain

$$\begin{aligned} P(p,q;n) &= = N(p,q;n)/C_n \\ &\sim 2n^{-11/4}\frac{1}{(2\pi i)^2}\int_{-i\infty}^{i\infty}\int_{-i\infty}^{i\infty} e^{-a\alpha}e^{-b\beta}[1 + \sqrt{\pi}H(a,b)]dadb \\ &\equiv 2n^{-11/4}p(\alpha,\beta). \end{aligned}$$

Thus

$$1 + \sqrt{\pi}H(a,b) = \int_0^\infty e^{a\alpha}\int_{-\infty}^\infty e^{b\beta}p(\alpha,\beta)d\beta d\alpha \tag{2.45}$$

is the moment generating function of the two-dimensional density, which has support over the range $\alpha \geq 0$ and $\beta \in \mathbb{R}$. We have $H(0,0) = 0$ and $p(\alpha, -\beta) = p(\alpha, \beta)$.

Setting $\alpha = 0$ with $\bar{H}(b) = H(0, b)$ we obtain the marginal distribution of the path length difference, with

$$p_-(\beta) = \int_0^\infty p(\alpha,\beta)d\alpha.$$

The marginal distribution of the total path length (without distinguishing between right and left paths) is given by

$$p_+(\alpha) = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty}[1 + \sqrt{\pi}H(a,0)]e^{-a\alpha}da$$

so that

$$1 + \sqrt{\pi}H(a,0) = \int_0^\infty e^{a\alpha}p_+(\alpha)d\alpha.$$

This has been previously shown to follow an Airy distribution [15].

The function $H(a, b)$ satisfies the integral equation

$$\begin{aligned} 0 &= \int_0^1 \frac{H(x^{3/2}a, x^{5/4}b)H((1-x)^{3/2}a, (1-x)^{5/4}b)}{[x(1-x)]^{3/2}}dx \\ &+ \frac{2}{\sqrt{\pi}}\int_0^1 \left\{\frac{H((1-x)^{3/2}a,(1-x)^{5/4}b)}{(1-x)^{3/2}} - H(a,b)\right\}\frac{dx}{x^{3/2}} \\ &- \frac{4}{\sqrt{\pi}}H(a,b) + (4a + 2b^2)\left[\frac{1}{\sqrt{\pi}} + H(a,b)\right]. \end{aligned} \tag{2.46}$$

In terms of the generating function in (2.8) the scaling (2.44) translates to

$$w = 1 + \frac{b}{n^{5/4}} + \frac{a}{n^{3/2}}, \ v = 1 - \frac{b}{n^{5/4}} + \frac{a}{n^{3/2}} \tag{2.47}$$

and then, for fixed a and b and $n \to \infty$,

$$G_n(w,v) \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}[1 + \sqrt{\pi}H(a,b)]. \tag{2.48}$$

Here we used the asymptotic behavior of the Catalan numbers C_n. We have thus identified the scaling (2.44) and the problem (2.46) that must be analyzed to obtain the joint distribution of left and right paths in binary trees with large numbers of nodes n. While it does not seem feasible to solve (2.46) exactly, we believe that an asymptotic analysis for a and/or b large should be possible, and from this one can obtain asymptotic properties of the joint density $p(\alpha, \beta)$ for α and/or $|\beta|$ large.

References

[1] D. Aldous, Tree-based Models for Random Distribution of Mass, *Journal of Statistical Physics*, 73, 625–641, 1993.

[2] M. Abramowitz, and I. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1964.

[3] G. Andrews, R. Askey and R. Roy, *Special Functions*, Cambridge University Press, 1999.

[4] C. Bender and S. Orszag, *Advanced Mathematical Methods for Scientists and Engineers*, Mc-Graw Hill, 1978.

[5] A. Dembo and O. Zeitouni, Large Deviations for Random Distribution of Mass, *Random Discrete Structures (Minneapolis, MN, 1993)*, 45–53, IMA Vol. Math. Appl., 76, Springer, New York, 1996.

[6] P. Flajolet, and A. Odlyzko, The Average Height of Binary Trees and Other Simple Trees, *J. Computer and System Sciences*, 25, 171–213, 1982.

[7] P. Flajolet, P. Poblete, and A. Viola, On the Analysis of Linear Probing Hashing, *Algorithmica*, 22, 490–515, 1998.

[8] P. Flajolet and R. Sedgewick, *Analytical Combinatorics*, in preparation.

[9] E. L. Ince, *Ordinary Differential Equations*, Dover, New York, 1956.

[10] S. Janson, The Wiener Index of Simply Generated Random Trees, *Random Structures and Algorithms*, 22, 337–358, 2003.

[11] S. Janson, Left and Right Pathlengths in Random Binary Trees, preprint, 2004.

[12] S. Janson and P. Chassaing, The Center of Mass of the ISE and the Wiener Index of Trees, *Electronic Comm. Probab.*, 9, 2004.

[13] J. Kim and B. Pittel, Confirming the Kleitman–Watson Conjecture on the Largest Coefficient in a q-Catalan Number, *J. Comb. Theory*, Ser. A, 92, 197-206, 2000.

[14] C. Knessl and W. Szpankowski, Quicksort algorithm again revisited, *Discrete Math. Theor. Comput. Sci.*, 3 43–64, 1999.

[15] C. Knessl and W. Szpankowski, Enumeration of Binary Trees, Lempel-Ziv'78 Parsings, and Universal Types, *Proc. of the Second Workshop on Analytic Algorithmics and Combinatorics*, (ANALCO04), Vancouver, 2005.

[16] C. Knessl and W. Szpankowski, On the Joint Path Length Distribution in Random Binary Trees, preprint, 2005.

[17] D. E. Knuth, *Selected Papers on the Analysis of Algorithms*, Cambridge University Press, Cambridge, 2000.

[18] D. E. Knuth, *The Art of Computer Programming. Fundamental Algorithms. Combinatorial Algorithms*, Vol. 4, 2004; see `http://www-cs-faculty.stanford.edu/knuth`.

[19] G. Louchard, The Brownian Excursion Area: A Numerical Analysis, *Comp. & Maths. with Appls.*, 10, 413-417, 1984.

[20] J-F. Marckert, The Rotation Correspondence is Asymptotically a Dilatation, *Random Structures & Algorithms*, 24, 118-132, 2004.

[21] L. Serlet, A Large Deviation Principle for the Brownian Snake, *Stochastic Processes and Applications*, 67, 101-115, 1997.

[22] G. Seroussi, On Universal Types, *Proc. ISIT 2004*, pp. 223, Chicago, 2004.

[23] G. Seroussi, "On the Number of t-ary Trees with a Given Pathlength", preprint.

[24] G. Slade, Scaling Limits and Super-Brownian Motion, *Notices of the AMS*, 49, 1056-1067, 2002.

[25] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York, 2001.

[26] L. Takács, A Bernoulli Excursion and its Various Applications, *J. Appl. Probab.*, 23, 557-585, 1991.

[27] L. Takács, Conditional Limit Theorems for Branching Processes, *J. Applied Mathematics and Stochastic Analysis*, 4, 263-292, 1991.

[28] L. Takács, The Asymptotic Distribution of the Total Heights of Random Rooted Trees, *Acta Sci. Math. (Szegad)*, 57, 613-625, 1993.

[29] K. Watson and D. Kleitman, On the Asymptotic Number of Tournament Score Sequences, *J. Comb. Theory*, Ser. A, 35, 208-230, 1983.

On the Variance of Quickselect*

Jean Daligault† Conrado Martínez‡

December 20, 2005

Abstract

Quickselect with median-of-three is routinely used as the method of choice for selection of the mth element out of n in general-purpose libraries such as the C++ *Standard Template Library*. Its average behavior is fairly well understood and has been shown to outperform that of the standard variant, which chooses a random pivot on each stage. However, no results were previously known about the variance of the median-of-three variant, other than for the number of comparisons made when the rank m of the sought element is given by a uniform random variable. Here, we consider the variance of the number of comparisons made by quickselect with median-of-three and other quickselect variants when selecting the mth element for $m/n \to \alpha$ as $n \to \infty$. We also investigate the behavior of proportion-from-s sampling as $s \to \infty$.

1 Introduction

Hoare's quickselect [5] finds the mth smallest element (equivalently, the element of *rank* m in ascending order, the mth order statistic) out of an array of n elements by picking an element from the array —the pivot— and rearranging the array so that elements smaller than the pivot are to its left and elements larger than the pivot are to its right. If the pivot has been brought to position $j = m$ then it is the sought element; otherwise, if $m < j$ then the procedure is recursively applied to the subarray to the left of the pivot, and if $m > j$ the process continues in the right subarray, now looking for the $(m - j)$th element.

The main measure of quickselect's performance is the number $C^{(0)}_{n,m}$ of comparisons made to select the mth smallest element out of n. Knuth [8] has shown that

$$\mathbb{E}\left[C^{(0)}_{n,m}\right] = 2\,(n + 3 + (n+1)H_n - (m+2)H_m - (n+3-m)H_{n+1-m})\,,$$

where $H_n = \sum_{1 \le j \le n} 1/j$ is the nth harmonic number. Thus $\mathbb{E}\left[C^{(0)}_{n,m}\right]$ is $\Theta(n)$ for all values of m, $1 \le m \le n$. Another interesting measure of performance is the number $C^{(0)}_n$ of comparisons needed when the rank of the sought element is given by a uniformly distributed random variable in $\{1, \ldots, n\}$. Since $\mathbb{E}\left[C^{(0)}_n\right] = (1/n) \cdot \sum_{1 \le m \le n} \mathbb{E}\left[C^{(0)}_{n,m}\right]$ it immediately follows that $\mathbb{E}\left[C^{(0)}_n\right] = 3n + o(n)$.

Sometimes it is more convenient (and more amenable to analysis) to consider the asymptotic behavior of $C^{(0)}_{n,m}$ as $n \to \infty$ and $m/n \to \alpha$, for some fixed α, $0 \le \alpha \le 1$. It is not difficult to show that $m_0(\alpha) = \lim_{\substack{n\to\infty \\ m/n\to\alpha}} \mathbb{E}\left[C^{(0)}_{n,m}\right]/n = 2 + 2\mathcal{H}(\alpha)$, where $\mathcal{H}(x) = -(x \log x + (1-x)\log(1-x))$ is the *entropy* function.

Kirschenhofer and Prodinger [6] have computed the exact form of $\mathbb{V}\left[C^{(0)}_{n,m}\right]$. It is $\Theta(n^2)$, but even its asymptotic behavior for $m = \alpha \cdot n + o(n)$ is expressed by a rather complicated formula:

(1.1)

$$\begin{aligned} v_0(\alpha) &= \lim_{\substack{n\to\infty \\ m/n\to\alpha}} \mathbb{V}\left[C^{(0)}_{n,m}\right]/n^2 \\ &= -2\mathcal{H}^2(\alpha) + 4\mathcal{H}(\alpha) - 4\log(\alpha)\log(1-\alpha) \\ &\quad + (5 + \frac{2\pi^2}{3})\alpha(1-\alpha) \\ &\quad + \frac{1}{2} - 4\alpha\,\mathrm{dilog}\,\alpha - 4(1-\alpha)\,\mathrm{dilog}(1-\alpha), \end{aligned}$$

where $\mathrm{dilog}\,x = \int_1^x \frac{\log z}{1-z}\,dz$ denotes the dilogarithm [1].

In *quickselect with median-of-three* the pivot of each recursive stage is the median of a sample of three elements of the array. This reduces the probability of uneven partitions and there is a correspond-

*The research of the authors was supported by the Spanish Min. of Science and Technology project TIC2002-00190 (AEDRI II).

†ENS Cachan. 94235 Cachan Cedex, France. jean_daligault at yahoo dot fr.

‡Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. E-08034 Barcelona, Spain. conrado at lsi dot upc dot edu.

ing reduction in the average performance (see [4, 7] and references therein). In particular, $m_1(\alpha) = \lim_{\substack{n\to\infty \\ m/n\to\alpha}} \mathbb{E}\left[C^{(1)}_{n,m}\right]/n = 2 + 3\alpha(1-\alpha)$, and the average number of comparisons to locate an element of random rank is $\mathbb{E}\left[C^{(1)}_n\right] = 5/2\, n + o(n)$. The generalization to *quickselect with median-of-*$(2t+1)$ has also been considered, both for fixed t and for variable-sized samples, i.e., when $t = t(n)$. Grübel [4] has investigated the properties of $m_t(\alpha) = \lim_{\substack{n\to\infty \\ m/n\to\alpha}} \mathbb{E}\left[C^{(t)}_{n,m}\right]/n$. Martínez and Roura [10] have computed the expected value and variance of the number of comparisons needed to locate an element of random rank $C^{(t)}_n$, for all fixed t. They also establish results for variable-size samples ($t = t(n)$), namely, the optimal sample size.

Martínez, Panario and Viola [9] have considered another family of sampling strategies that they call *proportion-from-s*. At each recursive stage, the chosen pivot has a relative rank within the sample as close as possible to the current relative rank $\alpha = m/n$ of sought element. As the algorithm proceeds, the size n of current subarray and the rank m of the sought element within the current subarray change and so does the relative rank; thus the rank of the selected pivot nicely "adapts" to the current input. Their results were established in a quite general framework, which encompasses the proportion-from-s sampling strategies, the standard variant and the median-of-$(2t+1)$ sampling strategies as particular instances of so-called *adaptive sampling strategies*[1].

The goal of this paper is to investigate and obtain explicit results about

$$v(\alpha) = \lim_{\substack{n\to\infty \\ m/n\to\alpha}} \frac{\mathbb{V}[C_{n,m}]}{n^2},$$

for several distinct variants of quickselect, each using its own sampling strategy.

First, in Section 2 we establish that, for any adaptive sampling strategy, $g(\alpha) = \lim_{\substack{n\to\infty \\ m/n\to\alpha}} \mathbb{E}\left[C^2_{n,m}\right]/n^2$ exists, with $x^{\underline{k}} = x(x-1)(x-2)\cdots(x-k+1)$ denoting the kth falling factorial of x [3], and we also give there the integral equation that $g(\alpha)$ satisfies. We get then a few general results about $g(\alpha)$. The techniques used are those developed in [4, 9]. In Section 3 we obtain explicit solutions to that integral equation for the particular case of median-of-three and thus for its variance. Afterwards, in Section 4, we show that if $s \to \infty$ then proportion-from-s sampling strategies achieve not only optimal expected performance (a result due to [9]) but subquadratic variance, i.e., $\lim_{\substack{n\to\infty \\ m/n\to\alpha}} \mathbb{V}[C_{n,m}]/n^2 = 0$. The immediate consequence is that $C_{n,m}$ exhibits concentration in probability. Median-of-$(2t+1)$ also achieves subquadratic variance in the limit $t \to \infty$, even though the expected performance is not optimal in that case.

[1] Even though some of them do not "adapt" their choice of pivots, like standard quickselect; but they are special degenerate cases of the general definition given there.

2 General results

Following [9], we say that a sampling strategy is *adaptive* if it can be fully described by a function $r : [0,1] \to \{1,\ldots,s\}$, where s is the size of the samples. Quickselect with adaptive sampling works as follows: if $n \geq s$, a random sample of s elements from the current subarray of size n is chosen and the element whose rank within the sample is $r = r(\alpha)$ is picked as the pivot of the current recursive stage, where $\alpha = m/n$ is the current relative rank of the sought element. We assume further that the function r can be finitely specified by the image of each interval of a partition of $[0,1]$ into ℓ intervals. For convenience, we will assume that the intervals I_k are defined by $\ell - 1$ endpoints $0 = a_0 < a_1 < a_2 < \cdots < a_{\ell-1} < a_\ell = 1$ as follows: $I_1 = [0, a_1]$, $I_\ell = [a_{\ell-1}, 1]$, $I_k = (a_{k-1}, a_k]$ if $k > 1$ and $a_k \leq 1/2$, $I_k = [a_{k-1}, a_k)$ if $k < \ell$ and $a_{k-1} > 1/2$, and $I_k = (a_{k-1}, a_k)$ if $a_{k-1} \leq 1/2 < a_k$ and $1 < k < \ell$. We will use the notation r_k for the value of $r(\alpha)$ when $\alpha \in I_k$.

When $s = 1$ we have standard quickselect, and $r(\alpha) = r_1 = 1$ for all $\alpha \in [0,1]$. Median-of-$(2t+1)$ sampling is characterized by $s = 2t+1$ and $r(\alpha) = r_1 = t+1$ for all $\alpha \in [0,1]$. In [9], proportion-from-s sampling is introduced; for these strategies $\ell = s$ and $r(\alpha) = r_k = k$ for all $\alpha \in I_k$ and $1 \leq k \leq s$. The choice of endpoints gives raise to interesting mathematical phenomena with relevant practical implications; for "pure" proportion-from-s, we have $a_k = k/s$.

We now restate two of the fundamental results of [9] concerning quickselect with adaptive sampling.

THEOREM 2.1. ([9]) *Let $C_{n,m}$ be the cost to select the mth out of n elements using an adaptive sampling strategy with $m/n \to \alpha$ for $0 \leq \alpha \leq 1$ as $n \to \infty$. Then we have that the* expectation characteristic function *of the algorithm*

$$f(\alpha) = \lim_{n\to\infty} \frac{\mathbb{E}[C_{n,m}]}{n},$$

is well defined, and

$$f(\alpha) = 1 + \frac{s!}{(r(\alpha)-1)!(s-r(\alpha))!} \times \left[\int_\alpha^1 f(\alpha/x) x^{r(\alpha)} (1-x)^{s-r(\alpha)}\, dx + \int_0^\alpha f\left(\frac{\alpha - x}{1-x}\right) x^{r(\alpha)-1} (1-x)^{s+1-r(\alpha)}\, dx \right].$$

It is important to notice that because $r(\alpha)$ is an integer function the discontinuities carry on to $f(\alpha)$. So in general $f(\alpha)$ is defined by ℓ pieces, say, $f_1, \ldots, f_\ell$, with f_k the restriction of $f(\alpha)$ for α in the kth interval. The integral equation above can be transformed, after careful manipulations, to a set of higher-order linear differential equations, as shown in the following lemma.

LEMMA 2.1. ([9]) *For any adaptive sampling strategy,*

$$\frac{d^{s+2}}{d\alpha^{s+2}} f_k(\alpha) = \frac{(-1)^{s+1-r_k}}{\alpha^{s+1-r_k}} \cdot \frac{s!}{(r_k-1)!} \cdot \frac{d^{r_k+1}}{d\alpha^{r_k+1}} f_k(\alpha) + \frac{1}{(1-\alpha)^{r_k}} \cdot \frac{s!}{(s-r_k)!} \cdot \frac{d^{s+2-r_k}}{d\alpha^{s+2-r_k}} f_k(\alpha),$$

where $f(\alpha)$ is the strategy's expectation characteristic function, and $\alpha \in I_k$, $1 \le k \le \ell$.

In order to obtain similar results about the variance of $C_{n,m}$, we consider its second factorial moment $\mathbb{E}\left[C_{n,m}^2\right] = \mathbb{E}[C_{n,m}(C_{n,m}-1)]$ since $\mathbb{V}[C_{n,m}] = \mathbb{E}\left[C_{n,m}^2\right] + \mathbb{E}[C_{n,m}] - \mathbb{E}[C_{n,m}]^2$. The starting point of our analysis is the recurrence satisfied by $C_{n,m}(v)$, the probability generating function (PGF) of $C_{n,m}$

$$C_{n,m}(v) = \sum_{k\ge 0} \Pr\{C_{n,m} = k\} v^k = v^{n-1} \left[\sum_{j=1}^{m-1} \pi_{n,j}^{(s,r)} C_{n-j,m-j}(v) + \pi_{n,m}^{(s,r)} + \sum_{j=m+1}^{n} \pi_{n,j}^{(s,r)} C_{j-1,m}(v) \right], \tag{2.2}$$

where $\pi_{n,j}^{(s,r)}$ denotes the probability that the rth element of the sample of size s is the jth element among the n elements. The recurrence accounts for the $n-1$ comparisons needed to partition the array around the pivot, but it disregards the comparisons needed to select the pivot from the sample; this is unimportant since we assume that s is fixed. Now, $\mathbb{E}[C_{n,m}] = C'_{n,m}(1)$ and $\mathbb{E}\left[C_{n,m}^2\right] = C''_{n,m}(1)$; hence,

$$\mathbb{E}\left[C_{n,m}^2\right] = 2(n-1)\,\mathbb{E}[C_{n,m}] - n(n-1) + \sum_{j=1}^{m-1} \pi_{n,j}^{(s,r)}\, \mathbb{E}\left[C_{n-j,m-j}^2\right] + \sum_{j=m+1}^{n} \pi_{n,j}^{(s,r)}\, \mathbb{E}\left[C_{j-1,m}^2\right].$$

From this recurrence, we can establish the following result.

THEOREM 2.2. *Let $C_{n,m}$ be cost to select the mth out of n elements using an adaptive sampling strategy with $m/n \to \alpha$ for $0 \le \alpha \le 1$ as $n \to \infty$. Then we have that the* second factorial moment characteristic function *of the algorithm*

$$g(\alpha) = \lim_{n\to\infty} \frac{\mathbb{E}\left[C_{n,m}^2\right]}{n^2},$$

is well defined, and

$$g(\alpha) = 2f(\alpha) - 1 + \frac{s!}{(r(\alpha)-1)!(s-r(\alpha))!} \times \left[\int_\alpha^1 g(\alpha/x) x^{r(\alpha)+1} (1-x)^{s-r(\alpha)}\, dx + \int_0^\alpha g\left(\frac{\alpha - x}{1-x}\right) x^{r(\alpha)-1} (1-x)^{s+2-r(\alpha)}\, dx \right],$$

where $f(\alpha)$ is the expectation characteristic function of the sampling strategy.

Once we have results about $g(\alpha)$, they can be easily translated to the variance since $v(\alpha) = \lim_{n\to\infty} \mathbb{V}[C_{n,m}]/n^2 = g(\alpha) - f^2(\alpha)$. For instance, a sampling strategy is said to be *symmetric* if $\lim_{z\to\alpha^+} r(z) = \lim_{z\to\alpha^+} s + 1 - r(1-z)$. This notion is quite natural, and both median-of-$(2t+1)$ and proportion-from-s strategies[2] are symmetric. If r is symmetric then both $g(\alpha)$ and $v(\alpha)$ are as well, now in the usual sense, i.e., $g(\alpha) = g(1-\alpha)$ and $v(\alpha) = v(1-\alpha)$. Another interesting result concerns the behavior of $v(\alpha)$ when $\alpha \to 0$.

[2] Provided that the endpoints are taken such that $a_k = a_{s-k}$ for $k > s/2$.

LEMMA 2.2. *For any adaptive sampling strategy*

$$\lim_{\alpha\to 0} v(\alpha) = \frac{r_0(s+1)}{(s+1-r_0)((s+2)(s+1)-r_0(r_0+1))},$$

where $r_0 = \lim_{\alpha\to 0} r(\alpha)$ *and all limits of* $\alpha \to 0$ *are taken from the right.*

In particular, for standard quickselect ($t = 0$) and quickselect with median-of-$(2t+1)$, we have $v_t(0) = v_t(1) = \frac{2}{3t+4}$, since $r_0 = t+1$ and $s = 2t+1$. For proportion-from-s, $v(0) = v(1) = \frac{s+1}{s^2(s+3)} \sim s^{-2} + O(s^{-3})$. So proportion-from-$s$ has smaller variance when locating elements of either low or high rank than median-of-$(2t+1)$. Furthermore, this result indicates that using large samples can reduce the order of magnitude of the variance; notice that for both types of strategies the coefficient of n^2 in the variance tends to 0 when the size of the samples tends to ∞ and we look for extreme ranks. We will show later that this is indeed true for all ranks.

We now state one of the important results of this paper, where we transform the original problem to one of solving linear differential equations; we arrive at this result after long and careful computations largely similar to those which yield Lemma 2.1.

LEMMA 2.3. *For any adaptive sampling strategy,*

$$\begin{aligned}\frac{d^{s+3}}{d\alpha^{s+3}} g_k(\alpha) &= 2\frac{d^{s+3}}{d\alpha^{s+3}} f_k(\alpha) \\ &+ \frac{(-1)^{s+1-r_k}}{\alpha^{s+1-r_k}} \cdot \frac{s!}{(r_k-1)!} \cdot \frac{d^{r_k+2}}{d\alpha^{r_k+2}} g_k(\alpha) \\ &+ \frac{1}{(1-\alpha)^{r_k}} \cdot \frac{s!}{(s-r_k)!} \cdot \frac{d^{s+3-r_k}}{d\alpha^{s+3-r_k}} g_k(\alpha),\end{aligned}$$

where $g(\alpha)$ *is the second factorial moment characteristic function,* g_k *is its restriction to the* k*th interval,* $f(\alpha)$ *is the expectation characteristic function, and* $\alpha \in I_k$, $1 \le k \le \ell$.

Thus, except for the independent term $2f^{(s+3)}(\alpha)$ and the higher order derivatives involved, we have the same differential equation as for the expectation characteristic function.

3 The variance of median-of-three

In the case of median-of-three ($s = 3$, $\ell = 1$ and $r_1 = 2$) we are specially lucky, since its expectation characteristic function is $m_1(\alpha) = 2 + 3\alpha(1-\alpha)$ and its sixth derivative vanishes in the differential equation satisfied by $g(\alpha)$ (Lemma 2.3). Hence the corresponding differential equation for $g(\alpha)$ is exactly the same as for the expectation characteristic function $m_1(\alpha)$, namely

$$\frac{d^2\phi}{d\alpha^2} - 6\left(\frac{1}{\alpha^2} + \frac{1}{(1-\alpha)^2}\right)\phi(\alpha) = 0,$$

but here $\phi(\alpha) = g^{(iv)}(\alpha)$, instead of $\phi(\alpha) = m_1'''(\alpha)$.

Hence, the solution has to be integrated four times to recover $g(\alpha)$. The symmetry $g^{(4)}(\alpha) = g^{(4)}(1-\alpha)$ can be used to show that

$$\begin{aligned}\phi(\alpha) &= g^{(4)}(\alpha) \\ &= C_1 \frac{-1+2\alpha-28\alpha^5+56\alpha^6-40\alpha^7+10\alpha^8}{2\alpha^2(1-\alpha)^2}.\end{aligned}$$

Integrating this four times we get four additional arbitrary constants. The technique that we shall use to obtain their value is to plug the general form of $g(\alpha)$ back into the integral equation and compare coefficients. This involves a somewhat long and tedious computation, but it is mostly mechanical and the use of a computer algebra systems is of great help. We finally get

$$\begin{aligned}g(\alpha) = &-\frac{288}{35}\alpha^2(\log(\alpha)+\log(1-\alpha)) - \frac{288}{35}\log(1-\alpha) \\ &+ \frac{576}{35}\alpha\log(1-\alpha) + \frac{30}{7} - \frac{24}{245}\alpha^8 \\ &+ \frac{96}{245}\alpha^7 - \frac{48}{175}\alpha^6 - \frac{96}{175}\alpha^5 \\ &- \frac{48}{35}\alpha^4 + \frac{144}{35}\alpha^3 - \frac{7332}{1225}\alpha^2 + \frac{132}{35}\alpha,\end{aligned}$$

and from there

(3.3)

$$\begin{aligned}v_1(\alpha) &= \lim_{n\to\infty, m/n\to\alpha} \mathbb{V}\left[C^{(1)}_{n,m}\right]/n^2 = g(\alpha) - m_1^2(\alpha) \\ &= -\frac{288}{35}\log(1-\alpha) + \frac{576}{35}\alpha\log(1-\alpha) + \frac{2}{7} \\ &\quad - \frac{288}{35}\alpha - \frac{288}{35}\alpha^2(\log\alpha + \log(1-\alpha)) \\ &\quad - \frac{24}{245}\alpha^8 + \frac{96}{245}\alpha^7 - \frac{48}{175}\alpha^6 - \frac{96}{175}\alpha^5 \\ &\quad - \frac{363}{35}\alpha^4 + \frac{774}{35}\alpha^3 - \frac{3657}{1225}\alpha^2,\end{aligned}$$

since $m_1(\alpha) = 2+3\alpha(1-\alpha)$. The function $v_1(\alpha)$ is symmetric and has two global maxima at $\alpha \doteq 0.263338\ldots$ and $\alpha \doteq 0.736661\ldots$, where it attains the value $v_1 \doteq 0.391151\ldots$. The global minima are $\alpha = 0$ and $\alpha = 1$, where $v_1 = 2/7 \doteq 0.285714\ldots$, while $v_0(0) = v_0(1) = 1/2$. The function is depicted in Figure 1. A plot of the function $v_0(\alpha)$ corresponding to standard quickselect is shown in Figure 2 for comparison.

The leading coefficient of the variance to locate the median of an array is given by $v_1(1/2) = \frac{144}{35}\log 2 - \frac{97121}{39200} \doteq 0.374229\ldots$. Compare this to $v_0(1/2) = -4\log^2 2 + 4\log 2 + \frac{7}{4} - \frac{\pi^2}{6} \doteq 0.955842\ldots$, where $v_0(\alpha)$ has its unique global maximum.

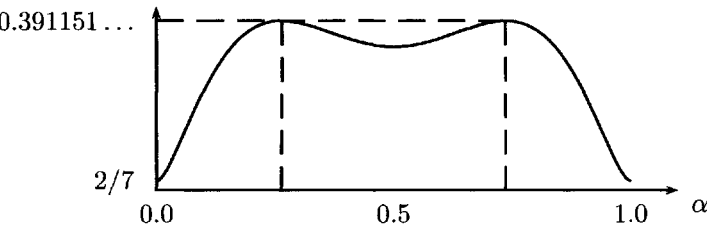


Figure 1: Plot of the coefficient $v_1(\alpha)$ of the variance of quickselect with median-of-three, when looking for the $m = \alpha \cdot n$ smallest element and $n \to \infty$.

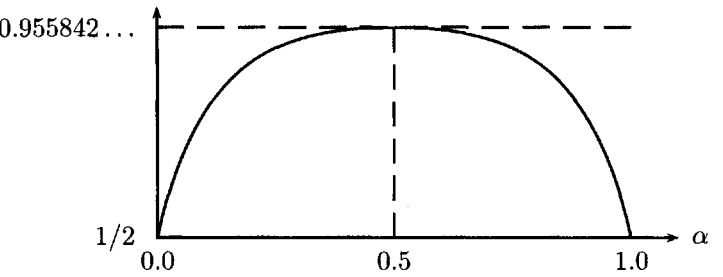


Figure 2: Plot of the coefficient $v_0(\alpha)$ of the variance of standard quickselect, when looking for the $m = \alpha \cdot n$ smallest element and $n \to \infty$ (after [6]).

4 Optimal sampling

Consider a family of *biased* symmetric proportion-from-s sampling strategies, $s \in \mathbb{N}$, with $r(\alpha)/s \to \alpha$ as $s \to \infty$. A sampling strategy is called *biased* [9] if $r(\alpha) > \alpha \cdot s + 1 - \alpha$ for $\alpha < 1/2$. The quantity $\delta = r - s\alpha - 1 + \alpha > 0$ is called the bias. For a biased proportion-from-s strategy the endpoints of the intervals are not evenly distributed in $[0,1]$ but shifted towards the left for $\alpha < 1/2$ and, symmetrically, towards the right when $\alpha > 1/2$. In [9], it has been shown that optimal average performance is achieved when $s \to \infty$ for such a family. In particular, the limiting expectation characteristic function is $f_\infty(\alpha) = \lim_{s\to\infty} f_s(\alpha) = 1 + \min(\alpha, 1-\alpha)$.

To tackle a similar analysis for the variance we have to study the behavior of

$$\begin{aligned} T^{(\infty)}(F,G)(\alpha) &= 2F(\alpha) - 1 \\ &\quad + \lim_{s\to\infty} \frac{s!}{(r(\alpha)-1)!(s-r(\alpha))!} \times \\ &\quad \left\{ \int_\alpha^1 G\left(\frac{\alpha}{x}\right) x^{r(\alpha)+1}(1-x)^{s-r(\alpha)}\,dx \right. \\ &\quad \left. + \int_0^\alpha G\left(\frac{\alpha-x}{1-x}\right) x^{r(\alpha)-1}(1-x)^{s+2-r(\alpha)}\,dx \right\}. \end{aligned}$$

Since $T^{(\infty)}(f_\infty, \cdot)$ is a contraction (see [4, 9]) it suffices to find a fixed point g_∞; if we find it then it is unique, and it is the limit characteristic function for the second factorial moment of our family of sampling strategies. In order to do that, we will need to analyze the asymptotic behavior of

$$I(s) = \frac{s!}{(r(\alpha)-1)!(s-r(\alpha))!} \times \int_a^b y(x) \cdot x^{r(\alpha)-1} \cdot (1-x)^{s-r(\alpha)}\,dx, \tag{4.4}$$

as $s \to \infty$, as the operator $T^{(\infty)}$ can be expressed as a combination of integrals with the form above, for suitable choices of the integral limits and the function $y(x)$.

We apply Laplace's method to find asymptotic estimations of the value of those integrals (see for instance [2, Ch. 5, p. 211–212]). The maximum of the "kernel" $x^{r-1}(1-x)^{s-r}$ occurs at $x^* = (r-1)/(s-1)$ and so we have two fundamental situations: either the maximum x^* is inside (a,b) and then $I(s) = y(x^*) + \mathcal{O}(s^{-1})$; otherwise, if $x^* \notin [a,b]$ then $I(s) = \mathcal{O}(s^{-1})$ (in general, $I(s) \to 0$ exponentially fast, except if the maximum c were $c = a$ or $c = b$ and $c \sim x^*$). Take $g_\infty(\alpha) = (1 + \min(\alpha, 1-\alpha))^2$. Then

$$\begin{aligned} T^{(\infty)}(f_\infty, g_\infty)(\alpha) &= 2(1+\min(\alpha,1-\alpha)) - 1 \\ &\quad + \lim_{s\to\infty} \frac{s!}{(r(\alpha)-1)!(s-r(\alpha))!} \times \Bigg[\\ &\quad \left\{ \int_\alpha^1 x^2 \left(1 + \min\left(\frac{\alpha}{x}, \frac{x-\alpha}{x}\right)\right)^2 \right. \\ &\quad \left. \cdot\, x^{r(\alpha)-1}(1-x)^{s-r(\alpha)}\,dx \right\} \\ &\quad + \left\{ \int_0^\alpha (1-x)^2 \left(1 + \min\left(\frac{\alpha-x}{1-x}, \frac{1-\alpha}{1-x}\right)\right)^2 \right. \\ &\quad \left. \cdot\, x^{r(\alpha)-1}(1-x)^{s-r(\alpha)}\,dx \right\} \Bigg]. \end{aligned}$$

Now, if $\alpha < 1/2$ then, by the definition of a biased strategy, $\alpha < x^*$ and thus the second integral tends to 0, while the first yields the main contribution with $y(x) = (2x-\alpha)^2$. Also, as $x^* = \alpha + \mathcal{O}(s^{-1})$, we have $T^{(\infty)}(g_\infty)(\alpha) \sim 2(1+\alpha) - 1 + \alpha^2 = (1+\alpha)^2$. Because of the symmetry of r, and after careful checking of the special case $\alpha = 1/2$, we finally arrive at

$$T^{(\infty)}(f_\infty, g_\infty) = g_\infty = (1 + \min(\alpha, 1-\alpha))^2.$$

That is, g_∞ is the limiting behavior of the second moment factorial characteristic function of any family of biased proportion-from-s sampling strategies; and $g_\infty = f_\infty^2$. Thus we obtain the following important result.

THEOREM 4.1. *For any family of symmetric* biased *sampling strategies such that* $\lim_{s\to\infty} r(\alpha)/s = \alpha$, *the variance of quickselect using the family of sampling strategies is subquadratic. Indeed,*

$$\lim_{s\to\infty} \lim_{n\to\infty, m/n\to\alpha} \frac{\mathbb{V}[C_{n,m}]}{n^2} = 0.$$

Despite median-of-$(2t+1)$ sampling doesn't yield optimal average behavior, an analogous to Theorem 4.1 holds. In particular, if $t \to \infty$ then the expectation characteristic function $m_t(\alpha) \to 2$ [4]. Using the same techniques that we have just used, we can also easily show that $g_t(\alpha) \to 4$. Hence, the coefficient of n^2 in $\mathbb{V}\left[C_{n,m}^{(t)}\right]$ vanishes as $t \to \infty$. The same type of result was already established for the variance of the cost of selecting an element of random rank by Martínez and Roura [10].

Now we turn our attention to the case of variable-sized samples, i.e., when $s = s(n)$. We assume that $s \to \infty$ as $n \to \infty$ but that it does grow sublinearly ($s = o(n)$). To begin with, the proof given in [9] that the expectation characteristic function of biased proportion-from-s strategies tends to $f_\infty(\alpha) = 1 + \min(\alpha, 1-\alpha)$ as $s \to \infty$ is valid for variable-size samples. The average number of comparisons $\mathbb{E}[C_{n,m}]$ satisfies the recurrence

$$\mathbb{E}[C_{n,m}] = n + \beta\cdot s + o(s) + \sum_{j=m+1}^{n} \pi_{n,j}^{(s,r)} \cdot \mathbb{E}[C_{j-1,m}] + \sum_{j=1}^{m-1} \pi_{n,j}^{(s,r)} \cdot \mathbb{E}[C_{n-j,m-j}];$$

notice the the terms $\beta\cdot s + o(s)$, which account for the average number of comparisons invested in selecting the pivots from the samples. Here, the factor $\beta = \beta(\alpha)$ is the coefficient of the linear cost of the algorithm used to select the pivot. For instance, $\beta(\alpha) = 2 + 3\alpha(1-\alpha)$ if we were using quickselect with median-of-three for selecting pivots from the samples. The full details are not straightforward, but the intuition is rather simple; since $s/n \to 0$ and the asymptotic estimate for the splitting probabilities $\pi_{n,j}^{(s,r)}$ are valid for $s = s(n)$, Theorems 2.1 and 2.2 hold too in this case.

By computing more precise asymptotic estimates of $I(s)$ (see (4.4)), we can establish the behavior of the lower order terms of f_∞ and g_∞. In particular, $f_\infty(\alpha) = 1 + \min(\alpha, 1-\alpha) + \mathcal{O}(s^{-1})$ and $g_\infty(\alpha) = (1+\min(\alpha,1-\alpha))^2 + \mathcal{O}(s^{-1})$. Then $\mathbb{E}[C_{n,m}] = n(1+\min(\alpha,1-\alpha)) + \beta\cdot s + \mathcal{O}(n/s)$ and $\mathbb{E}\left[C_{n,m}^2\right] = n^2(1+\min(\alpha,1-\alpha))^2 + \Theta\left(n\cdot s + n^2/s\right)$. Hence, we have the following result.

THEOREM 4.2. *Consider a symmetric* biased *sampling strategy with* $s = s(n) \to \infty$ *as* $n \to \infty$, *with* $s = o(n)$ *and such that* $\lim_{s\to\infty} r(\alpha)/s = \alpha$. *Then the variance of quickselect using this sampling strategy is*

$$\mathbb{V}[C_{n,m}] = \Theta\left(\max\left\{n\cdot s, n^2/s\right\}\right).$$

The variance of a sampling strategy satisfying the hypothesis of the theorem above is minimized when $s = \Theta(\sqrt{n})$. Notice that this is consistent with Theorem 4.1 and that we have then $\sqrt{\mathbb{V}[C_{n,m}]} = \Theta(n^{3/4})$. We prove thus correct several conjectures made in [9] concerning the variance of quickselect with proportion-from-s sampling for variable-sized samples. Also, as $\mathbb{E}[C_{n,m}] = n(1+\min(\alpha,1-\alpha)) + \beta\cdot s + O(n/s)$, it follows that $s = \Theta(\sqrt{n})$ minimizes the average number of comparisons made. Lacking of a better estimate for the term $O(n/s)$ we cannot obtain a more precise asymptotic estimate for the optimal sample size.

References

[1] M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions.* Dover Publ., New York, 1964.

[2] N. Bleistein and R. A. Handelsman. *Asymptotic Expansions of Integrals.* Dover Pub., New York, 1975.

[3] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, Mass., 2nd edition, 1994.

[4] R. Grübel. On the median-of-k version of Hoare's selection algorithm. *Theoretical Informatics and Applications*, 33(2):177–192, 1999.

[5] C.A.R. Hoare. FIND (Algorithm 65). *Communications of the ACM*, 4:321–322, 1961.

[6] P. Kirschenhofer and H. Prodinger. Comparisons in Hoare's Find algorithm. *Combinatorics, Probability and Computing*, 7:111–120, 1998.

[7] P. Kirschenhofer, H. Prodinger, and C. Martínez. Analysis of Hoare's FIND algorithm with median-of-three partition. *Random Structures and Algorithms*, 10(1):143–156, 1997.

[8] D.E. Knuth. Mathematical analysis of algorithms. In *Information Processing '71, Proc. of the 1971 IFIP Congress*, pages 19–27, Amsterdam, 1972. North-Holland.

[9] C. Martínez, D. Panario, and A. Viola. Adaptive sampling for quickselect. In *Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 440–448, 2004.

[10] C. Martínez and S. Roura. Optimal sampling strategies in quicksort and quickselect. *SIAM Journal on Computing*, 31(3):683–705, 2001.

Semirandom Models as Benchmarks for Coloring Algorithms *†

Michael Krivelevich ‡ Dan Vilenchik §

Abstract

Semirandom models generate problem instances by blending random and adversarial decisions, thus intermediating between the worst-case assumptions that may be overly pessimistic in many situations, and the easy pure random case. In the $G_{n,p,k}$ random graph model, the n vertices are partitioned into k color classes each of size n/k. Then, every edge connecting two different color classes is included with probability $p = p(n)$. In the semirandom variant, $G^*_{n,p,k}$, an adversary may add edges as long as the planted coloring is respected. Feige and Killian prove that unless $NP \subseteq BPP$, no polynomial time algorithm works **whp** when $np < (1-\epsilon)\ln n$, in particular when np is constant. Therefore, it seems like $G^*_{n,p,k}$ is not an interesting benchmark for polynomial time algorithms designed to work **whp** on *sparse* instances (np a constant). We suggest two new criteria, using semirandom models, to serve as benchmarks for such algorithms. We also suggest two new coloring heuristics and compare them with the coloring heuristics suggested by Alon and Kahale 1997 and by Böttcher 2005. We prove that in some explicit sense both our heuristics are preferable to the latter.

1 Introduction and Results

Introduction. A *k-coloring* f of a graph $G = (V, E)$ is a mapping from the set of vertices V to $\{1, 2, ..., k\}$. f is a *legal coloring* of G if for every edge $(u, v) \in E$, $f(u) \neq f(v)$. In the graph coloring problem we are given a graph $G = (V, E)$ and are asked to produce a legal k-coloring f with a minimal possible k. Such k is called the chromatic number, commonly denoted by $\chi(G)$. For a broad view of the coloring problem the reader is referred to [19].

The plethora of worst-case NP-hardness results for problems in graph theory motivates the study of heuristics that give "useful" answers for "typical" subset of the problem instances, where "useful" and "typical" are usually not well defined. One way of evaluating and comparing heuristics is by running them on a collection of input graphs ("benchmarks"), and checking which heuristic usually gives better results. Though empirical results are sometimes informative, we seek more rigorous measures of evaluating heuristics. A rigorous candidate for the analog of a "useful" answer is the notion of approximation, where the goal of the heuristic is to provide with a solution which is guaranteed to be within small distance from an optimal one. Although approximation algorithms are known for several NP-hard problems, the coloring problem is not amongst them. In particular, Feige and Kilian [10] prove that no polynomial time algorithm approximates $\chi(G)$ within a factor of $n^{1-\epsilon}$ for all input graphs G, unless ZPP=NP.

When very little can be done in the worst case, comparing heuristics' behavior on "average" instances comes in mind. One possibility of rigourously modeling "average" instances is to use random models. A good candidate can be the well known random graph model $G_{n,p}$ introduced by Erdös and Rényi. A random graph G in $G_{n,p}$ consists of n vertices, and each of $\binom{n}{2}$ possible edges is included w.p. $p = p(n)$ independently of the other. Bollobás [5] and Luczak [24] calculated the probable value of $\chi(G_{n,p})$ to be **whp** [1] approximately $np/(2\ln(np))$ for $p \in [C_0/n, \log^{-7} n]$ ([5] actually extends to $p \leq 0.99$ with a somewhat different expression for the chromatic number). Observe that the chromatic number is typically rather high (roughly comparable with the expected degree). In order to consider graphs with a smaller chromatic number, Kučera [23] suggested a model for generating random k-colorable graphs, denoted throughout by $G_{n,p,k}$. First, randomly partition the vertex set $V = \{1, ..., n\}$ into k classes $V_1, ..., V_k$, of size n/k each. Then, for every $i \neq j$, include every possible edge connecting a vertex in V_i with a vertex in V_j (abbreviated $V_i - V_j$ edges) with probability $p = p(n)$. This model is the analog of the planted clique, planted bisection, and planted SAT distributions, studied e.g. in [2], [11], [13], [14].

The Semirandom Model. The main drawback of random models is that they may simply not capture the space of "useful" problems. The instances generated using random models are extremely unstructured (see [16] for example), which probably does not reflect the real-world examples. Further, there is the temptation

*Supported in part by USA-Israel BSF Grant 2002-133, and by Grant 526/05 from the Israel Science Foundation.

†Part of the second author's PhD thesis prepared at Tel Aviv University under the supervision of Prof. Michael Krivelevich.

‡School Of Mathematical Sciences, Sackler Faculty of Exact Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel.

§School of Computer Science, Sackler Faculty of Exact Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel.

[1] Writing **whp** we mean with probability tending to 1 as n goes to infinity.

of over-exploiting the statistical properties of the random graph (eigenvalues structure, vertex degrees, etc) and design algorithms that perform well on a specific distribution but fail completely when slightly changing the distribution to a more realistic one (as many such graph properties no longer possess the "clean" and manageable behavior they have in the random setting).

To capture this notion of robustness desired from an algorithm, semirandom models are introduced. In the semirandom setting, first a random instance is generated. Next, an adversary may change the instance further. These modifications cannot be arbitrary, or the adversary can remake the graph into a worst-case instance. Put differently, semirandom models generate problem instances by blending random and adversarial decisions, thus intermediating between the worst-case assumptions, which may be overly pessimistic in many situations, and the easy pure random case. As such, they often serve as a driving force towards designing more natural and efficient algorithms (e.g., introducing semi-definite programming not only as an important tool in approximation algorithms but rather as part of heuristics that solve "typical", and adversarial, instances [9], [11]. [13] present a simpler and more natural algorithm for a semi-random planted 3SAT distribution, compared with [14] for the random setting).

The following semirandom variant of $G_{n,p,k}$ was suggested by Blum and Spencer [4], and is denoted throughout by $G^*_{n,p,k}$. First, a graph $G_0 = G_{n,p,k}$ is generated (throughout, we use G_0 to denote a random graph sampled according to $G_{n,p,k}$, or the underlying random part of a semirandom instance. The meaning will be clear from the context). Next, an adversary is allowed to add $V_i - V_j$ edges for $i \neq j$.

Related Work. [22] suggests an $O(\sqrt{np}/\log n)$-approximation algorithm for the chromatic number of graphs on n vertices. They prove that over $G_{n,p}$, $p \in [n^{-\frac{1}{2}+\epsilon}, \frac{3}{4}]$, the algorithm runs in expected polynomial time [2]. [8] extends the latter to $p \geq C/n$, C a sufficiently large constant. In this work we focus on heuristics that find a correct solution for "almost all" instances. Alon and Kahale [1] suggest a polynomial time algorithm based on spectral techniques that **whp** finds a k-coloring of $G_{n,p,k}$ with $np \geq C_0 k^2$, C_0 a sufficiently large constant. Combining techniques from [1] and [7], [6] suggests an expected polynomial time algorithm for $G_{n,p,k}$ based on SDP (semi-definite programming). Both [1] and [6] fail in $G^*_{n,p,k}$, as the adversary may foil many statistics of the random graph which both algorithms heavily rely on (eigenvalues structure, vertex degrees, etc). Blum and Spencer [4] present a heuristic that k-colors $G^*_{n,p,k}$ **whp** for a constant k, and $np \geq n^{\alpha_k}$, $\alpha_k \geq 2/5$. Feige and Kilian [9] improve upon this result, giving an SDP-based algorithm that k-colors $G^*_{n,p,k}$ for $np \geq c(1+\epsilon)k \ln n$. They also provide with a hardness result, proving that unless $NP \subseteq BPP$, it is hard[3] to k-color $G^*_{n,p,k}$ for $np < (1-\epsilon)\ln n$. The proof of the hardness result is based on the existence **whp** of isolated vertices when $np < (1-\epsilon)\ln n$, more details ahead. Coja-Oghlan [7] gives a simpler SDP-based heuristic that k-colors $G^*_{n,p,k}$ for $np \geq c(1+\epsilon)k \ln n$, and also provides a certificate for the optimality of the coloring. [7] improves upon the hardness result of [9], proving that unless $NP \subseteq RP$, it is hard to k-color $G^*_{n,p,k}$ for $np \leq (1-\epsilon)\frac{k}{2}\ln(n/k)$.

Our Results. In this work, we focus on the *sparse* case, namely when np is constant (that may depend on k). Semirandom distributions of sparse instances need to be addressed with more delicacy. For example, in $G^*_{n,p,k}$, np a constant, **whp** there will be a constant fraction of isolated vertices in G_0 on which the adversary (if not restricted otherwise) can plant a worst-case instance, turning the problem hard. This is the basic idea behind the hardness proofs in [7], [9]. Consequently, $G^*_{n,p,k}$ is not an interesting benchmark in our (sparse) setting . Therefore, different criteria for evaluating the robustness of algorithms in the sparse case are in due.

The first to consider semirandom models for sparse settings were [13] and [25], in the context of the planted SAT distribution. In this paper we discuss two alternatives for the coloring problem, which prove quite useful. One possibility is to further limit the adversary and to require that the algorithm succeeds **whp** when spending polynomial time. To this end we introduce the semirandom model $G^H_{n,p,k}$. The only difference between $G^*_{n,p,k}$ and $G^H_{n,p,k}$ is that in the latter, the adversary is allowed to add only edges whose both endpoints belong to a certain set $H \subseteq V$ (which will be rigorously defined and analyzed in the sequel). Another possibility is to require that the algorithm finds a solution **whp** over $G^*_{n,p,k}$. This time however it will not be realistic to require that the algorithm spends only polynomial time (the aforementioned hardness results), rather the algorithm is allowed to spend as much time as needed to guarantee a solution **whp**. The preferable heuristic in this case is the one guaranteeing a solution **whp** while spending as little time as possible.

Building upon [1] and [7] we present two new coloring heuristics, COLOR and COLOR2. Using the afore-

[2] An algorithm $\mathcal{A}$ with running time $t_{\mathcal{A}}(I)$ on an input instance I, has *expected polynomial running time* over a distribution $\mathcal{D}$ on the inputs, if $\sum_I t_{\mathcal{A}}(I) \cdot Pr_{\mathcal{D}}[I]$ is polynomial.

[3] By "hard" we mean there exists no polynomial time algorithm that solves the problem **whp**

mentioned criteria, we compare COLOR, COLOR2, [1], and [6]. We prove that in some exact sense, COLOR is the most robust heuristic of the four, and COLOR2 is preferable to [1], [6]. As a byproduct, we identify the weakest links in all four algorithms, which yields a deeper algorithmic understanding, thus serving as yet another motivation for using semirandom models. Formally, we prove:

THEOREM 1.1. *Let $np \geq C_0 k^2$ for some sufficiently large constant C_0 and a constant k, then there is an algorithm COLOR that* **whp** *k-colors $G_{n,p,k}$ in polynomial time.*

THEOREM 1.2. *In the setting of Theorem 1.1, the algorithm COLOR finds* **whp** *a k-coloring in polynomial time for the semirandom distribution $G^H_{n,p,k}$ defined in Section 4.*

THEOREM 1.3. *In the setting of Theorem 1.1, the algorithm COLOR finds* **whp** *a k-coloring for $G^*_{n,p,k}$ in time $(1+\alpha)^n$, where $\alpha = \exp\{-\Omega(np/k)\}$.*

Theorem 1.1 is already proven in [1], and an even stronger version of it is proven in [6]. However, we show that [1] fails to meet the requirements of Theorems 1.2 and 1.3, and that at least the analysis given in [6] fails Theorem 1.2, and in Theorem 1.3, α should be adjusted to $\Omega((np/k)^{-0.5})$. As for COLOR2,

THEOREM 1.4. *In the setting of Theorem 1.1, COLOR2 finds* **whp** *a k-coloring of $G_{n,p,k}$ in polynomial time.*

THEOREM 1.5. *In the setting of Theorem 1.2, the algorithm COLOR2 finds* **whp** *a k-coloring of $G^H_{n,p,k}$ in polynomial time.*

However COLOR2 fails to meet the requirements of Theorem 1.3. The proofs of Theorems 1.4 and 1.5 are not given fully and only an outline is sketched.

The rest of the paper is structured as follows. In Section 2 we present both algorithms, COLOR and COLOR2. In Section 3 we analyze COLOR in the random setting of $G_{n,p,k}$. In Section 4, we describe in details the semirandom variant $G^H_{n,p,k}$, and prove Theorems 1.2 and 1.3. We also compare all four algorithms using $G^H_{n,p,k}$ and $G^*_{n,p,k}$ as benchmarks. In Section 5 we discuss some possibly interesting topics for future research.

2 The Algorithms

In this section we present both algorithms, COLOR and COLOR2. As the underlying ideas are common to all four algorithms, and since we compare their performances, we start by giving a short description of [1] and [6]. In the description, we make remarks as to the aforementioned set H, which correspond to the analysis part. The reader at this stage may disregard these notes, as we do not assume familiarity with the analysis in [1]. The scheme in both algorithms is basically the same, we start with [1]. First, using a spectral technique, a $(1-\epsilon)$-approximation of the planted coloring is obtained **whp** (ϵ is a small constant, the probability is taken over $G_{n,p,k}$) . Next, a recoloring procedure is applied to reach an even closer distance from the planted coloring (and in particular, **whp** the set H is now colored correctly). Then, a careful uncoloring step guarantees that **whp** only correctly colored vertices remain colored (in particular, H remains colored). Finally, using exhaustive search on the graph induced by the uncolored vertices, the partial coloring is completed **whp** to a legal one. Since this graph breaks down **whp** to connected components of size $O(\log_k n)$, the latter can be carried out successfully while spending polynomial time. [6] runs in expected polynomial time over $G_{n,p,k}$. To achieve this, some modifications to [1] are done, amongst which, the spectral step is replaced with SDP, and possible mistakes in the recoloring and uncoloring steps are corrected using a careful exhaustive search.

Notation. For a graph G, we let $V(G)$ denote its set of vertices and $E(G)$ the set of edges. For a set $U \subseteq V$, $G[U]$ denotes the subgraph of G induced by the vertices of U. For a vertex v, $N(v)$ denotes its set of neighbors in G. For $A, B \subseteq V$ we let $e(A,B)$ be the number of edges connecting a vertex from A and a vertex from B in G. For two vertices $u, v \in V(G)$, $G + (u,v)$ denotes the graph G with the additional edge (u,v). For two graphs G_1, G_2, $G1 \setminus G2$ denotes $G1$ with all the edges of G_2 removed from it. The scalar product of two vectors $x, y \in \mathbb{R}^n$ is denoted $\langle x, y \rangle$.

Motivation. The algorithm COLOR consists of two steps. First (lines 1-12), using SDP, one obtains a partial coloring of the graph, which satisfies (a) it coincides with the planted coloring and (b) **whp** the partial coloring colors all but a small fraction of the vertices (in particular, the set H is colored). Next (lines 13-14), as done in Alon-Kahale, the partial coloring is completed to a legal one using exhaustive search on the set of uncolored vertices. In the setting of Theorems 1.1 and 1.2, the subgraph induced by the uncolored vertices breaks down **whp** to connected components of size at most $O(\log_k n)$, allowing the exhaustive search to remain polynomial. COLOR2 is more similar to [1] and [6]. Using SDP, a partial coloring is obtained. However, using different analysis than [6], one can show that the partial coloring is such that the recoloring step can be skipped, and one can immediately approach the

uncoloring procedure. Finally, using exhaustive search, a legal coloring is **whp** found.

As the first step in COLOR and COLOR2 is based on the SDP of the max h-cut problem, suggested by Frieze and Jerrum [15], we start by presenting it.

$$SDP_h(G) = \max \sum_{(u,v)\in E} \frac{h-1}{h}(1 - \langle x_u, x_v\rangle)$$

$$\text{s.t. } \forall\, u,v \in V,\ \langle x_u, x_v\rangle \geq -\frac{1}{h-1}$$

where the max is taken over all families $(x_v)_{v\in V}$ of unit vectors in $\mathbb{R}^{|V|}$. If $h \geq 2$ is an integer, then SDP_h is a relaxation of the MAX h-CUT problem. Since SDP_h is a semidefinite program, its optimal value can be computed up to an arbitrary high precision $\epsilon > 0$, in time polynomial in $|V|, h, \log\frac{1}{\epsilon}$ (e.g. using the Ellipsoid algorithm [17], [20]).

If G is k-colorable, then $SDP_k(G) = |E(G)|$ (since $|E(G)| = \text{max-k-cut}(G) \leq SDP_k(G) \leq |E(G)|$) . Moreover, if G is a subgraph of G', then $SDP_h(G) \leq SDP_h(G')$. The following Lemma is the key to understanding and analyzing both algorithms.

LEMMA 2.1. *Let $G = G^*_{n,p,k}$, $np \geq C_0k^2$. **Whp** there exists a set $V_0 \subseteq V$ of vertices, such that:*
(a) Let $V_0^{(i)} = V_0 \cap V_i$, then $|V_0^{(i)}|/|V_i| \geq 1 - \exp\{-\Omega(np/k)\}$.
(b) For every $i \in \{1,2,...,k\}$, for every $u^*, v^* \in V_0^{(i)}$, and for every $h \leq k$, $SDP_h(G + (u^*,v^*)) \leq |E(G)| - \Omega(\frac{n^2p}{hk})(k-h)$. In particular, for $h = k$, $SDP_k(G + (u^*,v^*)) = |E(G)|$.

In the sequel (Lemma 3.1), we explicitly identify such a set V_0, proving the lemma.
Another simple observation is that for any $u \in V_i$, $v \in V_j$ s.t. $i \neq j$, it holds that $SDP_k(G + (u,v)) = SDP_k(G) + 1$. This observation, combined with Lemma 2.1, are the motivation behind steps 5-10 of COLOR. As we shall prove in the next section, in steps 1-12, a set meeting the requirements of V_0 is colored according to the planted coloring **whp**. Further, $G[V \setminus V_0]$ breaks down **whp** to connected components of size $O(\log_k n)$.

REMARK 2.1. Instead of iteratively picking up the v_i's (line 4), one can go over all $\binom{n}{k}$ possibilities for $v_1, ..., v_k$. Another option is to choose all the v_i's in one step (and possibly amplify the success probability by repeating the execution).

REMARK 2.2. The algorithm receives k as a parameter. However, this is done only to simplify the description. If one could efficiently calculate a lower bound r on $\chi(G)$, then a simple way to circumvent the problem of not knowing k is to run the algorithm with $k = r, r+1, ..., n$ (of course the trivial bound $r = 1$ suffices, but one can do better). Such a non trivial lower bound can be calculated via SDP (in fact for $G^*_{n,p,k}$ the value $\chi(G)$ itself can be calculated **whp**, Lemma 2.3 ahead). To obtain this result and as motivation for the algorithm COLOR2, we need the following discussion.

Algorithm 1 : COLOR(G, k)

1: Compute the value of $SDP_k(G)$ up to precision of 0.05.
2: Let $W_i = \emptyset$ for $i = 1, 2, ..., k$.
3: **for** $i = 1$ to k **do**
4: Let $W = \bigcup_{j=1}^{i-1} W_j$, pick $v_i \in V \setminus W$ u.a.r and set $W_i = \{v_i\}$.
5: **for all** $u \in V \setminus W$ and non-edges $(v_i, u) \notin E$ **do**
6: Compute $SDP_k(G + (u, v_i))$ up to precision of 0.05.
7: **if** $|SDP_k(G + (v_i, u)) - SDP_k(G)| \leq 0.1$ **then**
8: $W_i \leftarrow W_i \cup \{u\}$
9: **end if**
10: **end for**
11: **end for**
12: Color W_i with color i, and let $U = V \setminus \bigcup_{i=1}^{k} W_i$
13: Find the connected components in $G[U]$.
14: In every component separately, use exhaustive search to complete the partial coloring of $\bigcup_{i=1}^{k} W_i$ to a legal k-coloring of G.

DEFINITION 2.1. *Let $G = (V, E)$ be a graph, $|V| = n$. We say that a family of n unit vectors $(x_v)_{v\in V}$ in $\mathbb{R}^n$ is a* rigid vector k-coloring *of G, if for every $u, v \in V$, $\langle x_u, x_v\rangle \geq -1/(k-1)$ and if $(u,v) \in E$ then $\langle x_u, x_v\rangle = -1/(k-1)$.*

$\bar{\vartheta}_2(G)$ commonly denotes the minimal real $k > 1$ such that G admits a rigid vector k-coloring. Since $\bar{\vartheta}_2(G)$ can be stated as a semidefinite program, it can be computed up to an arbitrary precision in polynomial time. The definition of $\bar{\vartheta}_2(G)$ in terms of vector coloring is related to the work of Karger, Motwani and Sudan [20].

LEMMA 2.2. *For every graph G, $\bar{\vartheta}_2(G) \leq \chi(G)$.*

Proof. Let $\chi(G) = c \leq n$. One can find c unit vectors $\{v_1, ..., v_c\}$ in $\mathbb{R}^n$ s.t. $\langle v_i, v_j\rangle = -1/(c-1)$ for every $1 \leq i \neq j \leq h$ (for a proof, see for example [21] Claim 2.2). Consider a c-coloring of G, and assign v_i to all vertices of color i. Clearly $\{v_1, ..., v_c\}$ is a rigid vector c-coloring of G, proving the Lemma.

LEMMA 2.3. *For $G = G^*_{n,p,k}$, **whp** $\bar{\vartheta}_2(G) = \chi(G) = k$.*

Proof. It is enough to show that $\bar{\vartheta}_2(G) = k$ (since $\chi(G) \leq k$ by definition, and $\bar{\vartheta}_2(G) \leq \chi(G)$ by the previous Lemma). By contradiction, assume that $\bar{\vartheta}_2(G) = h < k$. Let $(x_v)_{v\in V}$ be a rigid vector h-coloring of G. $(x_v)_{v\in V}$ is also a feasible solution to SDP_h, therefore, $SDP_h(G) \geq |E(G)|$. On the other hand, by Lemma 2.1, **whp** $SDP_h(G) < |E(G)|$ for $h < k$.

Therefore, when the input is $G = G^*_{n,p,k}$, we can **whp** calculate k. Using the terminology of [7], we call a rigid vector k-coloring $(x_v)_{v\in V}$ of $G = G^*_{n,p,k}$ *integral* w.r.t. the planted k-coloring of G, if there are k vectors $(x^*_i)_{i=1,\ldots,k}$ s.t. $x_v = x^*_i$ for all $v \in V_i$, and $\langle x^*_i, x^*_j\rangle = -\frac{1}{k-1}$ for $i \neq j$. [7] proves that for $np \geq \Omega(k \ln n)$, the rigid vector k-coloring is **whp** integral, therefore the planted coloring of the graph can be easily reconstructed. This is not necessarily true in the sparse case. However, in the sparse case, a rigid vector k-coloring of G is integral on V_0. Formally,

PROPOSITION 2.1. *Let $(x_v)_{v\in V}$ be a rigid vector k-coloring of $G = G^*_{n,p,k}$.* **Whp** *for every i, and every $s,t \in V_0^{(i)}$, $x_s = x_t$.*

Proof. Observe that $(x_v)_{v\in V}$ is also a feasible solution to $SDP_k(G+(s,t))$, though not necessarily the optimal one. Thus,

$$E(G) = \sum_{(u,v)\in E(G)} \frac{k-1}{k}(1 - \langle x_u, x_v\rangle) \leq$$
$$\sum_{(u,v)\in E(G)} \frac{k-1}{k}(1 - \langle x_u, x_v\rangle) + \frac{k-1}{k}(1 - \langle x_s, x_t\rangle)$$
$$\leq SDP_k(G+(s,t)) \underbrace{\leq}_{\text{Lemma 2.1}} |E(G)|$$

We conclude that $\frac{k-1}{k}(1 - \langle x_s, x_t\rangle) = 0$, or equivalently that $x_s = x_t$. Since **whp** there exist $V_0^{(i)} - V_0^{(j)}$ edges for every i,j, the second requirement, $\langle x^*_i, x^*_j\rangle = -\frac{1}{k-1}$, holds by the definition of a rigid vector k-coring.

REMARK 2.3. Since the SDP of the rigid vector k-coloring is not solved exactly (rather up to some predefined precision) there are some extra technical issues to take care of. For example, two vertices that should have received the same vector assignment x^*_i may have received in fact two different, though very close, vectors. As such issues are only technical in nature, details are omitted.

3 The Random Setting

In this section we prove Theorem 1.1, and sketch the analysis of COLOR2. The key to proving Theorem 1.1

Algorithm 2 : COLOR2(G,k)

1: Compute a rigid vector k-coloring of G, $\{x^*_i\}$.
2: Group the vertices in V according to the vectors $\{x^*_i\}$ assigned to them by the rigid vector k-coloring.
3: Let $W_i \subseteq V$ be the set of vertices assigned with the vector x^*_i.
4: Color the k largest W_i's (w.l.o.g $i = 1,\ldots,k$) with k different colors (one for every set).
5: **while** $\exists i,j \in [1..k], v \in V$ s.t. $v \in W_i$ and v has less than $0.98np/k$ neighbors colored $j \neq i$ **do**
6: Uncolor v.
7: **end while**
8: Let U be the set of uncolored vertices. Find the connected components in $G[U]$.
9: In every component separately, use exhaustive search to complete the partial coloring of $G[V \setminus U]$ to a legal k-coloring of G.

is to show that **whp**, in steps 1-12, a huge fraction of the vertices is colored correctly, and the remaining vertices break into connected components of size at most $\log_k(n)$. To this end, we introduce a set of vertices $H \subseteq V$ similar to the one described in [1]. Let us briefly define H (a similar description can also be found in [1]). First, let H_0 be the set of vertices having at most $1.01np/k$ neighbors in G_0 in every color class (other than their own). Consider the subgraph $G_0[H_0]$ (the subgraph induced by the vertices of H_0) and set $i = 0$. While there exists a vertex $v_i \in H_i$ that has less than $0.99np/k$ neighbors in $G_0[H_i]$ in some color class (other than its own), define H_{i+1} to be $H_i \setminus \{v\}$ and increment i by 1. Let H be the remaining set of vertices when the iterative procedure stops.

LEMMA 3.1. ***Whp**, the set H satisfies the requirements to the set V_0 in Lemma 2.1.*

Lemma 3.1 proves Lemma 2.1. The proof of requirement (a) in Lemma 2.1 is given in [1] Lemma 2.7 (for $k = 3$). The proof of requirement (b) follows closely the one given in [7] Lemma 10, while using results from [1] and [12]. The proof of Lemma 3.1 is highly technical in nature, thus it is deferred to the appendix. In the remainder of the section, it would be convenient for the reader to think of V_0 as H.

Recall that $V_0^{(i)}$ is the set of vertices in V_0 that belong to color class V_i. We start by proving that if v_i happens to fall in $V_0^{(i)}$, then **whp** most of the color class V_i is recovered. Formally,

LEMMA 3.2. *If v_i, chosen in line 4, belongs to $V_0^{(i)}$, then **whp** the corresponding W_i recovered in steps 5-8 satisfies $V_0^{(i)} \subseteq W_i \subseteq V_i$.*

Proof. First we prove that $W_i \subseteq V_i$ w.p. 1. By contradiction, suppose that in some step of the `for` iteration in lines 5-8, some $u \in V_j, j \neq i$ was included in W_i. Then the condition in line 7 held. However, it holds that $SDP_k(G + (v_i, u)) = SDP_k(G) + 1$ for $u \notin V_i$. Choosing the precision of the SDP solution to be high enough (0.05 in our case), we get a contradiction. $V_0^{(i)} \subseteq W_i$ holds **whp** by Lemma 2.1, requirement (b) and again choosing a sufficiently high precision for the SDP.

It therefore remains to prove that with rather high (we shall prove constant) probability, the vertices $v_1, v_2, ..., v_k$ chosen in line 4, satisfy $v_i \in V_0^{(i)}$. This, combined with Lemma 3.2, proves that **whp** most of the vertices (and in particular the set V_0) are colored correctly when line 13 begins to execute. Formally,

LEMMA 3.3. *With probability* $1 - \exp\{\Omega(np/k)\}$, *all* k *vectors* $v_1, v_2, ..., v_k$, *chosen in line 4, satisfy w.l.o.g.* $v_i \in V_0^{(i)}$.

Proof. For starters, consider the first iteration where v_1 is chosen. Assume that Lemma 2.1 indeed holds. By requirement (a) of Lemma 2.1, v_1 belongs to some $V_0^{(i)}$ w.p. $1 - \exp\{\Omega(np/k)\}$, assume w.l.o.g. $i = 1$. Now assume that $v_1, v_2, ..., v_{i-1}$ were all chosen to be in $V_0^{(1)}, V_0^{(2)}, ..., V_0^{(i-1)}$ respectively, and ask what is the probability that $v_i \in V_0^{(i)}$. When picking v_i, the bad events could be either that $v_i \in V_j$ for some $j \leq i-1$ (by Lemma 3.2 and the assumption on the v_j's, v_i will then belong to $V_j \setminus V_0^{(j)}$), or that $v_i \in V_i \setminus V_0^{(i)}$. The number of bad vertices is then at most $n \cdot \exp\{-\Omega(np/k)\}$. The total number of remaining vertices to choose from is at least n/k (since in every iteration at most one color class is colored). Therefore, the probability of a bad event happening is bounded by

$$\frac{n \cdot \exp\{-\Omega(np/k)\}}{n/k} = k \cdot \exp\{-\Omega(np/k)\} =$$

$$= \exp\{-\Omega(np/k)\}.$$

The last equality is due to $np \geq C_0 k^2$. Let A_i be the event $v_i \in V_0^{(i)}$. Then,

$$Pr[\bigwedge_{i=1}^{k} A_i] = Pr[A_1] \cdot Pr[A_2|A_1] \cdots Pr[A_k|\bigwedge_{i=1}^{k-1} A_i]$$
$$\geq (1 - \exp\{-\Omega(np/k)\})^k \simeq 1 - k \cdot \exp\{-\Omega(np/k)\}$$
$$= 1 - \exp\{-\Omega(np/k)\}.$$

Since Lemma 2.1 holds w.p. $1 - o(1)$, the lemma follows.

REMARK 3.1. One can ensure that Lemma 3.3 occurs with probability 1 by going over all possible $\binom{n}{k}$ choices for $v_1, ..., v_k$ and run COLOR for each such choice (skipping line 4).

PROPOSITION 3.1. *H is colored when the exhaustive search begins with probability* $1 - \exp\{-\Omega(np/k)\}$. *Further, (with probability 1) the partial coloring induced by the W_i's (line 12) coincides with the planted coloring (up to renaming of color classes).*

Proposition 3.1 follows readily from Lemmas 3.1, 3.2 and 3.3.

PROPOSITION 3.2. ***Whp**, the largest connected component in $G[V \setminus H]$ is of size at most* $\log_k(n)$.

The proof of this Lemma is given in [1] Proposition 2.11, for $k = 3$. It generalizes easily for any constant k. The intuition behind the proof comes from a well known result concerning $G_{n,p}$. If $np < 1$, then **whp** the largest connected component in $G_{n,p}$ is of size at most $O(\log n)$ (see e.g. [3] for a complete discussion). Now consider a random subgraph of size αn of $G_{n,p}$, this subgraph is exactly $G_{\hat{n},\alpha p}$, where $\hat{n} = \alpha n$. If $\alpha np < 1$, **whp** the largest connected component in $G_{\hat{n},\alpha p}$ is of size $O(\log \hat{n})$. In our case, **whp** $|V \setminus H| \cdot p < 1$ but unfortunately $V \setminus H$ is not a truly random subset of vertices. Therefore, the proof is burdened with more technicalities.

Combining Propositions 3.1 and 3.2, **whp** the exhaustive search consumes polynomial time, and succeeds in completing the partial coloring (line 12), to a legal coloring of G (since at least the partial coloring can be completed to the planted one). Theorem 1.1 then follows.

Let us now sketch the proof of Theorem 1.4. Proposition 2.1 and requirement (a) in Lemma 2.1 imply that the set H is set correctly before the uncoloring procedure begins. By the definition of H, and the fact that the coloring of H coincides with the planted coloring, it follows that H survives the uncoloring procedure. By expansion properties of $G_{n,p,k}$, it holds that **whp** every vertex that survived the uncoloring is colored according to the planted coloring (Lemma 2.9 in [1]). This combined with Proposition 3.2, allow the exhaustive search to succeed **whp** in completing the partial coloring to a legal one in polynomial time.

4 The Semirandom Setting

Let us recall the definition of the semirandom model $G^*_{n,p,k}$. First a random graph $G_0 = G_{n,p,k}$ is generated in the aforementioned way. Let $V_1, V_2, ..., V_k$ be the planted color classes. Next an adversary may add edges connecting a vertex in V_i with a vertex in V_j for $i \neq j$.

Improving upon [9], [7] proves that unless $NP \subseteq RP$, there is no polynomial time algorithm that **whp** k-colors $G^*_{n,p,k}$ when $np \leq (1-\epsilon)\frac{k}{2}\ln(n/k)$. In our case np is constant, therefore it is not realistic to expect COLOR to work **whp** in polynomial time over $G^*_{n,p,k}$. Let $A \subseteq V$ be an arbitrary set of vertices, denote by $G^A_{n,p,k}$ the following semirandom model. As in $G^*_{n,p,k}$, first G_0 is generated. Next, an adversary may add $V_i - V_j$ edges, only this time both endpoints belong to A. Observe that $G^V_{n,p,k} = G^*_{n,p,k}$ and $G^\emptyset_{n,p,k} = G_{n,p,k}$. Therefore, the choice of A in $G^A_{n,p,k}$ allows us to regulate the hardness of the resulting distribution. In this work we consider $A = H$. Note that H is in some sense a random set which depends on G_0. Arguably, $G^H_{n,p,k}$ is not the most natural semirandom model to consider. In particular, the fact that H depends on G_0 is not a desirable property. However, as we show shortly, it already suffices to separate COLOR and COLOR2 from [1] and [6], identifying two weakest links in the latter.

4.1 Proof of Theorem 1.2

Proof. Note that Lemmas 3.1, 3.2 and 3.3 are valid in $G^*_{n,p,k}$ to begin with, therefore remain valid in the more restricted $G^H_{n,p,k}$. Hence, Proposition 3.1 is also valid in $G^H_{n,p,k}$. Regarding Proposition 3.2, its proof for $G_{n,p,k}$ relies on the random properties of $G_0[V \setminus H]$. As the adversary is not allowed to add edges to $G_0[V \setminus H]$, it remains valid in $G^H_{n,p,k}$. To summarize, the analysis given in Section 3 remains valid for $G^H_{n,p,k}$ as well. Thus, Theorem 1.2 follows.

As for Theorem 1.5, Lemmas 2.1, 3.1, and Proposition 2.1 assume $G = G^*_{n,p,k}$. Proposition 3.2 is also valid in $G^H_{n,p,k}$. The only issue to address is the correctness of the uncoloring procedure. As the expansion properties of $G_0[V \setminus H]$ remain valid in $G^H_{n,p,k}$, the assertions regarding the uncoloring procedure carry through.

Let us now compare the four algorithms, [1], [6], COLOR and COLOR2, using $G^H_{n,p,k}$ as a benchmark. [1] fails already in the spectral step since **whp** H contains almost all vertices allowing the adversary to jumble with the spectra of the graph (emptying the eigenvectors, in particular the last two used in [1], from meaningful information regarding the planted coloring). One can prove that the SDP approximation (replacing the spectral one), used in [6], still provides a $(1-\epsilon)$-approximation in $G^*_{n,p,k}$ (though the issue of semirandomness is not addressed in [6]). However, the analysis of the recoloring step employed in [6] relies on random properties of $G_{n,p,k}$ which need not hold in $G^H_{n,p,k}$. Therefore, the analysis in [6] doesn't show that the algorithm finds a legal coloring **whp** in *polynomial* time. On the other hand, both COLOR and COLOR2 find **whp** a legal coloring in polynomial time when the graph G is sampled according to $G^H_{n,p,k}$. The two weakest links - the spectral technique, and the recoloring procedure - are not used in the latter. However, this model doesn't suffice to separate COLOR from COLOR2. This will be done promptly.

4.2 Proof of Theorem 1.3

Proof. Lemmas 3.1, 3.2 and 3.3 are valid in $G^*_{n,p,k}$, hence Proposition 3.1 is also valid in $G^*_{n,p,k}$. However, Proposition 3.2 need not necessarily hold. Therefore, based on the above analysis, it need not hold that the exhaustive search can end up successfully while spending polynomial time. However, requirement (a) in Lemma 2.1 and Lemma 3.1 imply that **whp** $|U| \leq \exp\{-\Omega(np/k)\}n$ (where U is the set of uncolored vertices). Thus, the exhaustive search consumes at most

$$k^{|U|} = k^{\exp\{-\Omega(np/k)\}n} \leq (1+\exp\{-\Omega(np/k)\})^n$$

steps. Theorem 1.3 then follows.

Now let us compare [1], [6], COLOR and COLOR2 in this setting. [1] fails for at least the aforementioned reason. The analysis of the uncoloring procedure employed by COLOR2 (as well as by [1] and [6]) breaks in $G^*_{n,p,k}$, failing to show that all vertices surviving the uncoloring are colored according to the planted coloring. Therefore, it might be the case that COLOR2 will not be able to produce a legal coloring regardless of the amount of time it is given (since the graph $G[U]$ may no longer be k-colorable when taking into account the constraints imposed by the coloring of $V \setminus U$). As part of the augmentations of [1] towards becoming an expected polynomial time algorithm, [6] employs a recovery step (which is basically a careful exhaustive search), which eventually sets correctly the ϵn vertices possibly missed by the SDP approximation. At that stage a legal coloring is found, but not necessarily beforehand (at least the analysis fails to show that). The analysis in [6] requires $\epsilon = \Omega((np/k)^{-0.5})$. Therefore, the time guaranteed by the current analysis in [6] to find a legal coloring is at least $(1+\Omega((np/k)^{-0.5}))^n$. This is of course much larger than the time COLOR spends.

5 Discussion

Semirandom models serve in many cases as useful benchmarks for evaluating the robustness of algorithms designed to work **whp** for random structures. However, when considering sparse random distributions, it might be the case that the interesting and natural adversaries render the problem hard (see for example [7], [9]). Therefore, one cannot expect them to serve as

useful benchmarks when inspecting *polynomial* time algorithms. In this work, we suggest two alternatives to address the issue, and as a case study apply them to [1], [6], COLOR, and COLOR2. The first alternative is to further restrict the adversary, which translates to $G^H_{n,p,k}$ in our setting. Using $G^H_{n,p,k}$, $np \geq C_0 k^2$, as a benchmark, we were able to claim that in some explicit sense COLOR and COLOR2 are more robust than [1] and [6]. However, $G^H_{n,p,k}$ doesn't separate COLOR from COLOR2.

The second alternative is to consider the natural semirandom distribution $G^*_{n,p,k}$, allow the algorithms to use as much time as needed to find a solution **whp** in the semirandom setting, and then compare the running times (which now may be superpolynomial). In our setting, for $G^*_{n,p,k}$, $np \geq C_0 k^2$, [1] and COLOR2 fail to produce a solution (at least their analysis fails to show it), regardless of the amount of time spent trying to find one. In contrast, COLOR and [6] find a legal coloring **whp** in time $(1+b)^n$, where $b > 0$ is a constant depending on np/k. The exponent base guaranteed by COLOR is much smaller than the one guaranteed by [6] (b decreases exponentially in np/k rather than polynomially).

As we already mentioned, though useful, $G^H_{n,p,k}$ is not the most natural model to consider. An interesting question for further research is to come up with a more natural semirandom model for $np \geq C(k)$, $C(k)$ a constant, while keeping the model interesting, in the sense that one can expect a polynomial time algorithm to solve it **whp**. Then, one naturally asks for a polynomial time algorithm that works **whp** in that model. Another interesting question is to characterize the maximal set A of vertices s.t. $G^A_{n,p,k}$ can be solved in polynomial time **whp**.

References

[1] N. Alon and N. Kahale. *A spectral technique for coloring random 3-colorable graphs.* SIAM J. Comput., 26 (1997), pp. 1733–1748.

[2] N. Alon, M. Krivelevich, and B. Sudakov. *Finding a large hidden clique in a random graph.* Random Structures and Algorithms, 13 (1998), pp. 457–466.

[3] N. Alon and J. H. Spencer. *The probabilistic method.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience [John Wiley & Sons], New York, second edition, 2000.

[4] A. Blum and J. Spencer. *Coloring random and semi-random k-colorable graphs.* J. of Algorithms, 19 (1995), pp. 204–234.

[5] B. Bollobás. *The chromatic number of random graphs.* Combinatorica, 1 (1988), pp. 49–55.

[6] J. Böttcher. *Coloring sparse random k-colorable graphs in polynomial expected time.* In Proc. 30th International Symp. on Mathematical Foundations of Computer Science. Lecture Notes in Comput. Sci. 3618 (2005), pp. 156–167.

[7] A. Coja-Oghlan. *Coloring semirandom graphs optimally.* In Proc. 31st International Colloquium on Automata, Languages, and Programming, pp. 383–395, 2004.

[8] A. Coja-Oghlan. *The Lovász number of random graphs.* Combin. Probab. Comput. 14 (2005) pp. 439–465.

[9] U. Feige and J. Kilian. *Heuristics for semirandom graph problems.* J. Comput. and Syst. Sci., 63 (2001), pp. 639–671.

[10] U. Feige and J. Kilian. *Zero knowledge and the chromatic number.* J. Comput. and Syst. Sci., 57 (1998), pp. 187–199.

[11] U. Feige and R. Krauthgamer. *Finding and certifying a large hidden clique in a semirandom graph.* Random Structures and Algorithms, 16 (2000), pp. 195–208.

[12] U. Feige and E. Ofek. *Spectral techniques applied to sparse random graphs.* Random Structures and Algorithms, 27 (2000), pp 251–275.

[13] U. Feige and D. Vilenchik. *A local search algorithm for 3SAT.* Technical report, The Weizmann Institute of Science, 2004.

[14] A. Flaxman. *A spectral technique for random satisfiable 3CNF formulas.* In Proc. 14th ACM-SIAM Symp. on Discrete Algorithms, pp. 357–363, 2003.

[15] A. Frieze, M. Jerrum. *Improved approximation algorithms for MAX k-CUT and MAX BISECTION.* Algorithmica, 18 (1997), pp. 67–81.

[16] A. Frieze and C. McDiarmid. *Algorithmic theory of random graphs.* Random Structures and Algorithms, 10 (1997), pp. 5–42.

[17] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization.* Algorithms and Combinatorics (2). Springer-Verlag, Berlin, second edition, 1993.

[18] C. Helmberg. *Semidefinite programming.* European Journal of Operational Research, 137 (2002), pp. 461–482.

[19] T. R. Jensen and B. Toft. *Graph coloring problems.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience [John Wiley & Sons], New York, 1995.

[20] D. Karger, R. Motwani, and M. Sudan. *Approximate graph coloring by semidefinite programming.* J. of the ACM, 45 (1998), pp. 246–265.

[21] M. Krivelevich. *Deciding k-colorability in expected polynomial time.* Info. Process. Letters, 81 (2002), pp. 1–6.

[22] M. Krivelevich and V. H. Vu. *Approximating the independence number and the chromatic number in expected polynomial time.* J. Comb. Optim., 6 (2002), pp. 143–155.

[23] L. Kučera. *Expected behavior of graph coloring algorithms.* Proc. Fundamentals of Computation Theory, 56 (1977), pp. 447–451.

[24] T. Łuczak. *The chromatic number of random graphs.* Combinatorica, 11 (1991), pp. 45–54.

[25] D. Vilenchik. *Finding a satisfying assignment for semi-random satisfiable 3CNF formulas.* Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, January 2004.

A Proof of Lemma 3.1

To prove Lemma 3.1, one has to prove that both requirements of Lemma 2.1 are met by H. The proof of requirement (a) is given in [1] Lemma 2.7, for $k = 3$, and readily generalizes to any constant k. The proof of requirement (b) follows closely the one given in [7] Lemma 10, while combining results from [1] and [12]. In this work we only prove requirement (b).

Notation. For a vector $x \in \mathbb{R}^n$, let $\|x\|$ be the ℓ_2-norm of x. We let $\mathbf{1}_n \in \mathbb{R}^n$ be the vector whose all entries equal 1, $\mathbf{I}_n \in \mathbb{R}^{n\times n}$ be the unit matrix, and $\mathbf{J}_n \in \mathbb{R}^{n\times n}$ the matrix whose all entries are 1 (the subscript n is omitted when it is clear from context). By $diag(x)$ we denote the $n \times n$ diagonal matrix whose entries are x. For two matrices $A, B \in \mathbb{R}^{n\times n}$, the notation $A \leq B$ means $(B - A)$ is a positive semidefinite matrix. For a graph G, let $A(G)$ denote the adjacency matrix associated with the graph. Let $L(G)$ be the Laplacian of G , namely, $L(G) = diag(A\mathbf{1}) - A(G)$.

Recall that given $G = G^*_{n,p,k}$, $G_0[H]$ stands for the random part of G induced by the vertices of H. The key to proving Lemma 3.1 is the following claim:

PROPOSITION A.1. ***Whp,*** *If $u^* \in V_i \cap H$, $v^* \in V_j \cap H$, $i \neq j$, then $SDP_h(G_0[H] + (u^*, v^*)) \leq |E(G_0[H])| - \Omega(\frac{n^2p}{hk})(k-h)$.*

COROLLARY A.1. *For $G = G^*_{n,p,k}$, u^*, v^* as in Proposition A.1,* ***whp*** *$SDP_h(G + (u^*, v^*)) \leq |E(G)| - \Omega(\frac{n^2p}{hk})(k-h)$.*

Proof.

$$SDP_h(G + (u^*, v^*)) \leq SDP_h(G_0[H] + (u^*, v^*))$$
$$+ SDP_h(G \setminus G_0[H]) \leq |E(G_0[H])| - \Omega(\frac{n^2p}{hk})(k-h)$$
$$+ |E(G \setminus G_0[H])| = |E(G)| - \Omega(\frac{n^2p}{hk})(k-h).$$

In the first inequality, we used the fact that for every G_1, G_2 s.t. G_1 is a subgraph of G_2, it holds that $SDP_h(G_2) \leq SDP_h(G_1) + SDP_h(G_2 \setminus G_1)$. In the second inequality, we used Proposition A.1 and the fact that for every graph G, $SDP_h(G) \leq |E(G)|$.

Corollary A.1 proves that H meets requirement (b) of Lemma 2.1, and finishes the proof of Lemma 3.1.

It remains to prove Proposition A.1, namely to bound $SDP_h(G_0[H] + (u^*, v^*))$ accordingly. As noted in [7], SDP duality is often a convenient tool to bound the value of a maximization problem. Given a graph G, one can bound the value of $SDP_h(G)$ using its dual problem $DSDP_h(G)$:

$$DSDP_h(G) = \min_Y \frac{h-1}{2h}\sum_{i=1}^{n} y_{ii} - \frac{1}{2h}\sum_{i\neq j} y_{ij}$$
$$\text{s.t. } Y = (y_{ij})_{i,j=1..n} \in \mathbb{R}^{n\times n},$$
$$L(G) \leq Y,\ y_{ij} \leq 0 \text{ for } i \neq j.$$

$SDP_h(G) \leq DSDP_h(G)$ by weak SDP duality (see for example [18]). In particular, we present a solution Y to $DSDP_h(G_0[H] + (u^*, v^*))$, which is **whp** feasible and whose value is $|E(G_0[H])| - \Omega(\frac{n^2p}{hk})(k-h)$. Since the dual is a minimization problem, Proposition A.1 follows.

Our last task is to provide with the aforementioned solution Y. Recall that $V_1, ..., V_k$ are the planted coloring classes of G_0. Let $W_i = V_i \cap H$, and $m = |H|$. Recall that $u^*, v^* \in W_i$ for some i. Assume some order on V, and let $i(W_a)$ denote the i'th vertex in color class W_a, let d_v be the degree of v in $G_0[H] + (u^*, v^*)$, and $d_v^{(i)}$ be $|N(v) \cap W_i|$. Moreover, we let

$$p_{ij}^{(ab)} = \frac{d^{(b)}_{i(W_a)} \cdot d^{(a)}_{j(W_b)}}{e(W_a, W_b)},\ p_{min} = \min_{i,j,a\neq b} p_{ij}^{(ab)},$$

$$d_{min} = p_{min} \cdot \min_a |W_a|.$$

Observe that $p_{ij}^{(ab)}$ has "probability units", and therefore can be viewed as an estimate for the probability of an edge between $i(W_a)$ and $j(W_b)$ given their degrees and $e(W_a, W_b)$. Following the same logic, d_{min} is the lowest expected degree of a vertex $v \in W_a$ in color class $W_b, b \neq a$.

We are ready to define the solution Y. For $1 \leq a, b \leq k$ define $|W_a| \times |W_b|$ matrices Y'_{ab} as follows:

$$Y'_{aa} = 0 \text{ for } a = 1, .., k.$$
$$\text{For } a \neq b,\ , Y'_{ab} = \left[\frac{d_{min}}{|W_b|} - p_{ij}^{(ab)}\right]_{i=1,...,|W_a|, j=1,...,|W_b|}$$

Let Y' be the $m \times m$ matrix whose blocks are the Y'_{ab}'s. Let $y' = (d_v + d_{min})_{v\in H} \in \mathbb{R}^m$. Finally, we let $Y = Y' + diag(y')$.

PROPOSITION A.2. *The value of the solution Y is* ***whp*** *$|E(G_0[H])| - \Omega(\frac{n^2p}{hk})(k-h)$.*

Proof. First observe that $\sum_{j=1}^{|W_b|} d^{(a)}_{j(W_b)} = e(W_a, W_b)$.

Therefore, for $a \neq b$,

$$Y'_{ab}\mathbf{1} = \left[d_{min} - \sum_{j=1}^{|W_b|} \frac{d^{(b)}_{i(W_a)} \cdot d^{(a)}_{j(W_b)}}{e(W_a, W_b)} \right]_{1 \leq i \leq |W_a|} = \left[d_{min} - d^{(b)}_{i(W_a)} \right]_{1 \leq i \leq |W_a|}.$$

Further, observe that

$$\sum_{i \neq j} y_{ij} = \sum_{a \neq b} \langle Y'_{ab}\mathbf{1}, \mathbf{1} \rangle = \sum_{a \neq b} |W_a| \cdot d_{min} - e(W_a, W_b)$$
$$= (k-1)md_{min} - 2|E(G_0[H])|.$$
$$\sum_{i=1}^{m} y_{ii} = \langle y', \mathbf{1} \rangle = 2|E(G_0[H])| + md_{min}.$$

Therefore,

$$DSPD_h(G_0[H] + (u^*, v^*)) \leq \frac{h-1}{2h} \sum_{i=1}^{n} y_{ii} - \frac{1}{2h} \sum_{i \neq j} y_{i,j}$$

$$= |E(G_0[H])| - \frac{md_{min}}{2h}(k-h).$$

By the definition of H, $p_{min} = \Omega(p)$. Further, **whp** $|W_a| = \Omega(n/k)$ for every $a = 1, ..., k$, (requirement (a) in Lemma 2.1) and hence $m = |H| \geq (1 - \exp\{-np/k\})n = \Omega(n)$. It therefore follows that $d_{min} \geq \Omega(np/k)$. Plugging the values of d_{min} and m in the above expression, the claim follows.

PROPOSITION A.3. *The matrix Y defined above is **whp** a feasible solution to the semidefinite programm $DSDP_h(G_0[H] + (u^*, v^*))$.*

Proof. Indeed, Y is a real symmetric matrix. Further, the definition of d_{min} ensures that every off diagonal y_{ij} obeys $y_{ij} \leq 0$. It remains to verify that $L(G_0[H] + (u^*, v^*)) \leq Y$, or equivalently, that $Y - L(G_0[H] + (u^*, v^*))$ is a positive semidefinite matrix (abbreviated **psd**). Let $A = A(G_0[H])$, $L = L(G_0[H])$, $L^+ = L(G_0[H] + (u^*, v^*))$, and $B = L^+ - L$. It is easy to see that $Y - L^+ = d_{min}\mathbf{I} - (B - A - Y')$. Therefore, it suffices to prove that the largest eigenvalue of $(B - A - Y')$ is at most d_{min} (since then, all eigenvalues of $Y - L^+$ are non-negative, and this is equivalent to being **psd**). The following lemma completes the proof of Proposition A.3. Proposition A.1 then follows from Propositions A.2, A.3, and SDP weak duality.

LEMMA A.1. *The largest eigenvalue of $B - A - Y'$ is **whp** at most d_{min}.*

Proof. (Lemma A.1) Our proof strategy is as follows. We identify a subspace $K \subseteq \mathbb{R}^m$ spanned by a subset of the eigenvectors of $(B - A - Y')$. K has two useful properties: (a) it contains no eigenvector whose corresponding eigenvalue is greater than d_{min} and (b) **whp**, $|\langle (B - A - Y')x, x \rangle| < d_{min}$ for every unit vector $x \perp K$. Let λ be an eigenvalue of $(B - A - Y')$, and v_λ its corresponding eigenvector (assume all eigenvectors are mutually perpendicular, and normalized to unit vectors, e.g. via the Graham-Schmidt procedure). If $v_\lambda \in K$, then by property (a), $\lambda \leq d_{min}$. Otherwise, $v_\lambda \perp K$, and by property (b), $|\langle (B - A - Y')v_\lambda, v_\lambda \rangle| = |\lambda| < d_{min}$.

Let us start by presenting the subspace K. For $a = 1, ..., k$, we let $\mathbf{1}_{W_a} \in \mathbb{R}^m$ denote the vector whose entries are 1 if the entry corresponds to a vertex in W_a and 0 otherwise. Let δ_{v,W_a} be 1 if $v \in W_a$ and 0 otherwise. Similar to to the proof of Proposition A.2,

$$Y'\mathbf{1}_{W_a} = [(1 - \delta_{v,W_a})(d_{min} - e(v, W_a))]_{v \in H}$$

Further,

$$A\mathbf{1}_{W_a} = [e(v, W_a)]_{v \in H} \text{ and } B\mathbf{1}_{W_a} = 0.$$

Therefore,

$$(B - A - Y')\mathbf{1}_{W_a} = [-(1 - \delta_{v,W_a})d_{min}]$$

Finally, let $\xi^{(a,b)} = \mathbf{1}_{W_a} - \mathbf{1}_{W_b} \in \mathbb{R}^m$. By the above,

$$(B - A - Y')\xi^{(a,b)} = d_{min}\xi^{(a,b)} \text{ for } a \neq b,$$

$$(B - A - Y')\mathbf{1} = -(k-1)d_{min}\mathbf{1}.$$

Let K be the vector space spanned by the $\xi^{(a,b)}$'s and $\mathbf{1}$. Property (a) stated above follows from the latter. It remains to prove that property (b) holds.

PROPOSITION A.4. *$|\langle (B - A - Y')x, x \rangle| < d_{min}$, for all unit vectors $x \perp K$.*

Before proving the Proposition we make the following observations regarding some spectral properties of A and Y'. Similar to Y'_{ab}, A_{ab} denotes the minor of A corresponding to W_a (rows) and W_b (columns).

LEMMA A.2. *Let $G_0 = G_{n,p,k}$, Let $A = A(G_0[H])$ be the adjacency matrix of the subgraph of G_0 induced by the vertices of H. Then the following holds **whp**:*
(1) For all unit vectors $x \perp K$, $|\langle Ax, x \rangle| \leq O(\sqrt{np})$.
(2) For all $a, b \in \{1, ..., k\}$ and all $\mathbf{1} \perp x \in \mathbb{R}^{|W_a|}$, $|\langle A_{ab}\mathbf{1}, x \rangle| \leq \|\mathbf{1}_{W_a}\| O(\sqrt{np/k})$

The proof of Lemma A.2 is given in parts in [1] and [12]. We only point out the differences. Observe that $\mathbf{1}_{W_a} \in K$ for every $a = 1, ..., k$ (using linear combinations of the

$\xi^{(a,b)}$'s and **1**). Therefore, the proof of (1) is essentially the one given in [1], Lemma 3.2. The only difference is that we consider the set H and [1] consider the set of vertices, call it W, whose degree in every color class doesn't exceed $5np/k$. The two properties of W used in [1] are $|W| \geq (1 - \exp\{-np/k\})n$ and the bound on the degree of vertices in W. Both properties hold for H as well **whp**. The proof of (2) is essentially the one given in [12], Lemma 3.2. Again, the only properties of W used in the proof are its size, and the bound on the degree of its vertices ([12] prove that property 2 holds with probability $1-O(\exp\{-np/k\})$, however, it can be shown that it holds with probability $1 - o(1)$ by using stronger methods than Markov's inequality).

LEMMA A.3. *For every unit vector $x \perp K$, **whp** $|\langle Y'x, x\rangle| \leq O(\sqrt{np})$*

Proof. Consider the following $|W_a| \times |W_b|$ matrix Z_{ab}: $Z_{aa} = 0$, $Z_{ab} = \frac{1}{|W_b|}\mathbf{J} - Y'_{ab}$. Let Z be the $m \times m$ matrix whose blocks are the Z_{ab}'s. Since $x \perp \mathbf{1}_{W_a}$ for every a, $\langle Y'x, x\rangle = -\langle Zx, x\rangle$. Therefore it suffices to estimate $|\langle Zx, x\rangle|$. Let $\xi \in \mathbb{R}^{|W_b|}, \eta \in \mathbb{R}^{|W_a|}$ be two vectors perpendicular to **1**.

$$e(W_a, W_b)|\langle Z_{ab}\xi, \eta\rangle| =$$
$$\left\langle \left[\sum_{j=1}^{|W_b|} d^{(b)}_{i(W_a)} d^{(a)}_{j(W_b)} \xi_j\right]_{1\leq i\leq |W_a|}, \eta \right\rangle =$$
$$\left\langle \left[d^{(b)}_{i(W_a)}\langle A_{ba}\mathbf{1}, \xi\rangle\right]_{1\leq i\leq |W_a|}, \eta \right\rangle =$$
$$\langle A_{ba}\mathbf{1}, \xi\rangle \sum_{i=1}^{|W_a|} d^{(b)}_{i(W_a)}\eta_i = \langle A_{ba}\mathbf{1}, \xi\rangle\langle A_{ab}\mathbf{1}, \eta\rangle$$

Putting everything together,

$$|\langle Z_{ab}\xi, \eta\rangle| \leq |\langle A_{ba}\mathbf{1}, \xi\rangle| \cdot |\langle A_{ab}\mathbf{1}, \eta\rangle|/e(W_a, W_b)$$
$$= O(np/k)\sqrt{|W_a|}\sqrt{|W_b|}/\Omega(p(n/k)^2) = O(1) \ (*)$$

The last equality is due to Lemma A.2 (2), the properties of H (in particular, $e(W_a, W_b) = \Omega(p(n/k)^2)$), and $|W_a| = O(n/k)$ for every a.
We are ready to bound $|\langle Zx, x\rangle|$. For $x \perp K \in \mathbb{R}^m$, we let $x_a \in \mathbb{R}^{|W_a|}$ denote the entries of x corresponding to W_a.

$$|\langle Zx, x\rangle| = \left|\sum_{a,b}\langle Z_{ab}x_a, x_b\rangle\right| \leq$$
$$\sum_{a,b}\|x_a\| \cdot \|x_b\| \cdot \left|\langle Z_{ab}\frac{x_a}{\|x_a\|}, \frac{x_b}{\|x_b\|}\rangle\right|$$
$$\underbrace{\leq}_{(*)} \sum_{a,b}\|x_a\| \cdot \|x_b\| \cdot O(1)$$
$$\leq O(1)\left(\sum_a \|x_a\|\right)^2 \leq O(k) \leq O(\sqrt{np})$$

The last equality is by the choice of p.

We are finally ready to prove property (b) of the vector space K (which was introduced at the beginning of this section), concluding the proof of Lemma A.1. It is readily seen that for every unit vector x, $|\langle Bx, x\rangle| \leq 2$. For a unit vector $x \perp K$,

$$|\langle (B - A - Y')x, x\rangle| \leq |\langle Bx, x\rangle| + |\langle Ax, x\rangle| + |\langle Y'x, x\rangle|$$
$$\underbrace{\leq}_{\text{Lemmas A.2, A.3}} O(\sqrt{np})$$

On the other hand, $d_{min} = \Theta(np/k)$ by the properties of H. Demanding $O(\sqrt{np}) \leq \Theta(np/k)$ (or equivalently, $C_0k^2 \leq np$ for a suitable choice of a constant C_0), Lemma A.1 then follows.

New Results and Open Problems for Deletion Channels

Michael Mitzenmacher
Harvard University

At this point, it seems that most everything is known about the basic channels studied in information theory. For the i.i.d. (independent and identically distributed) binary erasure channel and the i.i.d. binary symmetric error channel, the capacity has long been known, and there are very efficient encoding and decoding schemes that are near capacity.

The situation is very different for the i.i.d. binary deletion channel. With this channel, the sender sends n bits, and each bit is deleted with some fixed probability p. So, for example, the sender might send 10110010, and the receiver obtains 1100. The i.i.d. binary deletion channel is perhaps the most basic channel that incorporates the challenge of synchronization. Surprisingly, even the capacity of the deletion channel remains unknown!

In this talk, I will survey what is known about the deletion channel, focusing on our work on bounds on the capacity and on the many remaining open problems that seem well suited to our community.

No previous background is required.

Joint work with Eleni Drinea.

Partial Fillup and Search Time in LC Tries*

Svante Janson† Wojciech Szpankowski‡

Abstract

Andersson and Nilsson introduced in 1993 a *level-compressed trie* (in short: LC trie) in which a full subtree of a node is compressed to a single node of degree being the size of the subtree. Recent experimental results indicated a "dramatic improvement" when full subtrees are replaced by "partially filled subtrees". In this paper, we provide a theoretical justification of these experimental results showing, among others, a rather moderate improvement of the search time over the original LC tries. For such an analysis, we assume that n strings are generated independently by a binary memoryless source (a generalization to Markov sources is possible) with p denoting the probability of emitting a "1" (and $q = 1 - p$). We first prove that the so called α-fillup $F_n(\alpha)$ (i.e., the largest level in a trie with α fraction of nodes present at this level) is concentrated on two values whp (with high probability); either $F_n(\alpha) = k_n$ or $F_n(\alpha) = k_n + 1$ where $k_n = \log_{\frac{1}{\sqrt{pq}}} n - \frac{|\ln(p/q)|}{2\ln^{3/2}(1/\sqrt{pq})}\Phi^{-1}(\alpha)\sqrt{\ln n} + O(1)$ is an integer and $\Phi(x)$ denotes the normal distribution function. This result directly yields the typical depth (search time) $D_n(\alpha)$ in the α-LC tries with $p \neq 1/2$, namely we show that whp $D_n(\alpha) \approx C_1 \log\log n$ where $C_1 = 1/|\log(1 - h/\log(1/\sqrt{pq}))|$ and $h = -p\log p - q\log q$ is the Shannon entropy rate. This should be compared with recently found typical depth in the original LC tries which is $C_2 \log\log n$ where $C_2 = 1/|\log(1 - h/\log(1/\min\{p, 1-p\}))|$. In conclusion, we observe that α affects only the lower term of the α-fillup level $F_n(\alpha)$, and the search time in α-LC tries is of the same order as in the original LC tries.

1 Introduction

Tries and suffix trees are the most popular data structures on words [7]. A *trie* is a digital tree built over, say n, strings (the reader is referred to [12, 14, 24] for an in depth discussion of digital trees.) A string is stored in an external node of a trie and the path length to such a node is the shortest prefix of the string that is not a prefix of any other strings (cf. Figure 1). Throughout, we assume a binary alphabet. Then each branching node in a trie is a binary node. A special case of a trie structure is a *suffix trie* (tree) which is a trie built over suffixes of a *single* string.

Since 1960 tries were used in many computer science applications such as searching and sorting, dynamic hashing, conflict resolution algorithms, leader election algorithms, IP addresses lookup, coding, polynomial factorization, Lempel-Ziv compression schemes, and molecular biology. For example, in the internet IP addresses lookup problem [15, 22] one needs a fast algorithm that directs an incoming packet with a given IP address to its destination. As a matter of fact, this is the *longest matching prefix* problem, and standard tries are well suited for it. However, the search time is too large. If there are n IP addresses in the database, the search time is $O(\log n)$, and this is not acceptable. In order to improve the search time, Nilsson [15] introduced a novel data structure called the *level compressed trie* or in short LC trie (cf. Figure 1). In the LC trie we replace the root with a node of degree equal to the size of the largest *full subtree* emanating from the root (the depth of such a subtree is called the *fillup level*). This is further carried on recursively throughout the whole trie.

Some recent experimental results reported in [8, 18, 17] indicated a "dramatic improvement" in the search time when full subtrees are replaced by "partially fillup subtrees". In this paper, we provide a theoretical justification of these experimental results by considering α-LC tries in which one replaces a subtree with the last level only α% filled by a node of degree equal to the size of such a subtree (and we continue recursively). In order to understand theoretically the α-LC trie behavior, we study here the so called *α-fillup* level $F_n(\alpha)$ and the *typical depth* or the search time $D_n(\alpha)$. The α-fillup level is the last level in a trie that is α% filled up (e.g., in a binary trie level k is α% filled if it contains $\alpha 2^k$ nodes). The typical depth is the length of a path from the root to a randomly selected external node; thus it represents the typical search time.

*The work was supported by NSF Grants CCR-0208709 and DMS-02-02815, NIH Grant R01 GM068959-01, and NSA Grant MDA 904-03-1-0036

†Dept. Mathematics, Uppsala University, P.O. Box 480, SE-751 06 Uppsala, Sweeden.

‡Department of Computer Science, Purdue University, West Lafayette, IN 47907-2066 U.S.A.

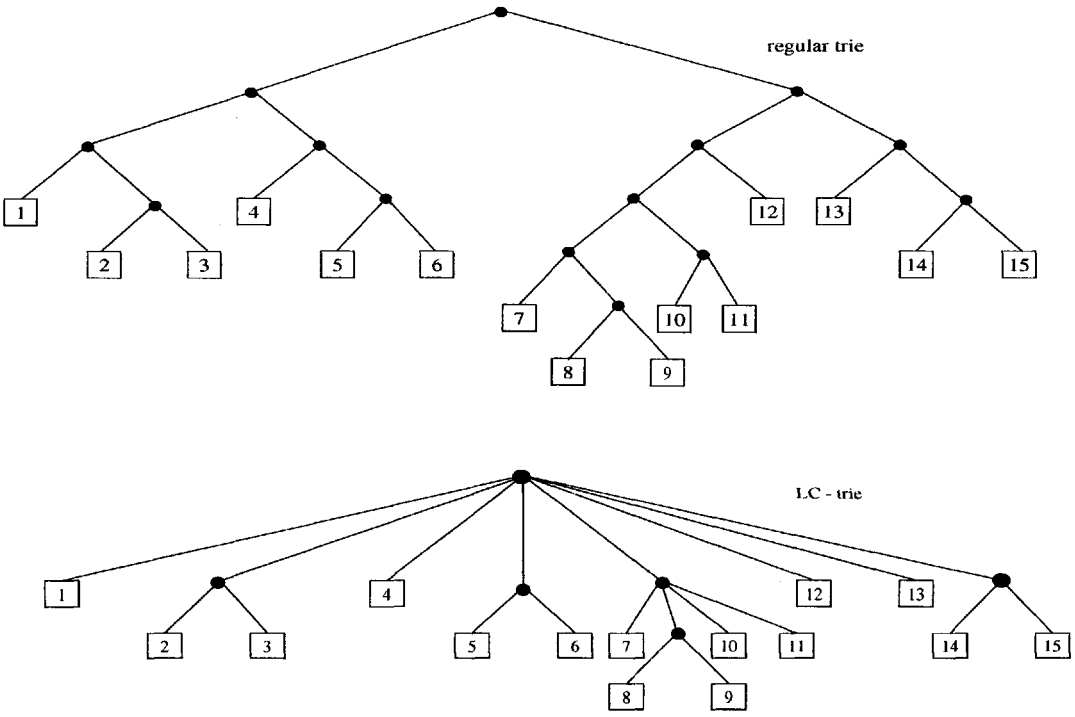


Figure 1: A trie and its associated full LC trie.

In this paper we analyze the α-fillup and the typical depth in an α-LC trie in a probabilistic framework when all strings are generated by a memoryless source with $\mathbb{P}(1) = p$ and $\mathbb{P}(0) = q := 1 - p$. Among other results, we prove that the α-LC trie shows a rather moderate improvement over the original LC tries. We shall quantify this statement below.

Tries were analyzed over the last thirty years for memoryless and Markov sources (cf. [2, 9, 11, 12, 14, 19, 20, 23, 24]). Pittel [19, 20] found the typical value of the fillup level F_n (i.e., $\alpha = 1$) in a trie built over n strings generated by mixing sources; for memoryless sources with high probability (whp)

$$F_n \overset{\mathrm{p}}{\sim} \frac{\log n}{\log(1/p_{\min})} = \frac{\log n}{h_{-\infty}}$$

where $p_{\min} = \min\{p, 1-p\}$ is the smallest probability of generating a symbol and $h_{-\infty} = \log(1/p_{\min})$ is the Rényi entropy of infinite order (cf. [24]). We let $\log := \log_2$.

This was further extended by Devroye [2], and Knessl and Szpankowski [11] who, among other results, proved that the fillup F_n is concentrated on two points k_n and $k_n + 1$, where k_n is an integer

$$\frac{1}{\log p_{\min}^{-1}} (\log n - \log\log\log n) + O(1) \tag{1.1}$$

for $p \neq 1/2$. The depth in regular tries was analyzed by many authors who proved that whp the depth is about $(1/h)\log n$ (where $h = -p\log p - (1-p)\log(1-p)$ is the Shannon entropy rate of the source) and that it is normally distributed when $p \neq 1/2$ [20, 24].

The original LC tries were analyzed by Andersson and Nilsson [1] for unbiased memoryless source and by Devroye [3] for memoryless sources (cf. also [21]). The typical depth (search time) for regular LC tries was only studied recently by Devroye and Szpankowski [4] who proved that for memoryless sources with $p \neq 1/2$

$$D_n \overset{\mathrm{p}}{\sim} \frac{\log\log n}{-\log(1 - h/h_{-\infty})}.$$

In this paper we shall prove some rather surprising results. First of all, for $0 < \alpha < 1$ we show that the α-fillup $F_n(\alpha)$ is whp equal either to k_n or $k_n + 1$ where

$$k_n = \log_{\frac{1}{\sqrt{pq}}} n - \frac{|\ln(p/q)|}{2\ln^{3/2}(1/\sqrt{pq})} \Phi^{-1}(\alpha)\sqrt{\ln n} + O(1). \tag{1.2}$$

As a consequence, we find that if $p \neq 1/2$, the depth $D_n(\alpha)$ of the α-LC is for large n typically about

$$\frac{\log\log n}{-\log\left(1 - h/\log(1/\sqrt{pq})\right)}.$$

The (full) 1-fillup F_n shown in (1.1) should be compared to the α-fillup $F_n(\alpha)$ presented in (1.2). Observe that the leading term of $F_n(\alpha)$ is *not* the same as the leading term of F_n when $p \neq 1/2$. Furthermore, α contributes only to the second term asymptotics. When comparing the typical depths D_n and $D_n(\alpha)$ we conclude that both grow like $\log\log n$ with two constants that do not differ by much. This comparison led us to a statement in the abstract that the improvement of α-LC tries over the regular LC tries is rather moderate. We may add that for relatively slowly growing functions such as $\log\log n$ the constants in front of them do matter (even for large values of n) and perhaps this led the authors of [8, 17, 18] to their statements.

The paper is organized as follows. In the next section we present our main results which are proved in the next two sections. We first consider a poissonized version of the problem for which we establish our findings. Then we show how to depoissonize our results completing our proof.

2 Main Results

Consider tries created by inserting n random strings of 0 and 1. We will always assume that the strings are (potentially) infinite and that the bits in the strings are independent random bits, with $\mathbb{P}(1) = p$ and thus $\mathbb{P}(0) = q := 1 - p$; moreover we assume that different strings are independent.

We let $X_k := \#\{\text{internal nodes filled at level } k\}$ and $\overline{X}_k := X_k/2^k$, i.e. the proportion of nodes filled at level k. Note that X_k may both increase and decrease as k grows, while

$$1 \geq \overline{X}_k \geq \overline{X}_{k+1} \geq 0.$$

Recall that the fillup level of the trie is defined as the last full level, i.e. $\max\{k : \overline{X}_k = 1\}$, (and for example the height is the last level with any nodes at all, i.e. $\max\{k : \overline{X}_k > 0\}$). Similarly, if $0 < \alpha \leq 1$, the α-fillup level is the last level where at least a proportion α of the nodes are filled, i.e. $\max\{k : \overline{X}_k \geq \alpha\}$.

We will in this paper study the α-fillup level for a given α with $0 < \alpha < 1$ and a given p with $0 < p < 1$.

We have the following result, where whp means with probability tending to 1 as $n \to \infty$, and Φ denotes the normal distribution function. Theorem 2.1 is proved in Section 4, after first considering a Poissonized version in Section 3.

THEOREM 2.1. *Let α and p be fixed with $0 < \alpha < 1$ and $0 < p < 1$, and let $F_n(\alpha)$ be the α-fillup level for the trie formed by n random strings as above. Then, for each n*

there is an integer

$$k_n = \log_{\frac{1}{\sqrt{pq}}} n - \frac{|\ln(p/q)|}{2\ln^{3/2}(1/\sqrt{pq})}\Phi^{-1}(\alpha)\sqrt{\ln n} + O(1)$$

such that whp $F_n(\alpha) = k_n$ *or* $k_n + 1$. *Moreover,* $\mathbf{E}\,\overline{X}_{k_n} = \alpha + O(1/\sqrt{\log n})$ *for* $p \neq 1/2$.

Thus the α-fillup $F_n(\alpha)$ is concentrated on at most two values; as in many similar situations (cf. [11]), it is easily seen from the proof that in fact for most n it is concentrated on a single value k_n, but there are transitional regimes, close to the values of n where k_n changes, where $F_n(\alpha)$ takes two values with comparable probabilities.

Note that when $p = 1/2$, the second term on the right hand side disappears, and thus simply $k_n = \log n + O(1)$; in particular, two different values of $\alpha \in (0,1)$ have their corresponding k_n differing by $O(1)$ only. When $p \neq 1/2$, changing α means shifting k_n by $\Theta(\log^{1/2} n)$. By Theorem 2.1, whp $F_n(\alpha)$ is shifted by the same amounts.

To the first order, we thus have the following simple result.

COROLLARY 2.1. *For any fixed* α *and* p *with* $0 < \alpha < 1$ *and* $0 < p < 1$,

$$F_n(\alpha) = \log_{\frac{1}{\sqrt{pq}}} n + O_p(\sqrt{\ln n});$$

in particular, $F_n(\alpha)/\log_{1/\sqrt{pq}} n \xrightarrow{\mathrm{p}} 1$ *as* $n \to \infty$.

Surprisingly enough, the fill up level for $\alpha = 1$ and $\alpha < 1$ are quantitatively different for $p \neq 1/2$. It is well known, as explained in the introduction, that the regular fillup F_n is concentrated on two points around $\log n/\log(1/p_{\min})$, while the partial fillup $F_n(\alpha)$ concentrates around $k_n \sim \log n/\log(1/\sqrt{pq})$. Secondly, the leading term of $F_n(\alpha)$ does not depend on α and the second term is proportional to $\sqrt{\log n}$, while for the regular fillup F_n the second term is of order $\log\log\log n$.

Theorem 2.1 yields several consequences for the behavior of α-LC tries. In particular, it implies the typical behavior of the depth, that is, the search time. Below we formulate our main second result concerning the depth for α-LC tries delaying the formal proof (that will follow the footsteps of [4]) till the final (journal) version of the paper. However, after the statement of theorem we provide a brief heuristics justification.

THEOREM 2.2. *For any fixed* $0 < \alpha < 1$ *and* $p \neq 1/2$ *we have*

$$D_n(\alpha) \overset{\mathrm{p}}{\sim} \frac{\log\log n}{-\log\left(1 - \frac{h}{\log(1/\sqrt{pq})}\right)} \tag{2.3}$$

as $n \to \infty$ *where* $h = -p\log p - (1-p)\log(1-p)$ *is the entropy rate of the source.*

First, let us explain heuristically our estimate for $D_n(\alpha)$. By the Asymptotic Equipartition Property (cf. [24]) at level k_n there are about $n2^{-hk_n}$ strings where h is the entropy. That is, $n2^{-hk_n} \approx n^{1-h/b}$ where for simplicity $b = \log(1/\sqrt{pq})$. In the next level, we shall have about $n^{(1-h/b)^2}$ nodes, and so on. In particular, at level $D_n(\alpha)$ we have approximately

$$n^{(1-h/b)^{D_n(\alpha)}} = O(1)$$

nodes. This leads to our estimate (2.3) of Theorem 2.2.

As a direct consequence of Theorem 2.2 we can numerically quantify experimental results observed by Nilsson and Karlsson who reported in [18] a "dramatic improvement" in the search time of α-LC tries over the regular LC tries. In a regular LC trie the search time is $O(\log\log n)$ with the constant in front of $\log\log n$ being $1/\log(1 - h/\log(1/p_{\min}))^{-1}$ [4]. For α-LC tries this constant decreases to $1/\log(1 - h/\log(1/\sqrt{pq}))^{-1}$. While it is hardly a "dramatic improvement", the fact that we deal with a slowly growing leading term $\log\log n$, may indeed lead to experimentally observed significant changes in the search time.

3 Poissonization

In this section we consider a Poissonized version of the problem, where there are $\mathrm{Po}(\lambda)$ strings inserted in the trie. We let $\tilde{F}_\lambda(\alpha)$ denote the α-fillup level of this trie.

THEOREM 3.1. *Let* α *and* p *be fixed with* $0 < \alpha < 1$ *and* $0 < p < 1$, *and let* $\tilde{F}_\lambda(\alpha)$ *be the* α*-fillup level for the trie formed by* $\mathrm{Po}(\lambda)$ *random strings as above. Then, for each* $\lambda > 0$ *there is an integer*

$$k_\lambda = \log_{\frac{1}{\sqrt{pq}}} \lambda - \frac{|\ln(p/q)|}{2\ln^{3/2}(1/\sqrt{pq})}\Phi^{-1}(\alpha)\sqrt{\ln\lambda} + O(1) \tag{3.4}$$

such that whp (as $\lambda \to \infty$*)* $\tilde{F}_\lambda(\alpha) = k_\lambda$ *or* $k_\lambda + 1$.

We shall prove Theorem 3.1 through a series of lemmas. Observe first that a node at level k can be labelled by a binary string of length k, and that the node is filled if and only if at least two of the inserted strings begin with this label. For $r \in \{0,1\}^k$, let $N_1(r)$ be the number of ones in r, and let $P(r) = p^{N_1(r)}q^{k-N_1(r)}$ be the probability that a random string begins with r. Then, in the Poissonized version, the number of inserted strings beginning with $r \in \{0,1\}^k$ has a Poisson distribution $\mathrm{Po}(\lambda P(r))$, and these numbers are independent for different strings r of the same length.

Consequently,

$$X_k = \sum_{r\in\{0,1\}^k} I_r \tag{3.5}$$

where I_r are independent indicators with

$$\mathbb{P}(I_r = 1) = \mathbb{P}(\mathrm{Po}(\lambda P(r)) \ge 2) = 1-(1+\lambda P(r))e^{-\lambda P(r)}. \tag{3.6}$$

Hence,

$$\mathbf{Var}\,(X_k) = \sum_{r\in\{0,1\}^k} P(I_r = 1)\big(1 - P(I_r = 1)\big) < 2^k$$

so $\mathbf{Var}\,(\overline{X}_k) < 2^{-k}$ and, by Chebyshev's inequality,

$$\mathbb{P}(|\overline{X}_k - \mathbf{E}\,\overline{X}_k| > 2^{-k/3}) \to 0. \tag{3.7}$$

Consequently, $\overline{X}_k$ is sharply concentrated, and it is enough to study its expectation. (It is straightforward to calculate $\mathbf{Var}\,(X_k)$ more precisely, and to obtain a normal limit theorem for X_k, but we do not need that.)

Assume first $p > 1/2$.

LEMMA 3.1. *If $p > 1/2$ and*

$$k = \log_{\frac{1}{\sqrt{pq}}} \lambda - \frac{\ln(p/q)}{2\ln^{3/2}(1/\sqrt{pq})}\Phi^{-1}(\alpha)\sqrt{\ln\lambda} + O(1), \tag{3.8}$$

then $\mathbf{E}\,\overline{X}_k = \alpha + O(k^{-1/2})$.

Proof. Let $\rho = p/q > 1$ and define γ by $\lambda p^\gamma q^{k-\gamma} = 1$, i.e.,

$$\rho^\gamma = \left(\frac{p}{q}\right)^\gamma = \lambda^{-1}q^{-k},$$

which leads to

$$\gamma = \frac{k\ln(1/q) - \ln\lambda}{\ln(p/q)}. \tag{3.9}$$

Let $\mu_j = \lambda p^j q^{k-j} = \rho^{j-\gamma}$. By (3.5) and (3.6),

$$\mathbf{E}\,\overline{X}_k = 2^{-k}\sum_{j=0}^{k}\binom{k}{j}\mathbb{P}(\mathrm{Po}(\mu_j) \ge 2). \tag{3.10}$$

If $j < \gamma$, then $\mu_j < 1$ and

$$\mathbb{P}(\mathrm{Po}(\mu_j) \ge 2) < \mu_j^2 < \mu_j.$$

If $j \ge \gamma$, then $\mu_j \ge 1$ and

$$1 - \mathbb{P}(\mathrm{Po}(\mu_j) \ge 2) = (1+\mu_j)e^{-\mu_j} \le 2\mu_j e^{-\mu_j} < 4\mu_j^{-1}.$$

Hence (3.10) yields, using $\binom{k}{j} \le \binom{k}{\lfloor k/2\rfloor} = O(2^k k^{-1/2})$,

$$\begin{aligned}\mathbf{E}\,\overline{X}_k &= 2^{-k}\sum_{j<\gamma}\binom{k}{j}O(\mu_j) + 2^{-k}\sum_{j\ge\gamma}\binom{k}{j}(1-O(\mu_j^{-1})) \\ &= 2^{-k}\sum_{j\ge\gamma}\binom{k}{j} + 2^{-k}\sum_{j=0}^{k}\binom{k}{j}O(\rho^{-|j-\gamma|}) \\ &= \mathbb{P}\big(\mathrm{Bi}(k,1/2) \ge \gamma\big) + O(k^{-1/2}).\end{aligned} \tag{3.11}$$

By the Berry–Esseen theorem [6, Theorem XVI.5.1],

$$\mathbb{P}(\mathrm{Bi}(k,1/2) \ge \gamma) = 1 - \Phi\Big(\frac{\gamma - k/2}{\sqrt{k/4}}\Big) + O(k^{-1/2}). \tag{3.12}$$

By (3.9) and the assumption (3.8),

$$\begin{aligned}\gamma - \frac{k}{2} &= \frac{1}{\ln(p/q)}\left(k\ln\frac{1}{q} - \ln\lambda - \frac{k}{2}\ln\frac{p}{q}\right) \\ &= \frac{1}{\ln(p/q)}\left(k\ln\frac{1}{\sqrt{pq}} - \ln\lambda\right) \\ &= \frac{\ln(1/\sqrt{pq})}{\ln(p/q)}\left(k - \log_{1/\sqrt{pq}}\lambda\right) \\ &= -\tfrac{1}{2}(\ln(1/\sqrt{pq}))^{-1/2}\Phi^{-1}(\alpha)\sqrt{\ln\lambda} + O(1) \\ &= -\tfrac{1}{2}\Phi^{-1}(\alpha)k^{1/2} + O(1).\end{aligned} \tag{3.13}$$

This finally implies

$$1 - \Phi\Big(\frac{\gamma - k/2}{\sqrt{k/4}}\Big) = 1 - \Phi(-\Phi^{-1}(\alpha)) + O(k^{-1/2}) = \\ = \alpha + O(k^{-1/2}),$$

and the lemma follows by (3.11) and (3.12).

LEMMA 3.2. *Fix $p > 1/2$. For every $A > 0$, there exists $c > 0$ such that if $|k - \log_{1/\sqrt{pq}}\lambda| \le Ak^{1/2}$, then $\mathbf{E}\,\overline{X}_k - \mathbf{E}\,\overline{X}_{k+1} > ck^{-1/2}$.*

Proof. A string $r \in \{0,1\}^k$ has two extensions $r0$ and $r1$ in $\{0,1\}^{k+1}$. Clearly, $I_{r0}, I_{r1} \le I_r$, and if there are exactly 2 (or 3) of the inserted strings beginning with r, then $I_{r0} + I_{r1} \le 1 < 2I_r$. Hence

$$\begin{aligned}\mathbf{E}\,(2X_k - X_{k+1}) &= \sum_{r\in\{0,1\}^k}\mathbf{E}\,(2I_r - I_{r0} - I_{r1}) \\ &\ge \sum_{r\in\{0,1\}^k}\mathbb{P}\big(\mathrm{Po}(\lambda P(r)) = 2\big).\end{aligned} \tag{3.14}$$

Let ρ and γ be as in the proof of Lemma 3.1, and let $j = \lceil\gamma\rceil$. Then $\mu_j = \rho^{j-\gamma} \in [1,\rho]$ and thus

$\mathbb{P}(\mathrm{Po}(\mu_j) = 2) \geq \frac{1}{2}e^{-\rho}$. Moreover, by (3.13) and the assumption,

$$|j - k/2| \leq \frac{\ln(1/\sqrt{pq})}{\ln(p/q)} Ak^{1/2} + 1 = O(k^{1/2}).$$

Thus, if k is large enough, we have by the standard normal approximation of the binomial probabilities (which follows easily from Stirling's formula, as found already by de Moivre [5])

$$2^{-k}\binom{k}{j} = \frac{1+o(1)}{\sqrt{2\pi k/4}} e^{-2(j-k/2)^2/k} \geq c_1 k^{-1/2}$$

for some $c_1 > 0$. Hence, by (3.14),

$$\begin{aligned} \mathbf{E}\,\overline{X}_k - \mathbf{E}\,\overline{X}_{k+1} &= 2^{-k-1}\mathbf{E}\,(2X_k - X_{k+1}) \\ &\geq 2^{-k-1}\binom{k}{j}\mathbb{P}(\mathrm{Po}(\mu_j) = 2) \\ &\geq \frac{c_1 e^{-\rho}}{4} k^{-1/2} \end{aligned}$$

as needed.

Now assume $p > 1/2$. Starting with any k as in (3.8), we can by Lemmas 3.1 and 3.2 shift k up or down $O(1)$ steps and find k_λ as in (3.4) such that, for a suitable $c > 0$, $\mathbf{E}\,\overline{X}_{k_\lambda} \geq \alpha + \frac{1}{2}ck_\lambda^{-1/2} > \mathbf{E}\,\overline{X}_{k_\lambda+1}$ and $\mathbf{E}\,\overline{X}_{k_\lambda+2} \leq \mathbf{E}\,\overline{X}_{k_\lambda+1} - ck_\lambda^{-1/2} < \alpha - \frac{1}{2}ck_\lambda^{-1/2}$. It follows by (3.7) that whp $\overline{X}_{k_\lambda} \geq \alpha$ and $\overline{X}_{k_\lambda+2} < \alpha$, and hence $\tilde{F}_\lambda(\alpha) = k_\lambda$ or $k_\lambda + 1$.

This proves Theorem 3.1 in the case $p > 1/2$. The case $p < 1/2$ follows by symmetry, interchanging p and q.

In the remaining case $p = 1/2$, all $P(r) = 2^{-k}$ are equal. Thus, by (3.5) and (3.6),

$$\mathbf{E}\,\overline{X}_k = \mathbb{P}(\mathrm{Po}(\lambda 2^{-k}) \geq 2). \tag{3.15}$$

Given $\alpha \in (0,1)$, there is a $\mu > 0$ such that $\mathbb{P}(\mathrm{Po}(\mu) \geq 2) = \alpha$. We take $k_\lambda = \lfloor \log(\lambda/\mu) - 1/2 \rfloor$. Then, $\lambda 2^{-k_\lambda} \geq 2^{1/2}\mu$ and thus $\mathbf{E}\,\overline{X}_{k_\lambda} \geq \alpha_+$ for some $\alpha_+ > \alpha$. Similarly, $\mathbf{E}\,\overline{X}_{k_\lambda+2} \leq \alpha_-$ for some $\alpha_- < \alpha$, and the result follows in this case too.

4 Depoissonization

To complete the proof of Theorem 2.1 we must depoissonize the results obtained in Theorem 3.1, which we do in this section.

Proof. [Proof of Theorem 2.1] Given an integer n, let k_n be as in the proof of Theorem 3.1 with $\lambda = n$, and let $\lambda_\pm = n \pm n^{2/3}$. Then $\mathbb{P}(\mathrm{Po}(\lambda_-) \leq n)) \to 1$ and $\mathbb{P}(\mathrm{Po}(\lambda_+) \geq n)) \to 1$ as $n \to \infty$. By monotonicity, we thus have whp $\tilde{F}_{\lambda_-}(\alpha) \leq F_n(\alpha) \leq \tilde{F}_{\lambda_+}(\alpha)$, and by Theorem 3.1 it remains only to show that we can take $k_{\lambda_-} = k_{\lambda_+} = k_n$.

Let us now write $X_k(\lambda)$ and $\overline{X}_k(\lambda)$, since we are working with several λ.

LEMMA 4.1. *Assume $p \neq 1/2$. Then, for every k,*

$$\frac{d}{d\lambda}\mathbf{E}\,\overline{X}_k(\lambda) = O(\lambda^{-1}k^{-1/2}).$$

Proof. We have

$$\frac{d}{d\mu}\mathbb{P}(\mathrm{Po}(\mu) \geq 2) = \frac{d}{d\mu}((1 - (1+\mu)e^{-\mu}) = \mu e^{-\mu}$$

and thus, by (3.10) and the argument in (3.11),

$$\begin{aligned} \frac{d}{d\lambda}\mathbf{E}\,\overline{X}_k(\lambda) &= 2^{-k}\sum_{j=0}^{k}\binom{k}{j}\mu_j e^{-\mu_j}\frac{d\mu_j}{d\lambda} \\ &= \lambda^{-1}2^{-k}\sum_{j=0}^{k}\binom{k}{j}\mu_j^2 e^{-\mu_j} \\ &= O\Big(\lambda^{-1}\sum_{j=0}^{k}2^{-k}\binom{k}{j}\min(\mu_j, \mu_j^{-1})\Big) \\ &= O(\lambda^{-1}k^{-1/2}) \end{aligned}$$

which completes the proof.

By Lemma 4.1, $|\mathbf{E}\,\overline{X}_k(\lambda_\pm) - \mathbf{E}\,\overline{X}_k(n)| = O(n^{-1/3}k^{-1/2}) = o(k^{-1/2})$. Hence, by the proof of Theorem 3.1, for large n, $\mathbf{E}\,\overline{X}_{k_n}(\lambda_\pm) \geq \alpha + \frac{1}{3}ck_n^{-1/2}$ and $\mathbf{E}\,\overline{X}_{k_n+2}(\lambda_\pm) < \alpha - \frac{1}{3}ck_n^{-1/2}$, and thus whp $\tilde{F}_{\lambda_\pm}(\alpha) = k_n$ or $k_n + 1$. Moreover, the estimate $\mathbf{E}\,\overline{X}_{k_n} = \alpha + O(1/\sqrt{\log n})$ follows easily from the similar estimate for the Poisson version in Lemma 3.1; we omit the details. This completes the proof of Theorem 2.1 for $p > 1/2$. The case $p < 1/2$ is again the same by symmetry. The proof when $p = 1/2$ is similar, now using (3.15).

References

[1] A. Andersson and S. Nilsson, Improved behavior of tries by adaptive branching, *Information Processing Letters*, **46**, 295–300, 1993.

[2] L. Devroye, A Note on the Probabilistic Analysis of Patricia Tries, *Random Structures and Algorithms*, **3**, 203–214, 1992.

[3] L. Devroye, An analysis of random LC tries, *Random Structures and Algorithms*, **19**, 359–375, 2001.

[4] L. Devroye and W. Szpankowski, Probabilistic behavior of asymmetric level compressed tries, *Random Structures & Algorithms*, **26**, 2005

[5] A. de Moivre, *The Doctrine of Chances*, 2nd ed., H. Woodfall, London, 1738

[6] W. Feller, *An Introduction to Probability Theory and its Applications,* Vol. II. 2nd ed., Wiley, New York, 1971.

[7] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, 1997.

[8] P. Iivonen, S. Nilsson and M. Tikkanen, An experimental study of compression methods for functional tries, in: *Workshop on Algorithmic Aspects of Advanced Programming Languages (WAAAPL'99)*, 1999.

[9] P. Jacquet and W. Szpankowski, Analysis of Digital Tries with Markovian Dependency, *IEEE Trans. Information Theory*, **37**, 1470–1475, 1991.

[10] P. Jacquet and W. Szpankowski, Analytical depoissonization and its applications, *Theoretical Computer Science*, **201**, 1–62, 1998

[11] C. Knessl and W. Szpankowski, On the number of full levels in tries, *Random Structures and Algorithms*, **25**, 247–276, 2004.

[12] D. E. Knuth, *Fundamental Algorithms*, 3rd ed, Addison-Wesley, Reading, Massachusetts, 1997.

[13] D. E. Knuth, *Selected Papers on Analysis of Algorithms*, CSLI, Stanford, 2000.

[14] H. Mahmoud, *Evolution of Random Search Trees*, John Wiley & Sons, New York, 1992.

[15] S. Nilsson, *Radix Sorting & Searching*, PhD Thesis, Lund University, 1996.

[16] S. Nilsson and G. Karlsson, Fast address look-up for Internet routers, in: *Proceedings IFIP 4th International Conference on Broadband Communications*, 11–22, 1998.

[17] S. Nilsson and G. Karlsson, IP-address lookup using LC-tries, *IEEE Journal on Selected Areas in Communications*, **17**(6), 1083–1092, 1999.

[18] S. Nilsson and M. Tikkanen, An experimental study of compression methods for dynamic tries, *Algorithmica*, **33**(1), 19–33, 2002.

[19] B. Pittel, Asymptotic Growth of a Class of Random Trees, *Annals of Probability*, **18**, 414–427, 1985.

[20] B. Pittel, Paths in a Random Digital Tree: Limiting Distributions, *Adv. in Applied Probability*, **18**, 139–155, 1986.

[21] Y. Reznik, Some Results on Tries with Adaptive Branching, *Theoretical Computer Science*, **289**, 1009–1026, 2002.

[22] V. Srinivasan and G. Varghese, Fast Address Lookups using Controlled Prefix Expansions, *ACM SIGMETRICS'98*, 1998.

[23] W. Szpankowski, On the Height of Digital Trees and Related Problems, *Algorithmica*, **6**, 256–277, 1991.

[24] W. Szpankowski *Average Case Analysis of Algorithms on Sequences*, John Wiley, New York, 2001.

Distinct Values Estimators for Power Law Distributions

Rajeev Motwani* Sergei Vassilvitskii†

Abstract

The number of distinct values in a relation is an important statistic for database query optimization. As databases have grown in size, scalability of distinct values estimators has become extremely important, since a naïve linear scan through the data is no longer feasible. An approach that scales very well involves taking a sample of the data, and performing the estimate on the sample. Unfortunately, it has been shown that obtaining estimators with guaranteed small error bounds requires an extremely large sample size in the worst case. On the other hand, it is typically the case that the data is not worst-case, but follows some form of a Power Law or Zipfian distribution. We exploit data distribution assumptions to devise distinct-values estimators with analytic error guarantees for Zipfian distributions. Our estimators are the first to have the required number of samples depend only on the number of distinct values present, D, and not the database size, n. This allows the estimators to scale well with the size of the database, particularly if the growth is due to multiple copies of the data. In addition to theoretical analysis, we also provide experimental evidence of the effectiveness of our estimators by benchmarking their performance against previously best known heuristic and analytic estimators on both synthetic and real-world datasets.

1 Introduction

The number of distinct values of an attribute in a relation is one of the critical statistics necessary for effective query optimization. It is well-established [9] that a bad estimate to the number of distinct values can slow down the query execution time by several orders of magnitude. Unfortunately, as the amount of data stored in a database increases, this vital statistic becomes increasingly difficult to estimate quickly with reasonable accuracy. While the exact number of distinct values in a column can be determined by a full scan of the table, query optimizers would like to obtain a (low-error) estimate with significantly lower effort. Even if one is willing to perform a full scan, determining the exact number of distinct values requires significant memory overhead.

Several approaches have been considered in the literature to deal with this issue. Recently, much of the work has focused on streaming models, or algorithms which are allowed to take only a single pass over the data [1, 5, 6]. The challenge for these algorithms lies in minimizing the space used, since the naïve schemes run out of memory long before a single scan is complete. Another natural approach is to take a small random sample from the large dataset (often on the order of 1-10%) and then to estimate the number of distinct values from the sample. This problem has a rich history in statistics [2, 8, 19], but the statistical methods are essentially heuristic and in any case do not perform well in the context of databases [12, 13]. There has been some recent work in database literature [3, 7, 9, 10] on trying to devise good distinct-values estimators for random samples; but again, these are mostly based on heuristics and are not supported by analytic error guarantees.

An explanation for the apparent difficulty of distinct-values estimation was provided in the powerful negative result of Charikar, Chaudhuri, Motwani, and Narasayya [3]. They demonstrate two data distribution scenarios where the numbers of distinct values differ dramatically, yet a large number of random samples is required to distinguish between the two scenarios. For example, to guarantee that an estimate has less than 10% error with high probability, requires sampling almost the entire table. While this negative result explains the difficulty of obtaining estimators with good analytic error guarantees, the worst case scenarios rarely occur in practice. This leaves open the possibility of exploiting our knowledge of real-world data distributions to obtain estimators that are efficient and scalable, have analytic error guarantees, and perform well in practice. Indeed, in this paper we show that such positive results are possible once we make some assumptions about the underlying data distribution, thereby allowing us to circumvent the seemingly crippling negative result of Charikar et al. [3].

It has been observed for over a half-century that many large datasets follow a Power Law (also known as Zipfian) distribution; for example, the distribution

*Stanford University. Supported in part by NSF Grants EIA-0137761 and ITR-0331640, and grants from Media-X and SNRC.

†Stanford University. Supported in part by NSF Grants EIA-0137761 and ITR-0331640, and grants from Media-X and SNRC.

of words in a natural language [20] or the distribution of the (out-)degrees in the web graph [14]. We refer the reader to the book by Knuth [15] and the survey article by Mitzenmacher [17] for further examples and an in-depth discussion. The underlying reasons for the ubiquity of this class of data distributions have been a subject of debate ever since the original paper by Zipf [20], but as Mitzenmacher points out one thing is clear: "Power law distributions are now pervasive in computer science."

1.1 Our Results In this work, we assume that the n data items in the column of interest follow a Zipfian distribution (with some skew parameter θ) on some number of distinct elements D. Our estimators work on a random sample of the data from the column. We assume that the value of θ is known ahead of time. In our experimental tests we show that the value of θ can be easily estimated on real world datasets using linear regression techniques. Of course, the value of D is assumed to be unknown, since that is precisely the quantity that we seek to estimate.

A key feature of the algorithms that we propose is the *independence* of their running time from the database size. These are the first algorithms where the number of samples and the running time are a function of solely the number of *distinct* elements present and not of the database size itself. This property allows our estimators to scale extremely well. In particular, the running time of the estimators remains the same if the database contains multiple copies of the same data.

We propose two algorithms for computing the number of distinct values. The first algorithm samples adaptively until a stopping condition is met. We prove that for a large family of distributions the algorithm returns D, the *exact* number of distinct values with high probability; and requires no more than $O(\log D/p_D)$ samples, where p_D is the probability of selecting the least likely element. Observe that if the underlying distribution is uniform, coupon-collector arguments provide a matching lower bound for the required number of samples.

The setting for the second algorithm is slightly different. In some applications we are not able to adaptively sample, but rather are presented with a small fraction of the database and are asked to provide the best possible estimate. In this setting the second algorithm returns $\widehat{D}$ a $(1+\epsilon)$ approximation to D with high probability after examining only this small number of random samples. In particular we analyze our algorithm for Zipfian distributions where the estimator is correct with probability $1 - \exp(-\Omega(D)/\epsilon^2)$ after examining roughly $1/p_D$ samples.

We demonstrate via experiments that our estimators not only have theoretical error guarantees, but also outperform previously-known estimators on synthetic and real world inputs.

The rest of this paper is organized as follows. We begin in Sections 2 and 3 by formally defining the problem and presenting the goals that an estimator should strive to achieve. In Section 4 we present an algorithm for computing the exact number of distinct values in a database with high probability. Section 5 introduces an estimator that returns an ϵ-error approximation to the true number of distinct values. Finally, in Section 6 we present experimental results comparing the performance of our estimator best known heuristic and guaranteed error estimators, on both synthetic and real world data.

2 Preliminaries

We assume the following set-up throughout the paper. Let $\mathcal{F} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots\}$ be a family of probability distributions, where $\mathcal{P}_j$ is a distribution on j elements. Let R be a relation on n rows, and assume that the elements in R are distributed along some $\mathcal{P}_{j^*} \in \mathcal{F}$. Our goal is to determine j^*, the number of distinct elements present in R.

To simplify notation, given a distribution $\mathcal{P}$ over a set X and an element $x \in X$, denote by $Pr_i(\mathcal{P})$ the probability of element $i \in X$ in the distribution $\mathcal{P}$.

As a special case we consider the family of Zipfian or Power Law distributions parametrized by their skew, θ. $\mathcal{Z}_\theta = \{Z_{1,\theta}, Z_{2,\theta}, \ldots\}$. where $Z_{D,\theta}$ is the Zipfian distribution of parameter θ on D elements defined as follows. Rank the elements 1 through D in decreasing order of probability mass. Then the probability of selecting the i^{th} element is

$$Pr_i(Z_{D,\theta}) = \frac{1}{i^\theta N_{D,\theta}},$$

$$\text{where } N_{D,\theta} = \sum_{i=1}^{D} \frac{1}{i^\theta} \text{ is a normalizing constant}$$

Observe that when $\theta = 0$, the Zipfian distribution is simply the uniform distribution and $N_{D,0} = D$. The skew in the distribution increases with θ. In real-life applications θ is typically less than 2.

We assume that the value of θ is known to our algorithm; in practice a good estimate for the value of θ can be obtained as a part of the sampling process as discussed in Section 6.1.

Our estimation algorithm will deliver an estimate $\widehat{D}$ of the number of distinct values in the column. To evaluate the performance of the estimators, we will use *ratio error*, which is the multiplicative error of $\widehat{D}$ with

respect to D. Formally, we define ratio error as

$$\max\left(\frac{\hat{D}}{D}, \frac{D}{\hat{D}}\right)$$

Under this definition, the error is always at least 1, and no distinction is made between underestimates and overestimates of the number of distinct values.

The following notation will be useful: After some r samples, let $f_{in}(r)$ be the number of distinct values that appeared in the sample and $f_{out}(r, D) = D - f_{in}$, the number of distinct values that were not part of the sample.

3 Estimators

Our goal is to obtain distinct-values estimators with the following desired properties:

Few Samples: The number of samples required for good performance by the estimator should be small.

Error Guarantees: The estimator should be backed by analytical error guarantees.

Scalability: The estimator should scale well as the database size n increases. This implies that the number of samples should grow sublinearly with (or ideally be independent of) the database size.

As mentioned earlier, the vast majority of estimators that operate on a random sample of the data (as opposed to those which perform a full scan of the data) do not provide any analytical guarantees for their performance. The exception is the GEE (Guaranteed-Error Estimator) estimator developed by Charikar et al. [3].

THEOREM 3.1. ([3]) *Using a random sample of size r from a table of size n, the expected ratio error for GEE is $O(\sqrt{n/r})$.*

We note that the result above is the best possible due to a matching lower bound described by the authors. Observe that the bound is quite weak — even if we allow a sample of 10%, the expected ratio error bound can be as high as $\sqrt{10} \approx 3.2$. Further, the GEE estimator does not scale well — to maintain the same error, the sample size needs to increase linearly with the size of the database.

Once we assume that the input distribution follows a Zipfian distribution with unknown parameter D, we can develop estimators which greatly improve upon the GEE estimator presented above. We focus first on determining exactly the number of distinct values in the database, and then we relax this requirement to devise estimators which may return a $(1+\epsilon)$-error approximation to the number of distinct values.

4 Exact Algorithm

We first seek to devise an algorithm which will return $\hat{D} = D$, but is allowed to fail with some small probability δ. Note that without the knowledge of the data distribution, the situation is grim — in the worst case we would need to sample a large fraction of the database to obtain the value of D with bounded error probability.

We begin by defining a notion of *c-regular* families of distributions.

DEFINITION 4.1. *A distribution family $\mathcal{F} = \{P_1, P_2, \ldots\}$ is called* c-regular *if the following two conditions hold:*

- **Monotonicity:** For any $i, 1 \leq j \leq i$: $Pr_i(P_j) \geq Pr_i(P_{j+1})$.
- **c-Regularity:** For any i: $Pr_i(P_i) \leq c \cdot Pr_{i+1}(P_{i+1})$.

The *monotonicty* condition ensures that the probability of an individual item i in the support, decreases as the overall support of the distribution increases. The *c-regularity* condition bounds the decrease in mass of the least weighted element.

Many common distribution families are c-regular for small values of c. For example, the family of uniform distributions is 2-regular. The family of Zipfian distributions of parameter θ is 5-regular for $\theta \leq 2$

To simplify notation, for a multiset S, let $Distinct(S)$ be the number of distinct values appearing in S.

The *Exact Count* algorithm presented below will continue to sample until a particular stopping condition is met, at which point the sample contains all of the distinct values with high probability.

Algorithm 1 Exact Count Algorithm

Let $Stop(t) = \frac{6\ln(t+1)}{Pr_{t+1}P_{t+1}}$
Let $S \subseteq R$ denote the current sample
Draw a sample of size $Stop(3)$
while $Stop(Distinct(S)) > |S|$ **do**
  Increase S until $|S|$ grows by a factor of $c\log_3 4$
end while
Output $\hat{D} = Distinct(S)$

4.1 Analysis

We will show that the above algorithm returns $\hat{D} = D$ with probability at least $1/2$.

LEMMA 4.1. *Let S be a sample of size at lest $Stop(3)$ drawn uniformly at random from R. Let t be such that $Stop(t+1) \geq |S| > Stop(t)$. If $t < D$ then $Pr[Distinct(S) < t+1] \leq (t+1)^{-2}$.*

Before proving the lemma, let us interpret its meaning. Observe that the algorithm halts when $Distinct(S) \leq t$. This lemma bounds the probability of early halting (halting when $Distinct(S) \neq D$).

Proof. Consider the elements $1, 2, ..., D$. For the sake of the proof suppose we can split them up into $t+2$ groups with the following properties:

1. each element appears in exactly one group.
2. For groups $j \in \{1, ..., t+1\}$:

$$Pr_j(\mathcal{P}_{t+1}) \geq \sum_{i \in group\ j} Pr_i(\mathcal{P}_D) \geq \frac{Pr_j(\mathcal{P}_{t+1})}{2}$$

Let $\mathcal{E}$ be the event that $Distinct(S) < t+1$. For $\mathcal{E}$ to occur, all of the elements from at least one of the first $t+1$ groups can not present in S. Let $\mathcal{E}_j$ be the event that no element from group j appears in the sample.

$$\begin{aligned} Pr[\mathcal{E}_j] &\leq (1 - Pr_j(\mathcal{P}_{t+1})/2)^{|S|} \\ &\leq \exp(-\frac{Pr_j(\mathcal{P}_{t+1})}{2Pr_{t+1}(\mathcal{P}_{t+1})} \cdot (6\ln(t+1))) \\ &\leq \frac{1}{(t+1)^3} \end{aligned}$$

Where the last inequality follows since $Pr_j(\mathcal{P}_{t+1}) \geq Pr_{t+1}(\mathcal{P}_{t+1})$. For event $\mathcal{E}$ to occur one of the events $\mathcal{E}_1, \ldots, \mathcal{E}_{t+1}$ must occur. We can use the union bound to limit $Pr[\mathcal{E}]$. In particular $Pr[\mathcal{E}] \leq \sum_{i=1}^{t+1}(t+1)^{-3} = (t+1)^{-2}$.

It remains to show how the elements are broken up into the aforementioned groups.

We can achieve this division using the following simple algorithm. Consider the elements in order of decreasing probability: $Pr_1(\mathcal{P}_D), Pr_2(\mathcal{P}_D), \ldots, Pr_D(\mathcal{P}_D)$. Put the first element into the first group. The monotonicity property ensures that it will fit. Continue with elements of weight $Pr_2(\mathcal{P}_D), Pr_3(\mathcal{P}_D)$, etc. Once an element no longer fits into the first group, begin filling the next group. Again, by the monotonicity property, the first element will fit in the group. After filling the first $t+1$ groups, put the remaining elements into group $t+2$.

It is clear that each element will be present in exactly one group. Further, each group $\in \{1, \ldots, t+1\}$ is at least half full, since it contains at least one element, and the elements are considered in order of decreasing weight.

To conclude the analysis we need to bound the total number of times that the event $\mathcal{E}$ can occur. We do this by showing that the value of t increases every time we evaluate the stopping condition.

LEMMA 4.2. *For $|S| \geq Stop(2)$, $Stop^{-1}(c\log_3 4 \cdot |S|) - Stop^{-1}(|S|) \geq 1$, where $Stop^{-1}(U) = \min_u\{u : Stop(u) > U\}$.*

Proof. We will prove this by bounding the ratio $\frac{Stop(t+1)}{Stop(t)}$ for $t \geq 2$.

$$\begin{aligned} \frac{Stop(t+1)}{Stop(t)} &= \frac{\ln(t+2)}{\ln(t+1)} \cdot \frac{P_{t+1}(t+1)}{P_{t+2}(t+2)} \\ &\leq \log_3 4 \cdot c \end{aligned}$$

Where the second inequality follows from the c-regularity condition.

THEOREM 4.1. *The Exact-Count algorithm terminates successfully with probability at least 1/2. Moreover, the number of samples necessary is $O(\log D / Pr_D(\mathcal{P}_D))$.*

Proof. The algorithm can potentially fail only when the stopping condition is evaluated. Let E_i be the event that the algorithm halts on the i^{th} evaluation, but the sample does not yet contain all of the distinct values. Lemma 4.1 implies that at the point of i^{th} evaluation, $|S| \geq Stop(i+1)$, and so $Pr[E_i] \leq (i+1)^{-2}$. By the union bound, the event that algorithm fails is bounded by $\sum_{i=1}^{\infty} \frac{1}{(i+1)^2} < 1/2$.

COROLLARY 4.1. *For Zipfian distributions with parameter θ the estimator above requires $O(D^{\theta} N_{D,\theta} \log D)$ samples.*

The bound we present above is tight up to very small factors. Consider two distributions: $\mathcal{P}$ which is a uniform distribution on m values and $\mathcal{Q}$, a uniform distribution on $m+1$ values. Let S be a sample from one of the distributions, such that $|S| = o(m\frac{\log m}{\log\log m})$ then by standard coupon collector arguments S contains less than m distinct values with high probability. It is easy to see that the relative frequency of the elements in S is the same whether the elements were drawn from $\mathcal{P}$ or from $\mathcal{Q}$, and so $\mathcal{P}$ and $\mathcal{Q}$ are indistinguishable.

THEOREM 4.2. *There exist c-regular families of distributions $\mathcal{F}$ on which any algorithm requires $\Omega(D\frac{\log D}{\log\log D})$ samples.*

5 Approximate Algorithm

Although the above algorithm provides us with analytical guarantees about the number of samples in the distribution, the required number of samples can be high. In some applications there exists a hard limit on the running time of the estimator, thus we look for a procedure

which returns the best possible estimate on the number of distinct values given a sample from the database.

In this section we present an estimator which returns $\widehat{D}$ such that with probability at least $(1-\delta)$ the ratio error is bounded by $(1+\epsilon)$. We provide the analysis below for the case of Zipfian distributions parametrized by their skew, θ.

5.1 Zipfian Distributions The algorithm we consider is similar to the maximum likelihood estimator. Recall that $f_{in}(r)$ is the number of distinct elements in a random sample of size r. Let $f^*_{in}(r, D, \theta)$ be the expected number of distinct elements in a random sample of size r coming from a Zipfian distribution of parameter θ on D distinct values.

Observe that $f^*_{in}(r, D, \theta)$ can be computed when D is known. Our estimator returns

$$\widehat{D} \text{ such that } f^*_{in}(r, \widehat{D}, \theta) = f_{in}$$

In other words, our guess for the number of distinct elements would (in expectation) have us see as many distinct values as we did.

5.2 Analysis The analysis of the simple algorithm above proceeds in two parts. First, we show that with high probability the observed value of f_{in} does not deviate by more than a $(1+\epsilon)$ factor from its expected value, $f^*_{in}(r, D, \theta)$. We then show that if this is the case, then $\widehat{D}$ does not deviate from D by a factor of more than $(1+\epsilon)$ with constant probability, provided that our sample size is large enough.

LEMMA 5.1. *In a sample of size r,*

$$Pr[|f_{in} - f^*_{in}(r, D, \theta)| \geq \epsilon f^*_{in}(r, D, \theta)] \leq$$

$$2\exp(-\frac{\epsilon^2 f^*_{in}(r, D, \theta)^2}{2r})$$

Proof. Let X_i be the number of distinct elements that appear in the sample after i samples. Let $Y_i = E[X_r | X_1, \ldots, X_i]$ be the expected number of distinct elements in the sample after r samples given the results of the first i samples. Note that $Y_r = f_{in}$, the number of distinct values observed, and $Y_0 = f^*_{in}(D, \theta)$, the expected number of unique values observed. By definition the Y_is form a Doob martingale. Now consider their successive difference, $|Y_i - Y_{i-1}|$. Since the only information revealed is the location of the i^{th} sample, $|Y_i - Y_{i-1}| \leq 1$.

We can now invoke Azuma's inequality (see Section 4.4 of Motwani and Raghavan [16]) to bound the difference $|Y_r - Y_0|$:

$$Pr[|Y_r - Y_0| \geq \lambda] \leq 2\exp(-\lambda^2/2r)$$

Plugging in the appropriate values for Y_r, Y_0 and $\lambda = \epsilon Y_0$ gives us the desired result.

We can prove an identical lemma for the values of f_{out} and $f^*_{out}(r, D, \theta)$ defined analogously.

COROLLARY 5.1. *After r samples,*

$$Pr[|f_{out} - f^*_{out}(r, D, \theta)| \geq \epsilon f^*_{out}(r, D, \theta)] \leq$$

$$2\exp(-\frac{\epsilon^2 f^*_{out}(r, D, \theta)}{2r})$$

We have shown that the number of distinct elements seen after r samples is very close to its expectation. We now show that this implies that the maximum likelihood estimator will produce a low ratio error.

LEMMA 5.2. *Suppose that $|f_{out} - f^*_{out}(r, D, \theta)| \leq \epsilon f^*_{out}(r, D, \theta)$ and $r \geq \frac{1}{Pr_{\widehat{D}}(\mathcal{P}_{\widehat{D}})}$. Then the ratio error,* $\max(\widehat{D}/D, D/\widehat{D}) \leq 1 + 2\epsilon$.

Proof. For simplicity of notation let $p_i = Pr_i(Z_{D,\theta})$ and $\hat{p}_i = Pr_i(Z_{\widehat{D},\theta})$, and N and $\hat{N}$ be the normalizing values for the two distributions. Let us consider the value of $f^*_{out}(r, D, \theta)$.

$$f^*_{out}(D, \theta) = \sum_{i=1}^{D}(1-p_i)^r$$

The estimate $\widehat{D}$ returned by the algorithm is such that $f^*_{out}(r, \widehat{D}, \theta) = f_{out} \leq f^*_{out}(r, D, \theta)(1+\epsilon)$. Assume that $f_{out} \geq f^*_{out}$ (The proof is identical in the opposite case). Then $\widehat{D} \geq D$ and we seek to bound the ratio $\widehat{D}/D$ from above. By corollary 5.1, $f_{out} \leq (1+\epsilon) f^*_{out}(r, D, \theta)$ w.h.p.

Therefore, $1 + \epsilon \geq$

$$\frac{f^*_{out}(r, \widehat{D}, \theta)}{f^*_{out}(r, D, \theta)} = \frac{\sum_{i=1}^{\widehat{D}}(1-\hat{p}_i)^r}{\sum_{i=1}^{D}(1-p_i)^r} \geq \frac{\sum_{i=\widehat{D}-D+1}^{\widehat{D}}(1-\hat{p}_i)^r}{\sum_{i=1}^{D}(1-p_i)^r}$$

The second inequality follows since all of the terms in the numerator are positive. Now consider the ratio of the i^{th} terms of each sum. Since the ratio of the sum is bounded, there must exist at least one value i where the ratio of the individual terms is bounded by $(1+\epsilon)$. A simple analysis shows that the ratio decreases as i increases, and thus the lowest ratio is achieved by the last term.

$$1+\epsilon \geq \frac{(1-\hat{p}_{\hat{D}})^r}{(1-p_D)^r} = \left(\frac{1-\hat{p}_{\hat{D}}}{1-p_D}\right)^r$$
$$\geq \left(1+\frac{p_D-\hat{p}_{\hat{D}}}{1-p_D}\right)^r \geq \exp\left(r\frac{p_D-\hat{p}_{\hat{D}}}{1-p_D}\right)$$
$$2\epsilon \geq r\cdot\frac{p_D-\hat{p}_{\hat{D}}}{1-p_D} \geq r(p_D-\hat{p}_{\hat{D}})$$
$$\geq \frac{1}{\hat{p}_{\hat{D}}}(p_D-\hat{p}_{\hat{D}})$$
$$1+2\epsilon \geq \frac{p_D}{\hat{p}_{\hat{D}}} = \left(\frac{\hat{D}}{D}\right)^{\theta}\cdot\left(\frac{\hat{N}}{N}\right)$$

For $\theta \geq 1$ the result follows since $\hat{N} \geq N$. For $\theta < 1$ it can be shown that $\frac{\hat{N}}{N} \geq \left(\frac{\hat{D}}{D}\right)^{1-\theta}$. Combining the two inequalities we get $\frac{\hat{D}}{D} \leq 1+2\epsilon$.

The probability of success of this procedure is $2\exp(-\frac{\epsilon^2 f^*_{out}(r,D,\theta)}{2r})$. To establish the result with constant probability, we have to show that even after r as above samples, $f^*_{out}(D,\theta)$ is sufficiently large.

LEMMA 5.3. *After* $r = (Pr_{\hat{D}}(P_{\hat{D}}))^{-1} = \hat{N}\hat{D}^{\theta}$ *samples,* $f^*_{out}(r,D,\theta) = \Omega(D)$.

Proof. Consider the elements of rank $D/2$ through D, and let i be one of these elements. $Pr_i(\mathcal{P}_D) = \frac{1}{i^{\theta}N} \leq \frac{1}{(D/2)^{\theta}N}$. Therefore the probability that i is not present in the sample of size r is at most $(1-\frac{1}{(D/2)^{\theta}N})^r \leq \exp(-\frac{\hat{N}\hat{D}^{\theta}}{(D/2)^{\theta}N}) \leq \exp(-2^{\theta}(1+2\epsilon)^{2\theta}) = \Theta(1)$. The expected number of items not present is therefore $\Omega(D)$.

From the above two lemmas, the main theorem for Zipfian distributions follows:

THEOREM 5.1. *Given* $r = N_{D,\theta}(1+2\epsilon)((1+2\epsilon)D)^{\theta}$ *samples the algorithm will produce an estimate* $\widehat{D}$*, such that* $\max(\widehat{D}/D, D/\widehat{D}) < 1+\epsilon$ *with probability of error less than* $2\exp(-\Omega(D)\epsilon^2)$.

We have chosen here to analyze in detail the case of Zipfian distributions. Observe that the main algorithm works even for non-Zipfian distributions. As long as the value of $f^*_{out}(r,\mathcal{P})$ can be estimated the algorithm presented above is well defined. However, the exact value for r and the estimation error need to be recomputed for each family of distributions.

6 Experimental Results

In this section we validate our estimator by comparing it against GEE (the only other sampling based estimator with analytical guarantees), as well as AE (Adaptive Estimator), which was shown to outperform all of the other heuristic estimators in the experiments of Charikar et al [3]. We will refer to our estimator as ZE (for Zipfian Estimator). We first test the three estimators on synthetic data. We generate datasets according to a Zipfian distribution with skew parameter $\theta \in \{0, 0.5, 1\}$. We vary the number of distinct elements from $10k$ to $100k$, and vary the size of the overall database from $100k$ to $1000k$. We present here the results of all three estimators on a dataset of 500,000 elements drawn under the corresponding Zipfian distribution on 50000 elements. The results for the other synthetic scenarios were almost identical to the ones shown.

Further, we tested the estimators on several real-world datasets that we assumed followed a Zipfian distribution. We present the results on the *Router* dataset was obtained from [18]. It is a packet trace from the Internet Traffic Archive. We are trying to predict the number of distinct IP addresses served by the router. Although this distribution is not a pure Zipfian, as the probabilities of the most frequent values and the least frequent values are a little bit skewed, the bulk of the data follows a Zipfian distribution with $\theta \approx 1.6$.

6.1 Estimating Zipfian Skew Parameter All of the analytical results above assumed that the parameter θ was known to us ahead of time. In practice, we can estimate the parameter from the data sampled for distinct value counts. Let f_i be the frequency of the i^{th} most common element. Then in expectation, $f_i = rp_i = \frac{r}{Z}i^{-\theta}$, and $\log f_i = \log\frac{r}{Z} - \theta\log i$. Since $\frac{r}{Z}$ is independent of i, we can estimate θ by doing linear regression on the log-log scale of the f_i vs i data. Many of the real world datasets (including *Router*) follow a Zipfian distribution for the bulk of the data, but not for the first or the last few elements, which can change the θ parameter of the sample. To counteract this problem we ignored the top 100 frequencies, as well as all of the elements which did not appear at least 10 times in the sample while estimating the value of the θ. Note that the value of the parameter was estimated for the synthetic datasets as well, even when we knew the exact value that generated the dataset.

6.2 Discussion In the synthetically generated datasets the ZE estimator was competitive with AE and often outperformed it. This is not surprising since ZE was designed particularly for Zipfian datasets. The GEE estimator performed poorly on most of the data, often having the results err by more than a factor of 5 even after a large sample.

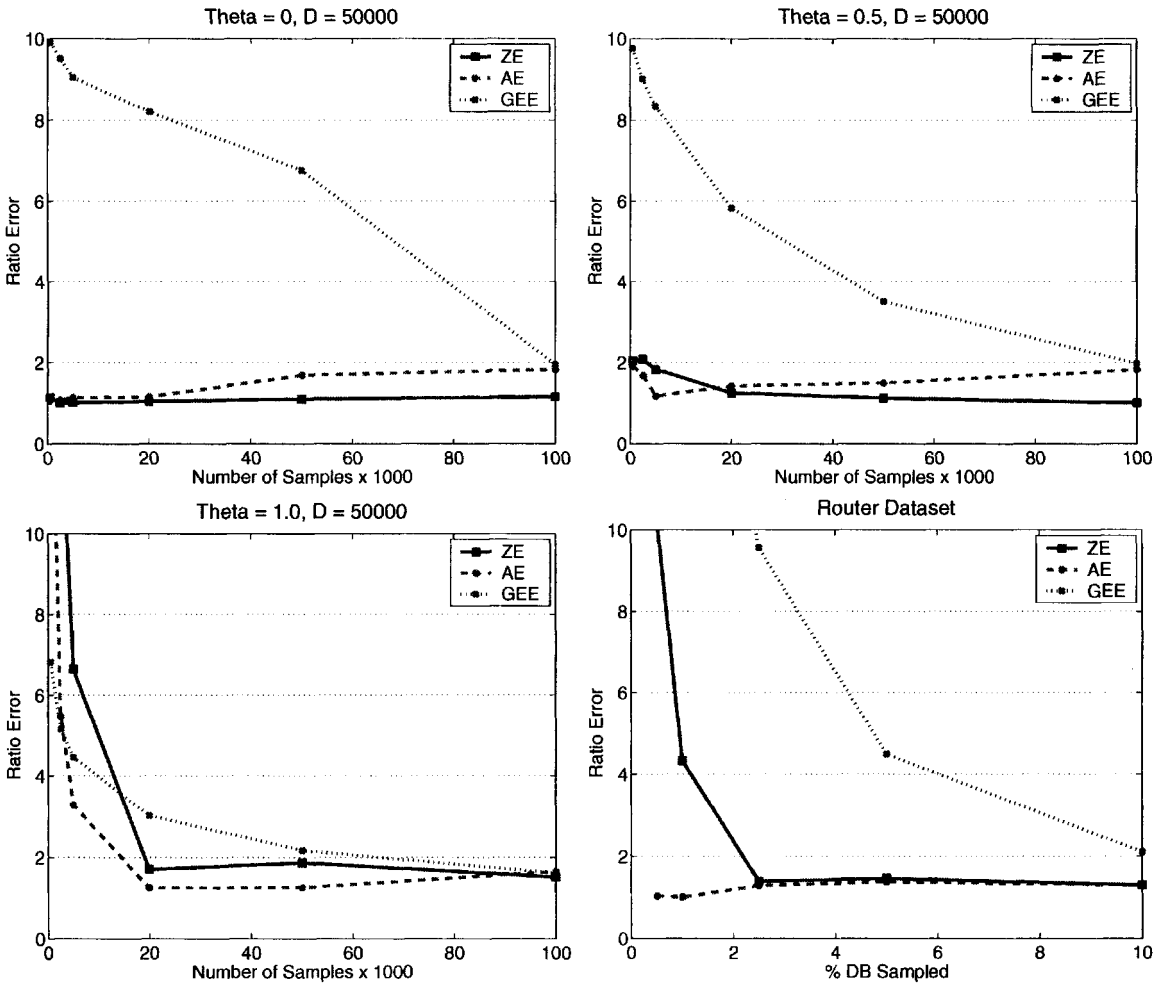


Figure 1: Empirical results on synthetic and real world data

On the real-world dataset AE performed very well, and ZE was competitive after about 2.5% of the database was sampled. One must keep in mind that the router dataset had very high skew ($\theta \approx 1.6$), and ZE was given fewer estimates than would be required by the theoretical guarantees, but performed well nonetheless. On real world data, the Zipfian Estimator, ZE grossly outperformed the other estimator with guaranteed error bounds. The results were comparable only after a 10% fraction of the database was sampled. It is important to note that because of the random access nature of the estimation algorithms, a 10% sample requires almost as much time to compute as a full linear scan of the database.

Although ZE and AE performed equally well, one must remember that the Zipfian Estimator presented here is guaranteed to perform well with high probability on all Zipfian inputs, while the AE estimator is only a heuristic and may perform poorly on some of the inputs. In particular, the error of AE often rises as more samples are taken from the database. When compared to the only other estimator which has guarantees on its results, GEE, the Zipfian estimator performed much better, often giving results more than 10 times more accurate on the same dataset.

References

[1] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, 1996, pp. 20–29.

[2] Bunge, J., and Fitzpatrick, M. Estimating the Number of Species: A review. *Journal of the American Statistical Association* 88(1993): 364–373.

[3] Charikar, M., Chaudhuri S., Motwani, R., and Narasayya, V. Towards Estimation Error Guarantees for Distinct Values. In *Proceedings of the Nineteenth*

ACM Symposium on Principles of Database System, 2000, pp. 268–279.

[4] Chaudhuri, S., Das, G., and Srivastava, U. Effective Use of Block-Level Sampling in Statistics Estimation. In *Proceedings of ACM-SIGMOD*, 2004.

[5] Durand, M., and Flajolet, P. Loglog Counting of Large Cardinalities. In *Proceedings of 11th Annual European Symposium on Algorithms (ESA)*, 2003, pp. 605–617.

[6] Flajolet P., and Martin, G.N. Probabilistic counting. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, 1983, pp 76–82.

[7] Gibbons, P.B. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *Proceedings of the 27th International Conference on Very Large Databases*, 2001.

[8] Goodman, L. On the estimation of the number of classes in a population. *Annals of Math. Stat.* 1949, pp. 72-579.

[9] Haas, P.J., Naughton, J., F., Seshadri, S., and Stokes, L. Sampling-based Estimation of the Number of Distinct Values of an Attribute. In *Proceedings of the 21st International Conference on Very Large Databases*, 1995.

[10] Haas, P.J., and Stokes, L. Estimating the number of classes in a finite population. In *Journal of the American Statistical Association* 1998, pp. 1475–1487.

[11] Heising, W.P. *IBM Systems J.* 2 (1963).

[12] Hou, W., Ozsoyoglu, G., and Taneja, B. Statistical estimators for relational algebra expressions. In *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, 1988.

[13] Hou, W., Ozsoyoglu, G., and Taneja, B. Processing aggregate relational queries with hard time constraints. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, 1989.

[14] Kleinberg, J., Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. The Web as a graph: measurements, models and methods. In *Proceedings of the International Conference on Combinatorics and Computing*, 1999.

[15] Knuth, D.E. **Sorting and Searching**. Volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1971.

[16] Motwani, R., and Raghavan, P. **Randomized Algorithms**.Cambridge University Press, 1995.

[17] Mitzenmacher, M. A Brief History of Generative Models for Power Law and Lognormal Distributions. In *Proceedings of the 39th Annual Allerton Conference on Communication, Control, and Computing*, 2001, pp. 182-191.

[18] A packet trace from the internet traffic archive. `http://ita.ee.lbl.gov/html/contrib/DEC-PKT.html`.

[19] Shlosser, A. On estimation of the size of the dictionary of a long text on the basis of a sample. *Engrg Cybernetics*, 1981, pp. 97–102.

[20] Zipf, G. **The Psycho-Biology of Language**. Houghton Mifflin, Boston, MA, 1935.

A Random-Surfer Web-Graph Model

Avrim Blum* T-H. Hubert Chan* Mugizi Robert Rwebangira*

Abstract

In this paper we provide theoretical and experimental results on a random-surfer model for construction of a random graph. In this model, a new node connects to the existing graph by choosing a start node uniformly at random and then performing a short random walk. We show that in certain formulations, this results in the same distribution as the preferential-attachment random-graph model, and in others we give a direct analysis of power-law distribution of degrees or "virtual degrees" of the resulting graphs. We also present experimental results for a number of settings of parameters that we are not able to analyze mathematically.

1 Introduction

There has been substantial work in recent years on the preferential attachment random graph model. In this model, a graph is constructed in the following manner. Nodes arrive one at a time, and each new node makes k connections to the existing graph. However, unlike classic random graph models, these connections are not made uniformly at random, but rather with probability proportional to the degree of existing nodes in the graph. This process is known to produce graphs with a power law degree distribution [2] and that have high conductance [15], and has been proposed as a model for graphs such as the graph of links between pages on the World Wide Web.

A natural question that arises when considering the preferential attachment model is *why*: why should a new node connect to existing nodes with probability proportional to their degree? Is it because we imagine that high degree nodes are "better" (and the degree of a node is an indicator of its quality) or is it for some other reason?

The starting point for this paper is the observation that a simple "random surfer" model provides a natural explanation for preferential attachment. In particular, imagine that each new node (a person setting up their web page) puts in k links into the existing graph by picking a random start node and then randomly surfing the web until it finds k interesting pages to connect to. Imagine also that each page is equally likely to be interesting to the surfer and each link is bidirectional (so we have an undirected graph). Then, if the probability p of a page being "interesting" is sufficiently small, these connections will be made (approximately) according to the stationary distribution of the walk, which is exactly the preferential attachment distribution. Furthermore, since such graphs have high conductance [15], one should not need an extremely low value of p for this to hold. Thus, preferential-attachment may arise even if all nodes are in a sense "equally good", and differences between degrees may not necessarily be an indicator of differences in inherent quality.

Based on this as motivation, in this paper we propose and analyze several "random surfer" models for graph construction. We also give a number of experimental results, both for models we know how to analyze and for several that we do not. Interestingly, the models we are best able to analyze in this setting are all *directed* graph models, rather than undirected models as the one described above. In addition, some of these models can be thought of as making a bridge between the preferential attachment model and the copying model of [13].

2 Random Surfer Models

In this section, we describe several random surfer models that we will examine in the rest of the paper. In each model, nodes arrive one at a time, making k connections to the existing graph. In some models these connections will be viewed as directed edges, and in some as undirected edges. All our models begin with a single start node v_0 having k self-loops. In general, we use v_t to denote the vertex added in the t^{th} step, and n as the total number of vertices.

To motivate our first model, note that if the connections to the existing graph are made uniformly at random, then we have an online version of the standard Erdos-Renyi graph model, and with high probability the maximum degree will be $O(\log n)$. On the other hand, suppose we make each connection by first picking a random start node in the existing graph, and then taking a random walk of *exactly one step*. Then, in the directed case, this will just produce a star (all edges will point to the root v_0), and in the undirected case, it is not hard to show that there is a good chance this produces something star-like of maximum degree $\Omega(n)$.[1]

*Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213. {*avrim, hubert, rweba*} *at cs.cmu.edu*. Supported in part by the National Science Foundation under grants CCR-0122581, IIS-0312814, and CCF-0514922.

[1] In particular, if the graph is currently a star of t nodes, then there is a $(t-1)/t$ chance the random start node is one of the spokes, so the 1-step walk moves to the center and the next edge maintains the star. More

However, if we flip a coin and with probability $p \in (0,1)$ connect to the random start and with probability $1-p$ take a 1-step walk, then we get something much more natural.

MODEL 1. (1-STEP WALK WITH SELF-LOOP) *In this model, we are given parameters k and p. At time t, vertex v_t makes k connections to the existing graph by repeating the following process k times:*

1. *Pick an existing node v uniformly at random from $\{v_0, \ldots, v_{t-1}\}$.*
2. *With probability p stay at v; with probability $1-p$ take a 1-step walk to a random neighbor of v.*
3. *Add an edge from v_t to the current node.*

In the directed version, the edges added are directed from v_t into the existing graph. In the undirected version, edges are undirected.

Our next model is a walk of the form given in the Introduction: instead of taking one step, we keep walking until we find a node of interest and then connect there. In order to make the model easier to think about, for the case $k > 1$ we imagine after each connection we re-start at a new random start node when performing the next walk.

MODEL 2. (RANDOM WALK WITH COIN FLIPS) *In this model, we are again given parameters k and p. At time t, vertex v_t makes k connections to the existing graph by repeating the following process k times:*

1. *Pick an existing node v uniformly at random from $\{v_0, \ldots, v_{t-1}\}$.*
2. *Flip a coin of bias p*
3. *If the coin comes up heads add an edge from v_t to the current node and stop.*
4. *If the coin comes up tails, move to a random neighbor of the current node and go back to (2).*

In the directed version, the edges added are directed from v_t into the existing graph. In the undirected version, edges are undirected.

3 Theoretical results

3.1 Directed Walk with Self-Loop. Our first (simple) result is that the directed version of Model 1 with $p = 1/2$ is exactly the preferential attachment model.

THEOREM 3.1. *The directed version of Model 1, with $p = 1/2$, has the same distribution as preferential attachment.*

generally, with high probability, the number of non-leaf vertices remains small and the expected degree of the initial node is $\Omega(n)$ See Section 3.3.

Proof. First, notice that the graph is necessarily a DAG, with all edges pointing backwards in time, and each vertex has an out-degree of k. Now, consider some vertex u in the existing graph with in-degree d_u. An edge from the new vertex v_t will connect to u if either the process chooses u as the start node of its walk and does not take a step, or else it chooses one of u's in-neighbors u' as the start node and *does* take a step, selecting the edge from u' to u. The first case has probability p/t, and the second case has probability $(1-p)d_u/(kt)$. For $p = 1/2$, the sum of these two quantities is $(k + d_u)/(2kt)$ which is exactly proportional to the total degree $k + d_u$ of u. ■

One implication of Theorem 3.1 is that for $p > 1/2$, the model is a mixture of preferential-attachment and uniform-random connections. That is, the case $p > 1/2$ can be viewed as: with probability $2p - 1$ choose a neighbor uniformly at random, and with the remaining probability choose a neighbor with probability proportional to degree. This process is known to produce power-law degree distributions. For general $p \in (0,1)$, we now give an argument for power-law degree distributions from first principles.

Let $d_i(t)$ be the number of nodes with in-degree i at step t, and $D_i(t)$ be the expectation of $d_i(t)$. We now analyze $D_i(t)$ via the following equation.

$$D_i(t+1) = D_i(t) + \tag{3.1}$$

$$\frac{pk}{t} \cdot \{D_{i-1}(t) - D_i(t)\} + \tag{3.2}$$

$$\frac{(1-p)k}{t} \cdot \{(i-1)D_{i-1}(t) - iD_i(t)\} \cdot \frac{1}{k}. \tag{3.3}$$

Observe that the number of nodes with in-degree i increases if the new node connects to an existing node of degree $i-1$ and decreases if the new node connects to one of degree i. The term in (3.2) is due to the fact that with probability p the new node is connected to an existing node picked uniformly at random. The term in (3.3) corresponds to the case when with probability $1-p$, the new node connects to a random out-going neighbor of a randomly picked node. The factor k appears in both (3.2) and (3.3) because each new node makes k connections to the existing nodes. The factor $1/k$ appears only in (3.3) because in the case where a random out-going neighbor is chosen, there are k possible choices. We require for large enough t, a new node does not make more than one connection to an existing node.

THEOREM 3.2. *There exists a constant $C > 0$ such that as t tends to infinity, $D_i(t) \sim Ci^{-\frac{2-p}{1-p}}t$.*

Proof. Using the above equations, the proof follows directly from the techniques of Kumar et al. [13], Cooper and Frieze

[10], and Mitzenmacher [16], which allow one to determine the asymptotic behavior of $D_i(t)$.

In particular, for each i, we make the substitution $D_i(t) = c_i t$ in (3.1) - (3.3) to obtain the following equation.

$$(3.4)\quad c_i = pk \cdot \{c_{i-1} - c_i\} + (1-p) \cdot \{(i-1)c_{i-1} - ic_i\}$$

Rearranging (3.4), we have

$$\frac{c_i}{c_{i-1}} = 1 - \frac{2-p}{1+pk+(1-p)i} \sim 1 - \frac{2-p}{1-p} \cdot \frac{1}{i},$$

for large values of i. Using the fact that $\Pi_{i=1}^{n}(1+\lambda/i) = \Theta(n^{\lambda})$, we have

$$c_i = \Theta(\Pi_{j=1}^{i}(1 - \frac{2-p}{1-p} \cdot \frac{1}{j})) \sim Ci^{-\frac{2-p}{1-p}},$$

for some $C > 0$. ∎

Moreover, using Theorem 4 of [10], one can also show that $d_i(t)$ is concentrated around its mean, as stated in the following theorem.

THEOREM 3.3. *For any $\rho > 0$,*

$$Pr(|d_i(t) - D_i(t)| \geq \rho) \leq \exp(-\frac{\rho^2}{8kt}).$$

3.2 Directed Walk with Coin Flipping. We now consider the directed case of Model 2, for the case $k = 1$. That is, we connect a new node to the existing graph by picking a start node u uniformly at random, and then performing a random walk, where at each step we halt the walk with probability p. Since $k = 1$, we can view the random graph constructed as a tree, in which the initial node is the root and every other node has an edge directed to its parent.

To analyze this walk, we define a notion of the *virtual degree* of a node that is related to the node's actual degree, but also contains terms for the local neighborhood of the node as well. We then prove that for this definition, at each step the expected increase in virtual degree of any given node is proportional to the virtual degree itself. (The virtual degree itself is a fractional quantity, and at each step will change by at most some constant.) Using this, we can show that the expected virtual degrees follow a power-law, and we can also give some bounds on their concentration about their means. Moreover, we can give a crude lower bound on the expected *real* degree of a given node, which is comparable to its expected virtual degree.

However, our concentration bounds are not sharp enough to give a true proof that the virtual degrees, or the real degrees, follow a power law.

DEFINITION 1. *Suppose u is a node in the tree. For $i \geq 0$, denote $L_i(u)$ to be the set of level i descendants of u and $l_i(u) = |L_i(u)|$. For instance, $L_0(u)$ is the set of children, $L_1(u)$ is the set of grandchildren, and so on. Let $\beta = \{\beta_i\}_{i\geq 0}$ be a sequence of real numbers such that $\beta_0 = 1$. The virtual degree of u with respect to β is*

$$\nu(u) = 1 + \sum_{k\geq 0} \beta_k l_k(u).$$

In the definition of virtual degree $\nu(u)$, the leading term 1 corresponds to the parent of u. We require $\beta_0 = 1$, for each child of u should contribute 1 towards the degree of v. We would like the virtual degree to reflect the actual degree of a node, and hence ideally, for $i \geq 1$, we would like β_i to be small. On the other hand, we also want that the expected increase in the virtual degree $\nu(u)$ of node u in each step to be proportional to its current virtual degree. The following theorem states we can satisfy these requirements simultaneously.

THEOREM 3.4. *Suppose we consider the directed walk with coin flipping probability $p \in (0,1)$. Then, there exists $\beta = \{\beta_k\}_{k\geq 0}$, dependent on p, with $\beta_0 = 1$ such that for each node u, the expected increase in $\nu(u)$ from step t to step $t+1$ is $p/t \cdot \nu(u)$. Moreover, for $k \geq 0$, $|\beta_k| \leq 1$, and as k tends to infinity, β_k tends to zero exponentially, i.e. there is some $C > 0$ and $0 < \rho < 1$ such that $|\beta_k| \leq C\rho^k$.*

Proof. We fix the coin flipping probability p and find some sequence β that satisfies the requirements.

For convenience, we denote $q = 1 - p$ and $L_{-1}(u) = \{u\}$. Then, for $i \geq 0$, if a new connection is made to a node in $L_{i-1}(u)$, then the increase in $\nu(u)$ is β_i.

Fix $i \geq 0$. We first calculate the probability that a new connection is made to a node in $L_{i-1}(u)$. Recall that we first pick a node uniformly at random to start the directed random walk. If we end up making a new connection to a node in $L_{i-1}(u)$, we must have begun the random walk at some node in $L_{i-1+j}(u)$, for some $j \geq 0$.

We fix some $j \geq 0$ and calculate the probability that the random walk starts at some node in $L_{i-1+j}(u)$ and ends up at some node in $L_{i-1}(u)$. Note that there are $l_{i-1+j}(u)$ nodes to start and there are j hops to be made. Hence, the probability is $l_{i-1+j}(u)/t \cdot q^j \cdot p$.

It follow that the probability that a new connection is made to some node in $L_{i-1}(u)$ is $\frac{p}{t}\sum_{j\geq 0} q^j l_{i-1+j}(u)$.

Hence, the expected increase in $\nu(u)$ from step t to step $t+1$ is

$$\begin{aligned}
&\sum_{i\geq 0} \beta_i \cdot \frac{p}{t} \sum_{j\geq 0} q^j\, l_{i-1+j}(u) \\
&= \frac{p}{t} \sum_{i\geq 0} \sum_{k\geq i-1} \beta_i\, q^{k-i+1}\, l_k(u) \\
&= \frac{p}{t} \sum_{k\geq -1} \sum_{0\leq i\leq k+1} \beta_i\, q^{k-i+1}\, l_k(u)
\end{aligned}$$

Recall we wish that the above quantity to be equal to

$$\frac{p}{t}\nu(u) = \frac{p}{t} \cdot \{1 + \sum_{k\geq 0} \beta_k\, l_k(u)\}.$$

Hence, it suffices to find a sequence β such that the corresponding coefficients of $l_k(u)$ are equal.

For $k = -1$, we require $\beta_0 = 1$; for $k = 0$, we have $\beta_0 q + \beta_1 = \beta_0$, which implies that $\beta_1 = p$. In general, for $k \geq 0$, we have

$$\beta_k = \sum_{0\leq i\leq k+1} \beta_i\, q^{k-i+1}.$$

Now, suppose $k \geq 0$. Then, we have

$$\begin{aligned}
\beta_{k+1} &= \sum_{0\leq i\leq k+1} \beta_i\, q^{k-i+1} \\
&= \beta_{k+2} + q \sum_{0\leq i\leq k+1} \beta_i\, q^{k-i+1} \\
&= \beta_{k+2} + q\beta_k.
\end{aligned}$$

Hence, the sequence β can be determined by the recurrence $\beta_0 = 1$, $\beta_1 = p$ and for $k \geq 0$, $\beta_{k+2} - \beta_{k+1} + q\beta_k = 0$.

We show inductively that $|\beta_k| \leq 1$. We first observe that this is true for $k = 0, 1, 2$. Assume that the result is true for integers up to $k+1$. In the first case, suppose β_k and β_{k+1} have the same sign. Then, $|\beta_{k+2}| = ||\beta_{k+1}| - q|\beta_k|| \leq 1$, by the induction hypothesis. In the second case, suppose β_k and β_{k+1} have different signs. Hence, $|\beta_{k+2}| = |\beta_{k+1} - q\beta_k| \leq |\beta_{k+1} - \beta_k| = q|\beta_{k-1}| \leq 1$, by the induction hypothesis.

For $p = 3/4$, we have $\beta_k = \frac{k+2}{2^{k+1}}$. Otherwise, for other values of p in $(0,1)$, let $\lambda_1 = (1 - \sqrt{1-4q})/2$ and $\lambda_2 = (1 + \sqrt{1-4q})/2$ and $\beta_k = A\lambda_1^k + B\lambda_2^k$, for some constants A and B. Observe that since $0 < p < 1$, the magnitudes of λ_1 and λ_2 are both strictly less than 1. Hence, in any case, as k tends to infinity, β_k tends to 0 exponentially. ■

For the rest of the discussion, we consider the virtual degree defined with respect to some sequence β that satisfies Theorem 3.4. We next explore how the virtual degree of a particular node changes with time. Define $\nu_t(u)$ to be the virtual degree of node u at step t and t_u to be the time when node u first appears. Then, it follows that $\nu_{t_u}(u) = 1$, since each new node is a leaf when it first appears.

THEOREM 3.5. *For any node u and step $t \geq t_u$, the expectation $E[\nu_t(u)] = \Theta((t/t_u)^p)$.*

Proof. For any $t > t_u$, we have from Theorem 3.4 that

$$E[\nu_t(u)] = (1 + p/(t-1))\, E[\nu_{t-1}(u)].$$

Hence,

$$E[\nu_t(u)] = \Pi_{i=t_u}^{t-1}(1 + p/i) = \Theta((t/t_u)^p).$$

■

We next give an intuition, similar in spirit to [3], of how Theorem 3.5 suggests that the virtual degrees of the random graph should follow a power law. Suppose the random process is run for n steps to form a random graph with n nodes. Then, from Theorem 3.5, the expected virtual degree of the ith node joining the graph is $\Theta((n/i)^p)$. If we let $\kappa \approx \Theta((n/i)^p)$, we would have $i \approx \Theta(n\kappa^{-1/p})$. Observing that nodes joining later should probably have smaller virtual degrees, one might expect that the proportion of nodes having virtual degrees smaller than κ to be $1 - \Theta(\kappa^{-1/p})$. Differentiating this quantity with respect to κ, we conjecture that the proportion of nodes having degree κ should be $\kappa^{-(1/p+1)}$.

Unfortunately, we do not have a strong enough concentration bound that would allow us to make the above intuition rigorous. However, using martingale techniques, we can show that the virtual degree cannot be *too* much larger than its mean for the case when the coin flipping probability $p > 1/2$.

THEOREM 3.6. *There exists a constant $C > 0$ such that for coin flipping probability $p > 1/2$ and any $\rho \geq 1$,*

$$Pr[\nu_t(u) \geq C\rho E[\nu_t(u)]] \leq \exp\{-\rho^2/t_u\}.$$

Proof. Consider a node u and recall that t_u is the time when it first appears. Define $a_i = 1 + p/i$. Recall from the proof of Theorem 3.5 that $E[\nu_t(u)] = \Pi_{i=t_u}^{t-1} a_i = \Theta((t/t_u)^p)$.

Define $Y_i = \nu_i(u)/E[\nu_i(u)]$, for $i \geq t_u$. Then, it follows that $\{Y_i\}$ is a martingale. Define $D_i := Y_i - Y_{i-1}$.

Recall that the sequence $\{\beta_k\}$ tends to zero. Hence, it follows that $|\nu_i(u) - \nu_{i-1}(u)| = \Theta(1)$, and we have $|D_i| = |Y_i - Y_{i-1}| = 1/E[\nu_i(u)] \cdot |\nu_i(u) - a_{i-1}\nu_{i-1}(u)| = 1/E[\nu_i(u)] \cdot |\Theta(1) - \frac{p}{i-1} \cdot \nu_{i-1}(u)| = \Theta(1/E[\nu_i(u)])$, since $\nu_{i-1}(u) = O(i-1)$. Hence, we can let $K_i = \Theta(1/E[\nu_i(u)])$, and so $|D_i| \leq K_i$. By the Azuma-Hoeffding martingale inequality, we have for any $x > 0$,

$$Pr[Y_t - Y_{t_u} \geq x] \leq \exp\{-x^2/2 \sum_{i=t_u+1}^{t} K_i^2\}.$$

Observe that for $p > 1/2$, we have

$$\begin{aligned}\sum_{i=t_u+1}^{t} K_i^2 &\leq \sum_{i=t_u+1}^{t} \Theta(1/E[\nu_i(u)]^2) \\ &= \sum_{i=t_u+1}^{t} \Theta((i/t_u)^{-2p}) \\ &= \Theta(t_u(2p-1)\cdot(1-(t/t_u)^{-(2p-1)})) \\ &= \Theta(t_u).\end{aligned}$$

Hence, for some large enough $C' > 0$, if we put $x = C'\sqrt{st_u}$, we have $Pr[Y_t - Y_{t_u} \geq x] \leq e^{-s}$. Observing that $Y_{t_u} = 1$ and taking $\rho = \sqrt{st_u}$, we have

$$Pr[\nu_t(u) \geq C\rho E[\nu_t(u)]] \leq \exp\{-\rho^2/t_u\},$$

where $C > 0$ is a constant large enough to absorb the 1. ■

3.2.1 A Crude lower bound for the expected *real* degree. Recall that for a given node u in the tree and $i \geq 0$, $L_i(u)$ is the set of level i descendants of u and $l_i(u) = |L_i(u)|$. In particular, $l_0(u)$ is the number of children node u has. We can give a crude lower bound for $l_0(u)$ for any given node u.

THEOREM 3.7. *For any node u and step $t \geq t_u$, the expectation $E[l_0(u)] \geq \Omega((t/t_u)^{p(1-p)})$.*

Proof. Let the number of level i descendants of node u at time step t be $l_i^t(u)$. It follows that

$$\begin{aligned}E[l_0^{t+1}(u)] &= E[l_0^t(u)] \\ &\quad + \frac{p}{t}\cdot\{1 + \sum_{j\geq 0} E[l_j^t(u)](1-p)^{j+1}\} \\ &\geq E[l_0^t(u)] + \frac{p(1-p)}{t}E[l_0^t(u)]\end{aligned}$$

Suppose that for some constant $A > 0$, for some $t > 0$, and α, we have $E[l_0^t(u)] \geq At^\alpha$. Observing that for $t \geq 1$, $(t+1)^\alpha - t^\alpha \leq \alpha t^{\alpha-1}$, we have

$$\begin{aligned}E[l_0^{t+1}(u)] &\geq A\{t^\alpha + \frac{p(1-p)}{t}\cdot t^\alpha\} \\ &\geq A\{(t+1)^\alpha + (p(1-p)-\alpha)t^{\alpha-1}\} \\ &\geq A(t+1)^\alpha,\end{aligned}$$

if we set $\alpha = p(1-p)$.

Note that for $t = t_u + 1$, $E[l_0^t(u)] = \Theta(1)$. Hence, it follows that $E[l_0^t(u)] \geq \Omega((t/t_u)^{p(1-p)})$. ■

3.3 Undirected Walk without Self-loop. We now consider the model mentioned when motivating Model 1 in which a new connection is made to a random neighbor of a randomly selected node. We show that there is a node, namely the initial node, that in expectation has degree linear in the size of the random tree produced. Thus, the self-loop in Model 1 is crucial for producing natural graphs.

THEOREM 3.8. *Under the undirected walk without self-loop model, the expected number of leaves connected to the initial node in the random tree produced is $\Omega(n)$, where n is the number of nodes.*

Proof. Let L_n be number of leaves connected to the initial node v_0 at step n and D_n be the degree of the initial node v_0 at time n.

Suppose we are at step n. With probability at least L_n/n, a leaf of v_0 would be picked and after one jump, a new connection would be made to v_0, causing the number of leaves connecting to v_0 to increase by 1. On the other hand, with probability $\frac{1}{n}\cdot\frac{L_n}{D_n}$, the initial node v_0 is picked and after one jump a new connection is made to an existing leaf, causing the number of leaves connected to v_0 to decrease by 1.

Hence, $E[L_{n+1} - L_n] \geq L_n/n - 1/n\cdot L_n/D_n \geq 1/n\cdot E[L_n - 1]$, with the last inequality holding because $L_n \leq D_n$. Hence, if we let $Z_n = L_n - 1$, we have $E[Z_{n+1}] \geq (1+1/n)E[Z_n]$. Observe that $E[Z_3] > 0$.

Hence, $E[Z_n] \geq \Pi_{i=3}^{n-1}(1+1/i)E[Z_3] = \Omega(n)$ and so $E[L_n] \geq \Omega(n)$. ■

4 Experimental results

All experiments were the average of 100 runs with a size $n = 100,000$ nodes and $k = 1$, i.e. the random graph produced is a tree. In each case, we investigate how the average proportion P_d of nodes having degree d varies with d. Since we wish to observe whether the degree distribution follows a power law, we plot $\log_{10} P_d$ against $\log_{10} d$, for d up to 40. All four models exhibits power-law like phenomena. Figure 5 shows the degree distribution for the four models and they behave similarly, although the maximum degree seen is much larger for the directed models than for the undirected ones.

4.1 Directed walk with self-loops. Figure 1 shows experimentally that the power-law phenomenon exhibited by the degree distribution becomes more apparent as the probability p decreases and the degree d increases. Notice that for $p = 1$, this is just the Erdos-Renyi random graph model, which does not obey the power law. Moreover, the maximum degree seen for $p = 1$ is only about 20. As p gets smaller the graph can be fitted better with a straight line. Note that even for $p = 0.75$, power law phenomenon is exhibited for large degrees d.

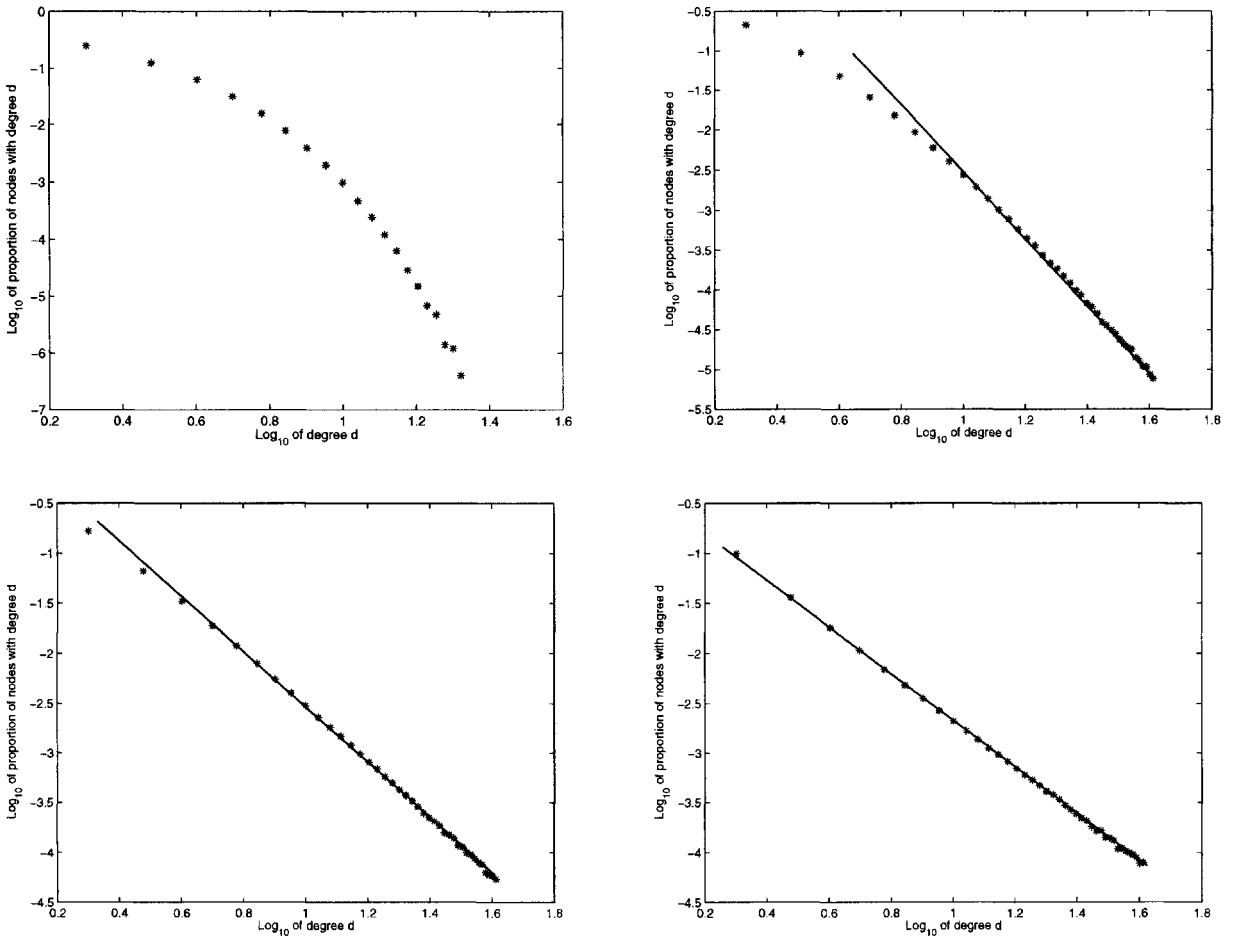


Figure 1: Directed walk with self-loops: (Top-Left) $p = 1$, (Top-Right) $p = 0.75$, (Bottom-Left) $p = 0.5$, (Bottom-Right) $p = 0.25$

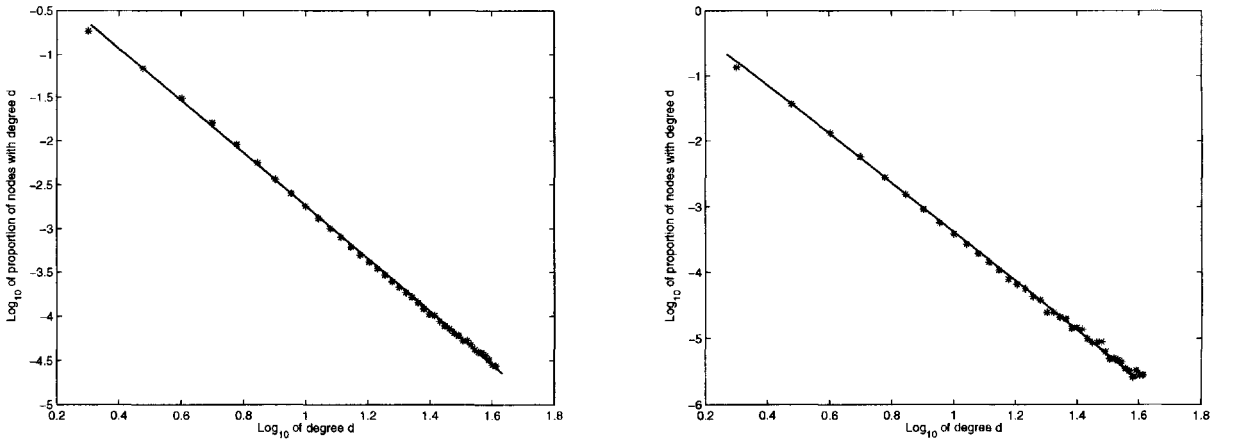


Figure 2: Directed walk with coin flips: (Left) $p = 0.5$ (Right) $p = 0.25$

4.2 Directed walk with coin flips. We do not have a proof, but Figure 2 is very similar to Figure 1, which indicates that in this case the degrees appear to be following a power law.

4.3 Undirected walk with self-loops. We do not know how to analyze this model yet. As seen in Figure 3, there are indications that a power law phenomenon is exhibited by large degrees. On the other hand, the distribution of degrees may follow some other nice distribution that is not very far from power law (e.g. log-normal distribution).

4.4 Undirected walk with coin flips. Like the previous model, this model is not easy to analyze. But Figure 4 shows that the degree sequence does not look too different from the undirected walk with self-loops model. We know theoretically that if p is very small the degree sequence will tend closer to a power law. Figure 4 indeed shows that for $p = 0.05$, the graph can be better fitted with a straight line.

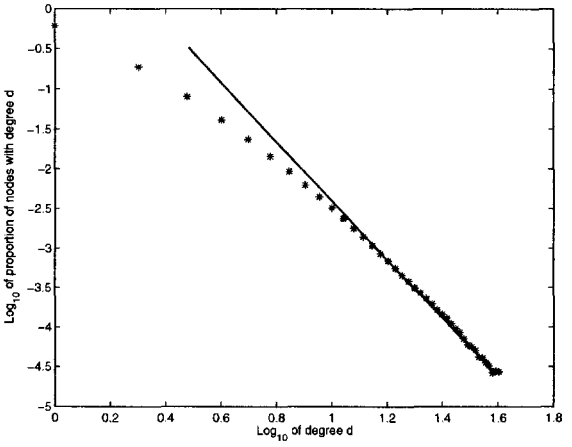


Figure 3: Undirected walk with self-loops: $p = 0.5$

5 Conclusions and Open Questions

In this paper we present some initial analysis and experimental results for several simple random-surfer models for web-graph construction. The models are similar in spirit to the copying model of [13], and in fact the directed case of Model 1, for $k = 1$ is identical to both the copying model and preferential-attachment. There are many open questions including:

1. In the case of the directed walk with self-loops, we can analyze the *expected* virtual degrees and provide some concentration bounds, but do not have a formal proof that the virtual degrees necessarily follow a power-law. Furthermore, even assuming this is the case, we do not have a proof that this implies that the actual degrees must be power-law, though our experimental results show this appears to in fact be the case. Thus, can one give a formal proof that the degrees indeed follow a power law for this model?

2. For the case of the *undirected* walk with self-loops, we know that as p goes to 0, this walk approaches the preferential-attachment distribution. However, experimentally, even for $p = 0.5$ the degrees follow some heavy-tailed distribution. Can one give a formal analysis of the degree distribution in this case?

3. Finally, another issue brought out by this work is that differences between degrees of nodes in the (real) web graph may not necessarily be due to a distinction in quality, but rather just the result of a random walk process. Thus, if one is using degree as a measure of quality, one may just be picking out nodes that have been around the longest. Instead, some measure that

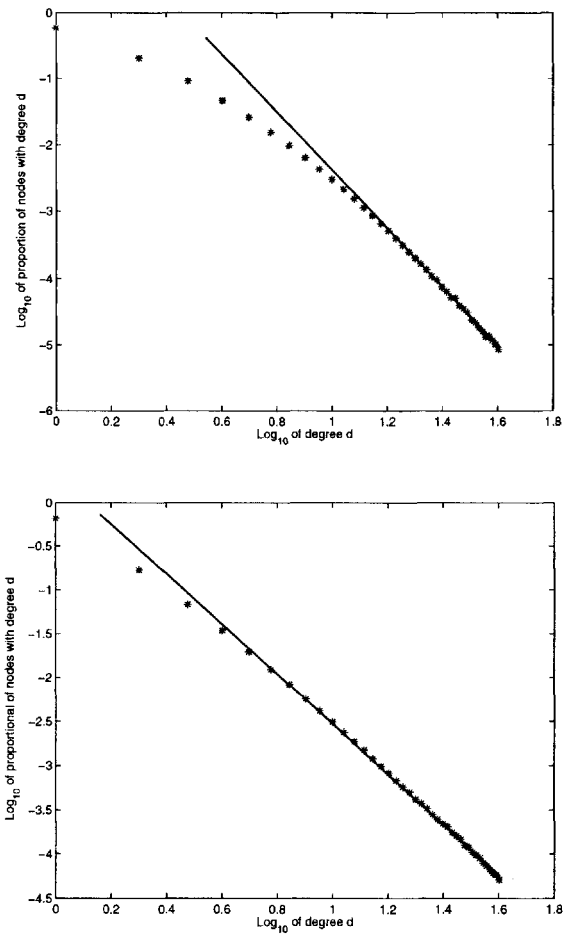


Figure 4: Undirected walk with coin flips: (Top) $p = 0.5$, (Bottom) $p = 0.05$

Degree	Directed walk with self-loops	Directed Walk with coin-flips	Undirected Walk with self-loops	Undirected Walk with coin-flips
1	0.6670	0.6672	0.6136	0.5840
2	0.1669	0.1862	0.1903	0.2044
3	0.06662	0.06929	0.08128	0.09132
4	0.03333	0.03107	0.04137	0.04652
5	0.01900	0.01607	0.02355	0.02596
6	0.01195	0.009108	0.01444	0.01546
7	0.007902	0.005607	0.009301	0.009703
8	0.005547	0.003662	0.006298	0.006354
9	0.004048	0.002524	0.004447	0.004286
10	0.003046	0.001809	0.003242	0.002992
11	0.002332	0.001322	0.002376	0.002134
12	0.001832	0.001006	0.001802	0.001540
13	0.001452	0.0008016	0.001405	0.001131
14	0.001187	0.0006195	0.001088	0.0008657
15	0.0009853	0.0005008	0.0008539	0.0006553
16	0.0007938	0.0004128	0.0006968	0.0005115
17	0.0007005	0.0003486	0.0005608	0.0003950
18	0.0005839	0.0002924	0.0004531	0.0003122
19	0.0005009	0.0002455	0.0003842	0.0002471
20	0.0004400	0.0002118	0.0003121	0.0002031
21	0.0003731	0.0001846	0.0002707	0.0001653
22	0.0003280	0.0001637	0.0002300	0.0001355
23	0.0003001	0.0001426	0.0001990	0.0001082
24	0.0002559	0.0001213	0.0001652	0.0000956
25	0.0002188	0.0001054	0.0001454	0.0000750
26	0.0002020	0.0001018	0.0001289	0.0000639
27	0.0001860	0.0000872	0.0001103	0.0000520
28	0.0001643	0.0000778	0.0000954	0.0000511
29	0.0001545	0.0000720	0.0000851	0.0000395
30	0.0001382	0.0000642	0.0000708	0.0000354
31	0.0001221	0.0000604	0.0000594	0.0000313
32	0.0001116	0.0000528	0.0000564	0.0000240
33	0.0001039	0.0000529	0.0000511	0.0000219
34	0.0000972	0.0000475	0.0000415	0.0000184
35	0.0000904	0.0000425	0.0000407	0.0000162
36	0.0000789	0.0000396	0.0000353	0.0000131
37	0.0000735	0.0000395	0.0000323	0.0000136
38	0.0000649	0.0000362	0.0000264	0.0000118
39	0.0000602	0.0000325	0.0000277	0.0000103
40	0.0000543	0.0000282	0.0000272	0.0000086
Max degree seen in 100 runs	1623	20612	325	138

Figure 5: Average proportion of nodes having different degrees under different models with $n = 100,000$, $p = 0.5$ and 100 runs

examines the degree of a node *relative* to what one would expect given the time the node has been in the system might be more appropriate.

References

[1] W. Aiello, F.R.K. Chung, and L. LU. A random graph model for massive graphs. *Proc. of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 171–180, 2000.

[2] Reka Albert and Albert-Laszlo Barabasi. Topology of evolving networks: Local events and universality. *Physical Review Letters*, pages 5234–5237, 2000.

[3] Sagy Bar, Mira Gonen, and Avishai Wool. An incremental super-linear internet topology model. *5th annual Passive and Active Measurement Workshop*, 2004.

[4] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, pages 509–512, 1999.

[5] Bollobas and O.Riordan. The diameter of a scale free random network.

[6] Bollobas and O.Riordan. *Handbook of Graphs and Networks*. Wiley VCH, Berln, 2002.

[7] Bollobas, O.Riordan, J.Spencer, and G.Tusanady. The degree sequence of a scale free random graph process. *Random Structures and Algorithms*, pages 279–290, 2001.

[8] F.R.K. Chung, L.LU, and V. Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, pages 21–33, 2003.

[9] F.R.K. Chung, L.LU, and V. Vu. The spectra of random graphs with expected degrees. *Proceedings of National Academies of Science*, pages 6313–6318, 2003.

[10] C. Cooper and A. M. Frieze. A general model of undirected web graphs. *Random Structures and Algorithms*, pages 311–335, 2003.

[11] P. Erdos and A. Renyi. On random graphs i. *Publicationes, Mathematicae, Debrecen*, pages 290–297, 1959.

[12] M. Faloutos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, 1999.

[13] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. *Proc. IEEE Symposium on Foundations of Computer Science*, 2000.

[14] M. Mihail and C. H. Papadimitriou. On the eigenvalue powerlaw. *Randomization and Approximation Techniques, 6th International Workshop*, pages 254–262, 2002.

[15] M. Mihail, C. H. Papadimitriou, and A. Saberi. On certain connectivity properties of the internet topology. *Proc. IEEE Symposium on Foundations of Computer Science*, 2003.

[16] M. Mitzenmacher. A brief history of generative models for lognormal and power law distributions.

[17] H.A. Simon. On a class of skew distribution functions. *Biometrika*, pages 425–440, 1955.

[18] Gilbert Strang. *Linear Algebra and its Applications*. Harcourt Brace Jacanovich, 1988.

[19] D.J. Watts. *Small Worlds:They Dynamics of Networks Between Order and Randomness*. Princeton University Press, Princeton, 1999.

[20] G. Yule. A mathematical theory of evolution based on the theories of j.c. willis. *Philosophical Transactions of the Royal Society of London (series B)*, pages 21–87, 1925.

Asymptotic Optimality of the Static Frequency Caching in the Presence of Correlated Requests*

Predrag R. Jelenković† Ana Radovanović‡

Abstract

Renewed interest in caching algorithms stems from their application to content distribution on the Web. When documents are of equal size and their requests are independent and equally distributed, it is well known that static algorithm that keeps the most frequently requested documents in the cache is optimal. However, there are no explicit caching algorithms that are provably optimal when the requests are statistically correlated. In this paper, we show, maybe somewhat surprisingly, that keeping the most frequently requested documents in the cache is still optimal for large cache sizes even if the requests are strongly correlated. We model the statistical dependency of requests using semi-Markov modulated processes that can capture strong statistical correlation, including the empirically observed long-range dependence in the Web access sequences.

Although frequency algorithm and its practical version least-frequently-used policy is not commonly used in practice due to their complexity and static nature, our result provides a benchmark for evaluating the popular heuristic schemes. In particular, an important corollary of our main theorem and recent result from [9] is that the widely used least-recently-used heuristic is asymptotically near-optimal under the semi-Markov modulated requests and generalized Zipf's law document frequencies.

Keywords: Web caching, cache fault probability, average-case analysis, lest-frequently-used caching, least-recently-used caching, semi-Markov processes, long-range dependence

1 Introduction

One of important problems facing current and future network designs is the ability to store and efficiently deliver a huge amount of multimedia information in a timely manner. Web caching is widely recognized as an effective solution that improves the efficiency and scalability of multimedia content delivery, benefits of which have been repeatedly verified in practice.

Caching is essentially a process of storing information closer to users so that Internet service providers, delivering a given content, do not have to go back to the origin servers every time the content is requested. It is clear that keeping more popular documents closer to the users can significantly reduce the traffic between the cache and the main servers and, therefore, improve the network performance, i.e., reduce the download latency and network congestion. One of the key components of engineering efficient Web caching systems is designing document placement/replacement algorithms (policies) that are managing cache content, i.e., selecting and possibly dynamically updating a collection of cached documents.

The main tendency in creating and implementing these algorithms is minimizing the long-term fault probability, i.e., the average number of misses during a long time period. In the context of equal size documents and independent reference model, i.e., independent and identically distributed requests, it is well known (see [5], Chapter 6 of [13]) that keeping the most popular documents in the cache optimizes the long term cache performance; throughout this paper we refer to this algorithm as *static frequency caching*. A practical implementation of this algorithm is known as Least-Frequently-Used rule (LFU). However, the previous model does not incorporate any of the recently observed properties of the Web environment, such as: variability of document sizes, presence of temporal locality in the request patterns (e.g., see [8], [12], [2], [6], [7] and references therein), variability in document popularities (e.g., see [3]) and retrieval latency (e.g., see [1]).

Many heuristic algorithms that exploit the previously mentioned properties of the Web environment have been proposed (e.g., see [7], [5], [11] and references therein). However, there are no explicit algorithms that are provably optimal when the requests are statistically correlated even if documents are of equal size. Our main result of this paper, stated in Theorem 3.1 of Section 3, shows that, under the general assumptions of semi-Markov modulated requests, the static frequency caching algorithm is still optimal for large cache sizes. The semi-Markov modulated processes, described in Section 2, are capable of modeling a wide range of statistical correlation, including the long-range dependence (LRD) that was repeatedly experimentally observed in Web access patterns; this type of models was recently used in [9] and their potential confirmed on real Web traces in [8]. In Section 4, under mild additional assumptions, we show how our result extends to variable page sizes. Our optimality result provides a benchmark for evaluating other heuristic schemes,

*This work is supported by the NSF Grant No. 0092113.

†Electrical Engineering Department, Columbia University.

‡Mathematical Sciences Department, IBM Research.

suggesting that any heuristic caching policy that approximates well the static frequency caching should achieve the nearly-optimal performance for large cache sizes. In particular, in conjunction with our result from [9], we show that a widely implemented Least-Recently-Used (LRU) caching heuristic is, for semi-Markov modulated requests and generalized Zipf's law document frequencies, asymptotically only a factor of 1.78 away from the optimal.

2 Modeling statistical dependency in the request process

In this section we describe a semi-Markov modulated request process. As stated earlier, this model is capable of capturing a wide range of statistical correlation, including the commonly empirically observed LRD. This approach was recently used in [9], where one can find more details and examples.

Let a sequence of requests arrive at Poisson points $\{\tau_n, -\infty < n < \infty\}$ of unit rate. At each point τ_n, we use R_n, $R_n \in \{1, 2, \dots, \}$, to denote a document that has been requested, i.e., the event $\{R_n = i\}$ represents a request for document i at time τ_n; we assume that the sequence $\{R_n\}$ is independent of the arrival Poisson points $\{\tau_n\}$ and that $\mathbb{P}[R_n = i] > 0$ for all i and $\mathbb{P}[R_n < \infty] = 1$.

Next, we describe the dependency structure of the request sequence $\{R_n\}$. We consider the class of finite-state, stationary and ergodic semi-Markov processes J, with jumps at almost surely strictly increasing points $\{T_n, -\infty < n < \infty\}$, $T_0 \le 0 < T_1$. The process $\{J_{T_n}, -\infty < n < \infty\}$ is an irreducible Markov chain with finitely many states $\{1, \dots, M\}$ and transition matrix $\{p_{ij}\}$. The explicit construction of process J_t, $t \in \mathbb{R}$ is presented in Subsection 4.3 of [9]. In addition, J_t is constructed piecewise constant and right-continuous *modulating process*, where

$$J_t = J_{T_n}, \quad \text{if} \qquad T_n \le t < T_{n+1}.$$

Let $\pi_r = \mathbb{P}[J_t = r]$, $1 \le r \le M$, be the stationary distribution of J and independent of Poisson points $\{\tau_n\}$. To avoid trivialities, we assume that $\min_r \pi_r > 0$. For each $1 \le r \le M$, let $q_i^{(r)}, 1 \le i \le N \le \infty$, be a probability mass function; $q_i^{(r)}$ is used to denote the probability of requesting item i when the underlying process J is in state r. Next, the dynamics of R_n are uniquely determined by the modulating process J according to the following equation

(2.1)
$$\mathbb{P}[R_l = i_l, 1 \le l \le n | J_t, t \le \tau_n] = \prod_{l=1}^{n} q_{i_l}^{(J_{\tau_l})}, \qquad n \ge 1,$$

i.e., the sequence of requests R_n is conditionally independent given the modulating process J. Given the properties introduced above, it is easy to conclude that the constructed request process $\{R_n\}$ is stationary and ergodic as well. We will use

$$q_i = \mathbb{P}[R_n = i] = \sum_{r=1}^{M} \pi_r q_i^{(r)}$$

to express the marginal request distribution, with the assumption that $q_i > 0$ for all $i \ge 1$. In addition, assume that requests are enumerated according to the non-increasing order of marginal request popularities, i.e., $q_1 \ge q_2 \ge \dots$. The preceding processes are constructed on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$.

In this paper we are using the following standard notation. For any two real functions $a(t)$ and $b(t)$ and fixed $t_0 \in \mathbb{R} \cup \{\infty\}$ we will use $a(t) \sim b(t)$ as $t \to t_0$ to denote $\lim_{t \to t_0} [a(t)/b(t)] = 1$. Similarly, we say that $a(t) \gtrsim b(t)$ as $t \to t_0$ if $\liminf_{t \to t_0} a(t)/b(t) \ge 1$; $a(t) \lesssim b(t)$ has a complementary definition.

Throughout the paper we will exploit the renewal (regenerative) structure of the semi-Markov process. In this regard, let $\{\mathcal{T}_i\}$, $\mathcal{T}_0 \le 0 < \mathcal{T}_1$, be a subset of points $\{T_n\}$ for which $J_{T_n} = 1$. Then, it is well known that $\{\mathcal{T}_i\}$ is a renewal process and that sets of variables $\{J_t, \mathcal{T}_j \le t < \mathcal{T}_{j+1}\}$ are independent for different j and identically distributed, i.e., $\{\mathcal{T}_i\}$ are regenerative points for $\{J_t\}$. Furthermore, the conditional independence of $\{R_n\}$ given $\{J_t\}$, implies that $\{\mathcal{T}_i\}$ are regenerative points for J_n as well.

Next we define $\mathcal{R}(u,t)$, $1 \le r \le M$, to be a set of distinct requests that arrived in interval $[u,t)$, $u \le t$, and denote by $N_r(u,t)$, $1 \le r \le M$, the number of requests in interval $[u,t)$ when process J_t is in state r. Furthermore, let $N(u,t) \triangleq N_1(u,t) + \dots + N_M(u,t)$ representing the total number of requests in $[u,t)$; note that $N(u,t)$ is Poisson with mean $t-u$.

The following technical lemma will be used in the proof of the main result of this paper.

LEMMA 2.1. *For the request process introduced above, the following asymptotic relation holds:*

(2.2) $$\mathbb{P}[i \in \mathcal{R}(\mathcal{T}_1, \mathcal{T}_2)] \sim q_i \mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1] \ \text{ as } \ i \to \infty,$$

where $\mathcal{R}(u,t) \triangleq \mathcal{R}_1(u,t) \cup \dots \cup \mathcal{R}_M(u,t)$.

Proof: Given in Section 5. ◇

3 Caching policies and the optimality

Consider infinitely many documents of unit size out of which x can be stored in a local memory called cache. When an item is requested, the cache is searched first and we say that there is a cache hit if the item is found in the cache. In this case the cache content is left unchanged. Otherwise, we say that there is a cache fault/miss and the missing item is brought in from the outside world. At the time of a

fault, a decision whether to replace some item from the cache with a missing item has to be made. We assume that replacements are optional, i.e., the cache content can be left unchanged even in the case of fault. A caching algorithm represents a set of replacement rules. We consider a class of caching algorithms whose information decisions are made using only the information of past and present requests and past decisions.

More formally, let $\mathcal{C}_t^\pi$ be a cache content at time t under policy π. When the request for a document R_n is made, the cache with content $\mathcal{C}_{\tau_n}^\pi$, $n = 0, 1, \ldots$, is searched first. If document R_n is already in the cache ($R_n \in \mathcal{C}_{\tau_n}^\pi$), then we use the convention that no document is replaced. On the other hand, if document R_n is not an element of $\mathcal{C}_{\tau_n}^\pi$, then a document to be replaced is chosen from a set $\mathcal{C}_{\tau_n}^\pi \cup \{R_n\}$ using a particular eviction policy. At any moment of request, τ_n, the decision what to replace in the cache is based on $R_1, R_2, \ldots, R_n, \mathcal{C}_{\tau_0}^\pi, \mathcal{C}_{\tau_1}^\pi, \ldots, \mathcal{C}_{\tau_n}^\pi$. Note that this information already contains all the replacement decisions made up to time τ_n. This is the same information as the one used in the Markov decision framework [5].

The set of the previously described cache replacement policies, say $\mathcal{P}_c$, is quite large and contains mandatory caching rules (more typical for a computer memory environment). Furthermore, the set $\mathcal{P}_c$ also contains the static algorithm, that places a fixed collection of documents $\mathcal{C}_t^\pi \equiv \mathcal{C}$ in the cache, and, after this selection is made, the content of the cache is never changed.

Now, define the long-run cache fault probability corresponding to the policy $\pi \in \mathcal{P}_c$ and a cache of size x as

$$P(\pi, x) \triangleq \limsup_{T\to\infty} \frac{\mathbb{E}\left[\sum_{\tau_n \in [0,T]} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]\right]}{T}, \tag{3.3}$$

recall that $\mathbb{E}N(0,T) = T$. Note that we use the lim sup in this definition since the limit may not exist in general.

Next, we show that

$$P(\pi, x) = \limsup_{k\to\infty} \frac{\mathbb{E}\left[\sum_{\tau_n \in (0,\mathcal{T}_k)} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]\right]}{\mathbb{E}N(0,\mathcal{T}_k)}, \tag{3.4}$$

where $\mathcal{T}_k$ are the regenerative points, as defined in the previous section. To this end, for the lower bound, for any $0 < \epsilon < 1$, let $k \equiv k(T,\epsilon) \triangleq \lfloor T(1-\epsilon)/\mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1]\rfloor$, where $\lfloor u \rfloor$ is the largest integer that is less or equal to u. Then, note that

$$\begin{aligned}
&\frac{1}{T}\mathbb{E}\left[\sum_{\tau_n \in (0,\mathcal{T}_k)} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]\right] \\
&\quad \geq \mathbb{E}\left[1[\mathcal{T}_k < T]\frac{\sum_{\tau_n \in (0,\mathcal{T}_k)} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]}{T}\right] \\
&\quad \geq \mathbb{E}\left[\frac{\sum_{\tau_n \in (0,\mathcal{T}_k)} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]}{T}\right] - \mathbb{E}\left[1[\mathcal{T}_k > T]\frac{N(0,T)}{T}\right].
\end{aligned} \tag{3.5}$$

Next, using the Weak Law of Large Numbers for $\mathbb{P}[\mathcal{T}_k > T]$ (as $T \to \infty$) and the fact that $N(0,T)$ is Poisson with mean T in the preceding inequality, we obtain

$$\begin{aligned}
P(\pi, x) &\geq (1-\epsilon) \limsup_{\substack{T\to\infty \\ k=\lfloor \frac{T(1-\epsilon)}{\mathbb{E}[\mathcal{T}_2-\mathcal{T}_1]}\rfloor}} \frac{\mathbb{E}\left[\sum_{\tau_n \in (0,\mathcal{T}_k)} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]\right]}{\mathbb{E}N(0,\mathcal{T}_k)} \\
&= (1-\epsilon) \limsup_{k\to\infty} \frac{\mathbb{E}\left[\sum_{\tau_n \in (0,\mathcal{T}_k)} 1[R_n \notin \mathcal{C}_{\tau_n}^\pi]\right]}{\mathbb{E}N(0,\mathcal{T}_k)},
\end{aligned}$$

since the set $\{k : k = \lfloor T(1-\epsilon)/\mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1]\rfloor, T > 0\}$, covers all integers. We complete the proof of the lower bound by passing $\epsilon \to 0$. Upper bound uses similar arguments and we omit the details.

Next, observe the static policy s, where $\mathcal{C}_{\tau_n}^\pi \equiv \{1, 2, \ldots, x\}$ for every n. Then, due to ergodicity of the request process

$$P_s(x) \triangleq P(s,x) = \sum_{i>x} q_i.$$

Since the static policy belongs to the set of caching algorithms $\mathcal{P}_c$, we conclude that

$$P_s(x) \geq \inf_{\pi\in\mathcal{P}_c} P(\pi, x). \tag{3.6}$$

Our goal in this paper is to show that for large cache sizes x there is no caching policy that performs better, i.e., achieves long-term fault probability smaller than $P_s(x)$. This is stated in the following main result of this paper.

THEOREM 3.1. *For the request process defined in Section 2, the static policy that stores documents with the largest marginal popularities minimizes the long-term cache fault probability for large cache sizes, i.e.,*

$$\inf_{\pi\in\mathcal{P}_c} P(\pi, x) \sim P_s(x) \text{ as } x \to \infty. \tag{3.7}$$

Remarks: (i) From the examination of the following proof it is clear that the result holds for any regenerative request process that satisfies Lemma 2.1. (ii) Though asymptotically

long-term optimal, static frequency rule possesses other undesirable properties such as high complexity and non-adaptability to variations in the request patterns. However, its optimal performance presents an important benchmark for evaluating and comparing widely implemented caching policies in the Web environment.

Proof: In view of (3.6), we only need to show that $\inf_{\pi\in\mathcal{P}_c} P(\pi,x)$ is asymptotically lower bounded by $P_s(x)$ as $x\to\infty$.

For any set $\mathcal{A}$, let $|\mathcal{A}|$ denote the number of elements in $\mathcal{A}$ and $\mathcal{A}\setminus\mathcal{B}$ represent the set difference. Then, it is easy to see that the number of cache faults in $[t,u)$, $t<u$, is lower bounded by $|\mathcal{R}(t,u)\setminus\mathcal{C}_t^\pi|$ since every item that was not in the cache at time t results in at least one fault when requested for the first time; in particular, if $t=\mathcal{T}_j$, $u=\mathcal{T}_{j+1}$,

$$\sum_{\tau_n\in[\mathcal{T}_j,\mathcal{T}_{j+1})} 1[R_n\notin\mathcal{C}_{\tau_n}^\pi]\geq|\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})\setminus\mathcal{C}_{\mathcal{T}_j}^\pi|. \tag{3.8}$$

This inequality and (3.4) results in

$$P(\pi,x)\geq\limsup_{k\to\infty}\frac{1}{\mathbb{E}N(0,\mathcal{T}_k)}\sum_{j=1}^{k-1}\mathbb{E}[|\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})\setminus\mathcal{C}_{\mathcal{T}_j}^\pi|]. \tag{3.9}$$

Now, since we consider caching policies where replacement decisions depend only on the previous cache contents and requests, due to renewal structure of the request process we conclude that for every $j\geq1$ and all $i\geq1$, events $\{i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})\}$ and $\{i\in\mathcal{C}_{\mathcal{T}_j}^\pi\}$ are independent. Therefore, for every $j\geq1$,

$$\begin{aligned}&\mathbb{E}\left[|\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})\setminus\mathcal{C}_{\mathcal{T}_j}^\pi||\mathcal{C}_{\mathcal{T}_j}^\pi=\mathcal{C}\right]\\&=\sum_{i\geq1}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1}),i\notin\mathcal{C}_{\mathcal{T}_j}^\pi|\mathcal{C}_{\mathcal{T}_j}^\pi=\mathcal{C}]\\&=\sum_{i\geq1}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]1[i\notin\mathcal{C}]\\&=\sum_{i\notin\mathcal{C}}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})].\end{aligned}$$

Thus, for any $j\geq1$,

$$\mathbb{E}[|\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})\setminus\mathcal{C}_{\mathcal{T}_j}^\pi|]\geq\inf_{\mathcal{C}:|\mathcal{C}|=x}\sum_{i\notin\mathcal{C}}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]. \tag{3.10}$$

Next, we show that the cache content $[1,x]\triangleq\{1,\dots,x\}$ achieves the infimum in the previous expression for large cache sizes. This is equivalent to proving that, as $x\to\infty$,

$$\inf_{\mathcal{C}:|\mathcal{C}|=x}\sum_{i\notin\mathcal{C}}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]\gtrsim\sum_{i\notin[1,x]}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]. \tag{3.11}$$

We will justify the previous statement by showing that for any set $\mathcal{C}$ obtained from $[1,x]$ by placing documents from the set $\{x+1,\dots\}$ instead of those in $[1,x]$ can not result in $\sum_{i\notin\mathcal{C}}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]<\sum_{i\notin[1,x]}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]$ for large cache sizes x.

Lemma 2.1 implies that for an arbitrarily chosen $\epsilon>0$ there exists finite integer i_0 such that for all $i\geq i_0$

$$\begin{aligned}&(1-\epsilon)q_i\mathbb{E}[\mathcal{T}_{j+1}-\mathcal{T}_j]<\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]\\&<(1+\epsilon)q_i\mathbb{E}[\mathcal{T}_{j+1}-\mathcal{T}_j].\end{aligned} \tag{3.12}$$

Thus, using the previous expression and $q_i\downarrow0$ as $i\to\infty$, we conclude that for all $k\leq i_0$ there exists $x_0\geq i_0$ large, such that for all $i\geq x_0$

$$\min_{1\leq k\leq i_0}\mathbb{P}[k\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]>\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]. \tag{3.13}$$

Now, assume that the cache is of size $x\geq x_0$ and observe different cache contents $\mathcal{C}$ obtained from $[1,x]$ by replacing its documents with items from $\{x+1,x+2,\dots\}$. Next, using (3.13), we conclude that replacing documents enumerated with $\{1,\dots,i_0\}$ can only increase the sum on the left hand side of (3.11). On the other hand, observe cache contents $\mathcal{C}$ that are obtained from $[1,x]$ by replacing documents enumerated as $\{i_0+1,\dots,x\}$ with items from $\{x+1,\dots\}$. Then, it is easy to see that proving inequality (3.11) is equivalent to showing that $\sum_{i\in[i_0+1,x]}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]\geq\sum_{i\in\mathcal{C}\setminus[1,i_0]}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]$. Next, since for any $i\geq i_0$ inequalities (3.12) hold, we conclude

$$\begin{aligned}\frac{\sum_{i\in[i_0+1,x]}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]}{\sum_{i\in\mathcal{C}\setminus[1,i_0]}\mathbb{P}[i\in\mathcal{R}(\mathcal{T}_j,\mathcal{T}_{j+1})]}&\geq\frac{(1-\epsilon)\sum_{i\in[i_0+1,x]}q_i}{(1+\epsilon)\sum_{i\in\mathcal{C}\setminus[1,i_0]}q_i}\\&\geq\frac{1-\epsilon}{1+\epsilon},\end{aligned}$$

where the second inequality in the previous expression follows from the monotonicity of q_is. Then, by passing $\epsilon\to0$ we prove inequality (3.11).

Note that after applying the lower bound (3.11) in (3.10), in conjunction with (3.9), the renewal nature of the regenerative points and Lemma 2.1, we obtain that as $x\to\infty$

$$\inf_{\pi\in\mathcal{P}_c}P(\pi,x)\gtrsim\sum_{i\geq x}q_i, \tag{3.14}$$

which completes the proof of the theorem. ◇

4 Further extensions and concluding remarks

In this paper we prove that the static frequency rule minimizes the long term fault probability, for large cache sizes, in the presence of correlated requests. Although the frequency algorithm and its practical version the LFU policy is not commonly used in practice due to their complexity and static nature, our result provides a benchmark for evaluating

the popular heuristic schemes. In order to capture dependency in the request patterns, we use semi-Markov modulation technique, which is capable of modeling a wide range of statistical correlation, including the LRD that was repeatedly experimentally observed in Web access patterns.

There are several generalizations of our results that are worth mentioning. First, the definition of the fault probability in (3.4) can be generalized by replacing terms $1[R_n \notin \mathcal{C}^{\pi}_{\tau_n}]$ with $f(R_n)1[R_n \notin \mathcal{C}^{\pi}_{\tau_n}]$, where $f(i)$ could represent the cost of retrieving document i, e.g., the delay of fetching item i. Then, using basically the same arguments as in the proof of Theorem 3.1, one can easily show that a static policy which maximizes $\sum_{i=1}^{x} f(i)q_i$ is asymptotically optimal.

Second, in the context of documents with different sizes, in view of Section 4.1 of [10] and the arguments from the proof of Theorem 3.1, one can prove the following result:

THEOREM 4.1. *Assume that there are $D < \infty$ different document sizes. Then, if marginal request distribution is long tailed, i.e., $q_i \sim q_{i+k}$ as $i \to \infty$ for any finite integer k, the static rule that places documents with the largest ratio q_i/s_i, subject to the constraint $\sum_i s_i \le x$, is asymptotically optimal.*

Finally, in light of our recent result on the asymptotic performance of the ordinary LRU caching rule in the presence of semi-Markov modulated requests and Zipf's law marginal distributions ($q_i \sim c/i^{\alpha}$ as $i \to \infty$, $c > 0$) obtained in Theorem 3 of [9], asymptotic optimality of the static frequency rule implies that the LRU is factor $e^{\gamma} \approx 1.78$ away from the optimal (γ is the Euler constant, i.e. $\gamma \approx 0.57721\ldots$). Therefore, in view of other desirable properties, such as self-organizing nature and low complexity, the LRU rule has excellent performance even in the presence of statistically correlated requests.

5 Proof of Lemma 2.1

In this section, we prove the asymptotic relation (2.2) stated at the end of Section 2.

Note that

$$\begin{aligned}
&\mathbb{P}[i \in \mathcal{R}(\mathcal{T}_1, \mathcal{T}_2)] \\
&\quad = 1 - \mathbb{P}[i \notin \mathcal{R}_1(\mathcal{T}_1, \mathcal{T}_2), \ldots, i \notin \mathcal{R}_M(\mathcal{T}_1, \mathcal{T}_2)] \\
&\quad = \mathbb{E}[1 - (1 - q_i^{(1)})^{N_1} \ldots (1 - q_i^{(M)})^{N_M}], \qquad (5.15)
\end{aligned}$$

where $N_r \triangleq N_r(\mathcal{T}_1, \mathcal{T}_2)$, $1 \le r \le M$. Then, since $q_i \to 0$ as $i \to \infty$, it follows that $q_i^{(r)} \to 0$ as $i \to \infty$, $1 \le r \le M$. In addition, $1 - e^{-x} \le x$ for all $x \ge 0$ and for any $1 > \epsilon > 0$, there exists $x_0(\epsilon) > 0$, such that for all $0 \le x \le x_0(\epsilon)$ inequality $1 - x \ge e^{-x(1+\epsilon)}$ holds, and, therefore, for i large enough

$$\begin{aligned}
&\mathbb{E}\left[1 - e^{-(q_i^{(1)}N_1 + \cdots + q_i^{(M)}N_M)}\right] \\
&\le \mathbb{E}\left[1 - (1 - q_i^{(1)})^{N_1} \ldots (1 - q_i^{(M)})^{N_M}\right] \\
&\le \mathbb{E}\left[1 - e^{-(1+\epsilon)(q_i^{(1)}N_1 + \cdots + q_i^{(M)}N_M)}\right]. \qquad (5.16)
\end{aligned}$$

Then, since $1 - e^{-x} \le x$ for $x \ge 0$, we obtain, for i large enough,

$$\begin{aligned}
&\mathbb{E}\left[1 - e^{-(1+\epsilon)(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)})}\right] \\
&\le (1+\epsilon)\mathbb{E}\left[q_i^{(1)}N_1 + \cdots + q_i^{(M)}N_M\right]. \qquad (5.17)
\end{aligned}$$

Next, let $N \triangleq N_1 + \cdots + N_M$. Then, we show that $q_i^{(1)}\mathbb{E}N_1 + \cdots + q_i^{(M)}\mathbb{E}N_M = q_i\mathbb{E}N$. From ergodicity of the process J_t, it follows

$$\pi_r = \frac{\mathbb{E}\mathcal{T}_{1r}}{\mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1]},$$

where $\mathcal{T}_{1r}$, $1 \le r \le M$, is the length of time that J_t spends in state r during the renewal interval $(\mathcal{T}_1, \mathcal{T}_2)$ (see Section 1.6 of [4]). Finally, using $\mathbb{E}N = \mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1]$ and $\mathbb{E}N_r = \mathbb{E}\mathcal{T}_{1r}$, $1 \le r \le M$ (Poisson process of rate 1), in conjunction with (5.17), we conclude, for i large

$$\mathbb{E}\left[1 - e^{-(1+\epsilon)(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)})}\right] \le (1+\epsilon)q_i\mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1]. \qquad (5.18)$$

Next, we estimate the lower bound in (5.16). After conditioning we obtain

$$\begin{aligned}
&\mathbb{E}\left[1 - e^{-(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)})}\right] \\
&\ge \mathbb{E}\left[\left(1 - e^{-(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)})}\right) 1\left[N \le \bar{q}_i^{-\frac{1}{2}}\right]\right], \qquad (5.19)
\end{aligned}$$

where $q_i^{(r)} \le \bar{q}_i \triangleq \frac{q_i}{\min_r \pi_r} \le Hq_i$, $1 \le r \le M$. Then, note that for every $\omega \in \{N \le \bar{q}_i^{-\frac{1}{2}}\}$, $q_i^{(1)}N_1 + \cdots + q_i^{(M)}N_M \le \frac{\bar{q}_i}{\sqrt{\bar{q}_i}} = \sqrt{\bar{q}_i}$. In addition, for any $1 > \epsilon > 0$, there exists $x_0(\epsilon) > 0$, such that for all $0 \le x \le x_0(\epsilon)$ inequality $1 - e^{-x} \ge (1-\epsilon)x$ holds, and, therefore, for i large enough such that $\sqrt{\bar{q}_i} \le x_0(\epsilon)$

$$\begin{aligned}
&\mathbb{E}\left[\left(1 - e^{-(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)})}\right) 1\left[N \le \bar{q}_i^{-\frac{1}{2}}\right]\right] \\
&\ge (1-\epsilon)\mathbb{E}\left[(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)}) 1\left[N \le \bar{q}_i^{-\frac{1}{2}}\right]\right] \\
&\ge (1-\epsilon)q_i\mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1] - (1-\epsilon)\bar{q}_i\mathbb{E}[N1[N > \bar{q}_i^{-\frac{1}{2}}]].
\end{aligned}$$

Then, $\mathbb{E}[N1[N > \bar{q}_i^{-\frac{1}{2}}]] \to 0$ as $i \to \infty$ since $1/\sqrt{\bar{q}_i} \to \infty$ as $i \to \infty$ and $\mathbb{E}N < \infty$ and, therefore, in conjunction with (5.19), we obtain, as $i \to \infty$,

$$\mathbb{E}\left[1 - e^{-(N_1 q_i^{(1)} + \cdots + N_M q_i^{(M)})}\right] \gtrsim (1-\epsilon)q_i\mathbb{E}[\mathcal{T}_2 - \mathcal{T}_1].$$

Finally, after letting $\epsilon \to 0$ in the previous expression and using (5.18), we complete the proof of this lemma. $\diamond$

References

[1] M. Abrams and R. Wooster. Proxy caching that estimates edge load delays. In *Proceedings of 6th Int. World Wide Web Conf.*, Santa Clara, CA, April 1997.

[2] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliviera. Characterizing reference locality in the WWW. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[3] M. Arlitt and C. Williamson. Web server workload characteristics: The search for invariants. In *Proceedings of ACM SIGMETRICS*, May 1996.

[4] F. Baccelli and P. Brémaud. *Elements of Queueing Theory*. Springer-Verlag, 2002.

[5] O. Bahat and A. M. Makowski. Optimal replacement policies for non-uniform cache objects with optional eviction. In *Proceedings of Infocom 2003*, San Francisco, California, USA, April 2003.

[6] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE INFOCOM*, 1999.

[7] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX 1997 Annual Technical Conference*, Anaheim, California, January 1997.

[8] P. R. Jelenković and A. Radovanović. Asymptotic insensitivity of least-recently-used caching to statistical dependency. In *Proceedings of INFOCOM 2003*, San Fransisco, April 2003.

[9] P. R. Jelenković and A. Radovanović. Least-recently-used caching with dependent requests. *Theoretical Computer Science*, 326:293–327, 2004.

[10] P. R. Jelenković and A. Radovanović. Optimizing LRU for variable document sizes. *Combinatorics, Probability & Computing*, 13:1–17, 2004.

[11] Shudong Jin and Azer Bestavros. GreedyDual* Web Caching Algorithm. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[12] Shudong Jin and Azer Bestavros. Sources and characteristics of Web temporal locality. In *Proceedings of Mascots'2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Fransisco, CA, August 2000.

[13] E. G. Coffman Jr. and P. J. Denning. *Operating Systems Theory*. Prentice-Hall, 1973.

Exploring the Average Values of Boolean Functions via Asymptotics and Experimentation

Robin Pemantle*
Department of Mathematics
University of Pennsylvania
Philadelphia, PA 19104–6395
pemantle@math.upenn.edu

Mark Daniel Ward
Department of Mathematics
University of Pennsylvania
Philadelphia, PA 19104–6395
ward2@math.upenn.edu

Abstract

In recent years, there has been a great interest in studying Boolean functions by studying their analogous Boolean trees (with internal nodes labeled by Boolean gates; leaves viewed as inputs to the Boolean function). Many of these investigations consider Boolean functions of n variables and m leaves. Our study is related but has a quite different flavor.

We investigate the mean output X_n of a Boolean function defined by a complete Boolean tree of depth n. Each internal node of such a tree is labeled with a Boolean gate, via $2^n - 1$ IID fair coin flips. The value of the input at each leaf can be simply fixed at $1/2$, so the randomness of X_n derives only from the selection of the gates at the internal nodes.

For each n, there are $2^{(2^n-1)}$ possible Boolean binary trees to consider, so we cannot expect to obtain a complete description of the probability distribution of X_n for large n. Therefore, we perform a twofold investigation of the X_n, using both asymptotics and experiments. We prove that, with probability 1, $X_n \to 0$ or $X_n \to 1$. Then we directly compute the asymptotics of the first four moments of X_n. Writing $Z_n = X_n(1 - X_n)$, we also prove that $E(Z_n)$ and $E(Z_n^2)$ are both $\Theta(1/n)$. Finally, we utilize C++ and a significant amount of computation and experimentation to obtain a more descriptive understanding of X_n for small values of n (say, $n \leq 100$).

1 Introduction.

We first outline the construction of a Boolean function using a binary tree. We utilize complete binary trees T_n of depth n. The tree T_n has 2^n leaves and $2^n - 1$ internal nodes. At each of the internal nodes, we place either an AND gate or an OR gate, with probability $1/2$ each. Selection of the gates at distinct nodes is independent, so the gates are essentially chosen by IID fair coin flips. In other words, we uniformly select a vector consisting of $2^n - 1$ AND's and OR's, namely $\vec{g}_n \in \{\text{AND}, \text{OR}\}^{2^n-1}$. By labeling the internal nodes of a complete binary tree of depth n with this collection $\vec{g}_n$ of $2^n - 1$ gates, we naturally define a random Boolean function $\phi_n(\vec{g}_n) : \{0,1\}^{2^n} \to \{0,1\}$. The leaves of the tree, say $i_1, i_2, \ldots, i_{2^n}$, are considered as the inputs to the Boolean function. The output at the root of the tree is viewed as the output of the Boolean function. Thus we write

$$\phi_n(\vec{g}_n)(i_1, i_2, \ldots, i_{2^n}) \in \{0,1\}\,.$$

for each $(2^n - 1)$-tuple $\vec{g}_n$ of gates and each 2^n-tuple of inputs $i_1, i_2, \ldots, i_{2^n}$.

In this investigation, we are interested in studying the behavior of the random variable X_n, which denotes the *mean* output of $\phi_n(\vec{g}_n)$ on 2^n Boolean inputs. In other words,

$$X_n := \frac{1}{2^{2^n}} \sum_{i_1, i_2, \ldots, i_{2^n}} \phi_n(\vec{g}_n)(i_1, i_2, \ldots, i_{2^n})\,.$$

We observe that X_n is a random variable because the selection of the $2^n - 1$ gates in $\vec{g}_n$ is performed at random. Once the selection of the gates $\vec{g}_n$ is determined, then X_n is completely determined, because X_n is the average of all possible 2^{2^n} selections of inputs $i_1, i_2, \ldots, i_{2^n}$ to the Boolean tree described above. So the randomness of X_n does not stem from a random choice of the inputs $i_1, i_2, \ldots, i_{2^n}$ at all; X_n's randomness only depends on the random selection of gates at the internal nodes of the tree. Once the gates at the nodes are chosen, then we average over all possible inputs to the binary tree.

We will see in (2.6) below that X_n converges in distribution to the measure $(1/2)\delta_0 + (1/2)\delta_1$ that gives mass one half each to 0 and 1. Intuitively, there will probably be a sufficient preponderance of AND gates to drive X_n to 0 or a sufficient preponderance of OR gates to drive X_n to 1. This is not too surprising (although it does not occur in the related model of [7], where the random tree of size n has leaves at distance $\Theta(1)$ from the root). A more interesting question is how fast X_n approaches the set $\{0,1\}$, and what this reveals about the structure of the random Boolean function $\phi_n(\vec{g}_n)$.

For each selection $\vec{g}_n$ of gates, we note that $\phi_n(\vec{g}_n)$ is a function with 2^n inputs. If the inputs

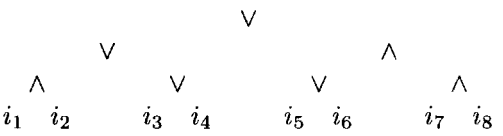
*Supported by NSF grant DMS-0401246.

$i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_{2^n}$ are all fixed, then $\phi_n(\vec{g}_n)$ is a *linear* function of i_j. Since $i_j \in \{0,1\}$ for each j, then we conclude that X_n can be computed easily, once the gates $\vec{g}_n$ are chosen, by simply taking $1/2$ as the value of each input i_j to the Boolean function $\phi_n(\vec{g}_n)$. In other words, for each selection of $\vec{g}_n$, we have

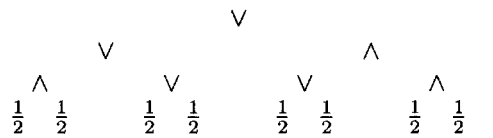
$$X_n = \phi_n(\vec{g}_n)(1/2, 1/2, \ldots, 1/2);$$

in this representation, it is perhaps easiest to see that the randomness of X_n is due to the random selection of the gates in the $(2^n - 1)$-tuple $\vec{g}_n$.

An example is useful for clarification. Consider the selection of $\vec{g}_3$ given below in this tree of depth 3:



For complete trees of depth 3, we see that X_3 denotes the mean output of a Boolean random function with gates $\vec{g}_3$. If the choice of $\vec{g}_3$ is the one given above, this results in X_3 having the value 217/256. To see this, simply evaluate the tree:



Evaluating such a tree with inputs besides the familiar $\{0,1\}$ requires a bit of explanation. The evaluation of expressions such as $i_1 \wedge i_2$ is quite easy. This expression, for instance, evaluates to 1 if both i_1 and i_2 have the value 1; otherwise, the expression evaluates to 0. Unfortunately, this evaluation is useful only for $i_1, i_2 \in \{0,1\}$. So we instead use the following equivalent interpretation (which is quite standard). We write

$$\begin{aligned} i_1 \wedge i_2 &:= i_1 i_2 \\ i_1 \vee i_2 &:= 1 - (1 - i_1)(1 - i_2). \end{aligned} \tag{1.1}$$

This interpretation has the benefit that i_1 and i_2 can be any real numbers; in particular, they can each be set to the value $1/2$.

The trees T_n are naturally embedded as increasing rooted subtrees of the infinite binary tree T. All the variables X_n may be constructed on a single probability space $(\Omega, \mathcal{F}, P)$ on which are defined variables $\{G(v) : v \in V(T)\}$ taking values AND and OR independently with probability $1/2$ each. If T' is any (possibly random) rooted subtree of T, let $\vec{g}_{T'}$ be the Boolean function with inputs at the leaves of T' defined by having the gate $G(v)$ at each internal node v of T'. Let $X_{T'}$ denote the mean of T' if all input vectors are equally likely. Thus $\phi_n(\vec{g}_n) = \vec{g}_{T_n}$ and $X_{T_n} = X_n$. Let $X_{T'}(v)$ denote the mean value of the Boolean function computed by the subtree of T' rooted at v; we may omit T' from the notation when it is understood. Another consequence of linearity is that the values $X_{T'}(v)$ are determined by a recursion. That is, if v is a leaf of T' then $X_{T'}(v) = 1/2$ by definition, while for an internal node v of T' having children w and w',

$$X_{T'}(v) = \Psi(X_{T'}(w), X_{T'}(w'), G(v)) \tag{1.2}$$

where

$$\Psi(x, y, \eta) := \begin{cases} xy & \text{if } \eta = \text{AND}, \\ 1 - (1-x)(1-y) & \text{if } \eta = \text{OR}. \end{cases}$$

Evaluating a binary tree with inputs of $1/2$ at each of the leaves yields the value of X_n for each particular selection of gates. Considering all possible selections of gates, however, is computationally infeasible for even small trees of small depth. For only the smallest values of n, say $n \leq 5$, can we possible hope to compute X_n for all of the possible choices of gates. Therefore, for medium sized values of n, say $n \leq 20$, we can readily compute the value of X_n for one particular selection of gates for one complete tree of depth n, but we cannot hope to compute X_n for every selection of gates. Therefore, we simply compute X_n on a large number of trees, but we cannot perform an exhaustive investigation of all trees and their associated Boolean functions. For large values of n, say $n \geq 30$, it becomes computationally intractable to even compute the value of X_n for one particular selection of gates on a complete binary tree of depth n. In such cases, we must discriminately choose which gates to evaluate, because we cannot possibly hope to evaluate them all.

In such cases, where we want to approximate the value of X_n on a complete tree of depth n, but where we cannot hope to evaluate all gates of the Boolean tree, we consider a growing tree. We begin simply with the root of a Boolean binary tree. At every stage, we select one leaf of the tree and change it into an internal leaf, by giving it two children and a Boolean gate. Which leaves should be transformed into parent nodes first? We utilize the concept of sensitivity of a leaf to select the next leaf to transform. The leaves that are the most sensitive, i.e., that have the largest potential effect on the evaluation of X_n, should be first. We now formalize this notion.

Let T' be a finite subtree of T and let L be a leaf of T'. Let $\vec{g}_{T',L+}$ be the Boolean function with inputs at all leaves of T' except L, computed by evaluating $g_{T'}$ with the input at L set equal to 1. Let $\vec{g}_{T',L-}$ be analogous but with the input at L set equal to 0. Let

$X_{T',L+}$ and $X_{T',L-}$ denote the respective means of the functions $\vec{g}_{T',L+}$ and $\vec{g}_{T',L-}$. Then we formally define the sensitivity of the leaf L in T' as

$$S(L,T') = X_{T',L+} - X_{T',L-}\,.$$

Another description of $S(L,T')$ is quite useful when computing the sensitivities of the leaves.

We label the root node of T' as v_0. For a leaf L at depth k in T', we write $v_0, v_1, v_2, \ldots, v_k = L$ to describe the path within the tree, from the root node, to the leaf L. For $i \geq 1$, we note that v_{i-1} has two children, namely, v_i and one other child, which we refer to as w_i. Thus v_i and w_i are distinct nodes at level i with the same parent; such nodes are frequently referred to as siblings.

LEMMA 1.1.

$$\begin{aligned} S(v_k,T') &= \prod_{i=0}^{k-1} \left([\![G(v_i) = \text{AND}]\!] X(w_{i+1}) \right. \\ &\quad \left. + [\![G(v_i) = \text{OR}]\!] (1 - X(w_{i+1})) \right) \end{aligned} \tag{1.3}$$

where the Iverson notation $[\![A]\!]$ is 1 if event A holds and is 0 otherwise.

Proof. Induct on T'. When T' is the single vertex $v_0 = v_k = L$ then by definition $g_{T',L+} \equiv 1$ and $g_{T',L-} \equiv 0$, so $S(L,T') = 1$ which is equal to the empty product in (1.3). If not, then assume the lemma is true for the subtree $T'(v_1)$:

$$\begin{aligned} S(L,T'(v_1)) &= \prod_{i=1}^{k-1} \left([\![G(v_i) = \text{AND}]\!] X(w_{i+1}) \right. \\ &\quad \left. + [\![G(v_i) = \text{OR}]\!] (1 - X(w_{i+1})) \right). \end{aligned}$$

By definition, $S(L,T')$ is the difference of the means of $\vec{g}_{T',L+}$ and $\vec{g}_{T',L-}$. By independence, the mean of $\vec{g}_{T',L+}$ is $\vec{g}_{T'(v_1),L+} X(w_1)$ if $G(v_0) = \text{AND}$, and $1 - (1 - \vec{g}_{T'(v_1),L+})(1 - X(w_1)) = \vec{g}_{T'(v_1),L+}(1 - X(w_1)) + X(w_1)$ if $G(v_0) = \text{OR}$. Subtracting the analogous representation of the mean of $\vec{g}_{T',L-}$ (just replace each $L+$ in the previous sentence with $L-$), it follows that $S(L,T') = S(L,T'(v_1)) \cdot \xi$, where $\xi = X(w_1)$ if $G(v_0) = \text{AND}$ and $1 - X(w_1)$ if $G(v_0) = \text{OR}$, completing the induction.

We developed a C++ program to investigate the growth of Boolean binary trees, using the sensitivity of the leaves as a guide for which subtrees to explore first. The program is completely adaptive, according to the sensitivities of the leaves. At each stage of the execution of the program, the most sensitive leaf is chosen, using the definition of sensitivity described above. If several leaves have the same sensitivity, the program selects one of the candidate leaves uniformly at random; sometimes the candidate leaves are at different levels, so this is an important subtlety in the implementation of the program. Once a leaf L is selected to be updated, we consider the path $v_0, v_1, \ldots, v_k = L$ from the root of the tree to the leaf. Only the X-values $X(v_0), X(v_1), \ldots, X(v_k)$ must be updated; this is extremely efficient in terms of the computation required, because at most n nodes are found on the path from the root to the leaf. The sensitivities of every leaf in the tree must be updated afterwards. We note that if $v'_0, v'_1, \ldots, v'_k = L'$ denotes another leaf, then the sensitivity of L' is exactly

$$\begin{aligned} &\prod_{i=0}^{k-1} \left([\![G(v'_i) = \text{AND}]\!] X(w'_{i+1}) \right. \\ &\quad \left. + [\![G(v'_i) = \text{OR}]\!] (1 - X(w'_{i+1})) \right). \end{aligned} \tag{1.4}$$

A subtlety here is the observation that only $X(w_0), X(w_1), \ldots, X(w_j)$ were changed at this stage in the growth of the tree. For some value of j (which is, with high probability, quite small), we note that L and L' have common ancestors $v_i = v'_i$ for all $i \leq j$, but $v_i \neq v'_i$ for all $i > j$. Thus $w_i = w'_i$ for all $i \leq j$, but $w_i \neq w'_i$ for all $i > j$. Therefore, only the X-values $X(w'_1), \ldots, X(w'_j)$ need to be updated in the sensitivity of L', as written in (1.4). In other words, the only $X(w'_i)$-values that need updating are those for which $v_i = v'_i$ is a common ancestor of both L and L'.

We wrote several C++ programs to perform the computations in this project. Some sample output from the programs is presented graphically at the end of this paper.

We have computed millions and millions of values of X_n for various values of n. For instance, when $n = 15$, we are able to compute approximately 30 values of X_n per second on a 1.42 GHz Power Macintosh G4 machine. We have built a large database that archives all of the output from these investigations. It has grown so large that it is unwieldy to distribute all of it publicly on the Internet, but we summarize some of the results of our computations at the end of this report.

2 Main results

We were inspired to pursue an analysis of X_n because of Gardy and Woods' intriguing study [7], in which various measures on Boolean functions are analyzed. Gardy and Woods consider trees chosen uniformly among all sub-binary trees with n leaves; they also place randomly assigned logical gates at the internal nodes. We note that a uniformly chosen tree with n leaves is stringy. The typical random function produced in this way is therefore dominated by the $\Theta(1)$ many inputs at leaves

of distance $\Theta(1)$ from the root. Their model is natural for some purposes, but we are interested in considering the model in which, as $n \to \infty$, the distance from the root to the boundary goes to infinity. For this reason, we consider the simplest such model, namely, the complete binary tree. The typical behavior of a random Boolean function produced by a complete binary tree turns out to be interesting but in some ways elusive.

Besides the analysis contained in [7], we note that many attributes of Boolean functions, binary Boolean trees, and tree recurrences have been analyzed recently. For instance, consider [1], [2], [3], [6], [8], [10], [11], [12], and [13]. These papers constitute a starting point from which readers can delve further into the literature.

We recall that X_n is the mean output of a Boolean function defined by a complete Boolean tree of depth n. In this report, we prove the following facts about X_n.

THEOREM 2.1. *The sequence $\{X_n\}$ is a Martingale. With probability 1, the limit $\lim_{n\to X_n}$ exists and is either 0 or 1. The moments of X_n may all be computed recursively. In particular, the first four moments of X_n are*

$$
\begin{aligned}
E(X_n) &= \frac{1}{2} \\
E(X_n^2) &= \frac{1}{2} - \frac{1}{n} + O\left(\frac{\log n}{n^2}\right) \\
E(X_n^3) &= \frac{1}{2} - \frac{3}{2n} + O\left(\frac{\log n}{n^2}\right) \\
E(X_n^4) &= \frac{1}{2} + \frac{\alpha - 2}{n} + O\left(\frac{\log n}{n^2}\right) \qquad (2.5)
\end{aligned}
$$

where $\alpha = \frac{\sqrt{7}-1}{2}$. To understand the rate at which a variable with symmetric distribution on $[0,1]$ converges to $\{0,1\}$, it is natural to analyze the moments of $Z_n := X_n(1-X_n)$. We have

$$E(Z_n) = \frac{1}{n + O(\log n)}, \qquad (2.6)$$

$$E(Z_n^2) \sim \frac{\alpha}{n}. \qquad (2.7)$$

It follows that, for some $a > 0$, $P(Z_n \geq a) = \Theta(1/n)$.

Left open is whether the rest of the time Z_n is typically of order $1/n$ or of some smaller order.

Just as the right $1/n$-tail of Z_n is larger than one might initially expect, it is also not hard to show that the left $1/n$-tail of Z_n is quite small.

PROPOSITION 2.1. *There are $c, c' > 0$ such that $P(Z_n < \exp(-cn^2)) > c'/n$.*

We believe in fact that the distribution of $\log Z_n$ is spread over an interval of increasing size as $n \to \infty$. Perhaps, for instance, $\sqrt{\log Z_n}/n$ has a nondegenerate distributional limit.

We point out that there are issues in effective simulation that are bound up with theoretical analyses of the problem. In particular, exact simulation of Z_n (the study of Z_n and X_n is basically interchangeable) requires a time that is exponential in n. Thus, for example, we cannot even obtain one sample of Z_{100} (just one sample of Z_{100} requires the generation of $2^{100} - 1$ Boolean gates). We can only hope to obtain exact samples of Z_n for medium-sized n. For larger n, say $n \geq 30$, our remedy is an extensive experimental analysis of Z_n by repeatedly approximating Z_n; we do this by exploring only nodes of the tree that one expects to have high impact on the value of Z_n. At each stage in the growth of the tree, there is a well defined most sensitive remaining node (there may be ties); therefore, one may define a greedy search algorithm which always looks next at this node. Revealing the gate will reduce the variance by the most. If one can then compute how close one is to X_n then one will know how far to go in order to simulate a pick from X_n with the desired precision. If, further, one can analyze the growth of the exploration tree, then one will know how long it takes to simulate X_n, and this will have implications directly on the distribution of X_n. For example, if X_n is typically well approximated by a tree of depth $m < n$, then the distributions of X_n and X_m are close and, if $m = o(n)$, this precludes a limit law with n in the denominator.

Ample data generated by our various **C++** programs for studying the behavior of X_n when n is small (say, $n \leq 100$) can be obtained from the authors. Our files of data are too large to distribute on the internet at present (we have hundreds of megabytes of files, containing millions of samples of various X_n).

At the present time, it suffices to present a few graphs of sample data concerning X_n at the end of the paper. In particular, we give plots of values related to possible limit laws for X_{15} and X_{20}, using numerical data from millions of samples of X_{15} and X_{20}.

3 Analysis and Proofs.

We establish the fact that $\{X_n\}$ is a martingale. We also derive the first four moments of X_n. Using a similar methodology, one can set up similar recurrences and use analogous arguments to derive any of the moments of X_n.

By a *sampling scheme* we mean a rule for producing a sequence of vertices $y_1, y_2, \ldots$ such that for each k the vertices $\{y_1, \ldots, y_k\}$ span a rooted subtree W_k of T and each y_{k+1} is a function of $G(y_1), \ldots, G(y_k)$. Associated with each such rule is the σ-field $\mathcal{F}^{(k)} := \sigma(G(y_1), \ldots, G(y_k))$ of the first k gates one looks at.

LEMMA 3.1. *The sequence $\{X_{W_k}\}$ is a martingale with respect to $\{\mathcal{F}^{(k)}\}$. In particular, it follows from the breadth-first sampling scheme that the sequence $\{X_n\}$ is a martingale with respect to the filtration $\{\mathcal{F}_n\} := \sigma(G(v) : v \text{ is an internal node of } T_n)$.*

Proof. To see that $\{X_{W_k}\}$ is a martingale, observe that, conditional on $\mathcal{F}^{(k)}$, the vertex y_{k+1} is known and its output in the tree W_{k+1} has mean $1/2$. By linearity, $E(X_{W_{k+1}}|\mathcal{F}^{(k)})$ is the mean output at the root of W_k, with the input at y_{k+1} set to $1/2$; however, the mean may be computed by setting all the inputs to $1/2$, so this is equal to X_{W_k}.

COROLLARY 3.1. *With probability 1, $\lim_{n\to\infty} X_n$ exists.*

Proof. Since $0 < X_n < 1$ for each n, we have $E(|X_n|) \leq 1$ for all n. By Lemma 3.1, we know that $\{X_n\}$ is a martingale. Thus, the corollary follows immediately by the Martingale Convergence Theorem (see [4], [5]).

We now evaluate moments of X_n.

LEMMA 3.2.

$$E(X_n) = 1/2\,.$$

Proof. Reversing each gate and each input reverses the output but preserves the measure on functions of $\vec{g}_n$. Thus $1 - X_n$ and X_n have the same distribution.

We use the following Lemma to aid in the proof of Theorem 3.2. If we define $Z_n = X_n(1 - X_n)$, then we make the following observations.

LEMMA 3.3.

1. *$E(X_n^2)$ increases to the limiting value of $1/2$.*
2. *The rate of convergence is given by*

$$E(X_n^2) = \frac{1}{2} - \frac{1}{n} + O\left(\frac{\log n}{n^2}\right).$$

3. *Asymptotics for Z_n are given by*

$$E(Z_n) := E(X_n(1 - X_n)) = \frac{1}{n} + O\left(\frac{\log n}{n^2}\right).$$

Proof. That $E(X_n^2)$ increases follows from $\{X_n\}$ being a martingale. To find the limit, we establish a recurrence for $E(X_n^2)$. When computing X_{n+1}, we let $X_n' := X_{T_{n+1}}(v_1)$ and $X_n'' := X_{T_{n+1}}(w_1)$ denote the outputs of the Boolean functions for the left and right subtrees of the root node; note that X_n' and X_n'' are independent, and each is distributed as X_n. Taking expectations in (1.2), we have

$$\begin{aligned} E(X_{n+1}^2) &= \frac{1}{2}E((X_n'X_n'')^2) \\ &\quad + \frac{1}{2}E((1 - (1 - X_n')(1 - X_n''))^2)\,. \end{aligned} \tag{3.8}$$

The variables X_n', X_n'', $1 - X_n'$, and $1 - X_n''$ all have the same distribution. Together with independence of X_n' and X_n'' this gives

$$E(X_{n+1}^2) = E(X_n^2)^2 + \frac{1}{4}\,. \tag{3.9}$$

Since $E(X_n^2)$ increases and is bounded above by 1, then a limiting value exists; we take a limit on both sides of (3.9) to obtain

$$\lim_{n\to\infty} E(X_{n+1}^2) = (\lim_{n\to\infty} E(X_n^2))^2 + 1/4\,. \tag{3.10}$$

Thus $\lim_{n\to\infty} E(X_n^2) = 1/2$, which completes the proof of the first statement of the Lemma.

Now we observe

$$\begin{aligned} E(Z_n) &= E(X_n(1 - X_n)) \\ &= E(X_n) - E(X_n^2) \\ &= \frac{1}{2} - E(X_n^2)\,. \end{aligned} \tag{3.11}$$

Thus $E(X_n^2) = \frac{1}{2} - E(Z_n)$. From (3.9), it follows immediately that

$$\frac{1}{2} - E(Z_{n+1}) = \left(\frac{1}{2} - E(Z_n)\right)^2 + 1/4;$$

after simplifying, we obtain

$$E(Z_{n+1}) = E(Z_n) - E(Z_n)^2\,.$$

For ease of notation, we write $a_n = E(Z_n)$. So we have $a_{n+1} = a_n - a_n^2$. Then we write $b_n = 1/a_n$, and we compute

$$\begin{aligned} b_{n+1} &= \frac{1}{a_n - a_n^2} \\ &= \frac{b_n^2}{b_n - 1} \\ &= b_n + 1 + \frac{1}{b_n - 1}\,. \end{aligned} \tag{3.12}$$

Iterating this yields

$$b_{n+1} = b_1 + n + \sum_{k=1}^{n} \frac{1}{b_k - 1}\,. \tag{3.13}$$

From (3.13), we observe that $b_{n+1} > n$, so $b_k > k-1$ for all k. Thus, the summation in (3.13) can be bounded by writing

$$\begin{aligned}\sum_{k=1}^{n} \frac{1}{b_k - 1} &= \frac{1}{b_1 - 1} + \frac{1}{b_2 - 1} + \sum_{k=3}^{n} \frac{1}{b_k - 1} \\ &\leq \frac{1}{\frac{16}{3} - 1} + \frac{1}{\frac{256}{39} - 1} + \sum_{k=3}^{n} \frac{1}{(k-1) - 1} \\ &= O(\log n)\,.\end{aligned}$$

Returning to (3.13), we conclude that

$$b_n = n + O(\log n)\,.$$

Again using (3.13), it follows that

$$\begin{aligned}b_{n+1} &= b_1 + n + \sum_{k=1}^{n} \frac{1}{n + O(\log n)} \\ &= b_n = n + \log n + O(1)\end{aligned}$$

and we conclude that

$$E(Z_n) = a_n = \frac{1}{n} + O\left(\frac{\log n}{n^2}\right)\,.$$

This proves the final sentence of the lemma. All that remains to show is $E(X_n^2) = 1/2 - 1/n + O\left(\frac{\log n}{n^2}\right)$, but this follows immediately from $E(X_n^2) = 1/2 - E(Z_n)$.

We recall from Corollary 3.1 that $\lim_{n\to\infty} X_n$ exists with probability 1. So we define $X := \lim_{n\to\infty} X_n$. Using Lemma 3.3, we have the following result about the limiting behavior of X_n.

COROLLARY 3.2.

$$P(X = 0) = P(X = 1) = \frac{1}{2}\,.$$

Proof. By bounded convergence, we have $E(X^2) = \lim_{n\to\infty} E(X_n^2) = 1/2$. Since $X \in [0,1]$ is symmetric around $1/2$, this implies $X = 0$ or 1 with probability $1/2$ each.

The next two lemmas compute the remaining moments in Theorem 2.1.

LEMMA 3.4. *The third moment of X_n is given by*

$$E(X_n^3) = \frac{1}{2} - \frac{3}{2n} + O\left(\frac{\log n}{n^2}\right)\,.$$

Proof. As in Lemma 3.3, we establish a recurrence for $E(X_n^3)$. When computing X_{n+1}, we again write X_n' and X_n'' to denote the output of the Boolean functions for the left and right subtrees of the root node, which are independent. Then we compute

$$\begin{aligned}E(X_{n+1}^3) &= \frac{1}{2}E((X_n' X_n'')^3) \\ &\quad + \frac{1}{2}E((1 - (1 - X_n')(1 - X_n''))^3)\,. \qquad (3.14)\end{aligned}$$

We once again use the fact that X_n', X_n'', $1 - X_n'$, and $1 - X_n''$ share a common distribution. Thus,

$$\begin{aligned}E(X_{n+1}^3) &= \frac{1}{2}E(X_n^3)^2 + \frac{1}{2} - \frac{3}{2}E(X_n)^2 \\ &\quad + \frac{3}{2}E(X_n^2)^2 - \frac{1}{2}E(X_n^3)^2 \\ &= \frac{3}{2}E(X_n^2)^2 + \frac{1}{8}\,. \qquad (3.15)\end{aligned}$$

Recall from (3.9) that

$$E(X_{n+1}^2) = E(X_n^2)^2 + \frac{1}{4}\,. \qquad (3.16)$$

Plugging this result into (3.15) yields

$$\begin{aligned}E(X_{n+1}^3) &= \frac{3}{2}\left(E(X_{n+1}^2) - \frac{1}{4}\right) + \frac{1}{8} \\ &= \frac{3}{2}E(X_{n+1}^2) - \frac{1}{4} \qquad (3.17)\end{aligned}$$

and by Lemma 3.3, we conclude that

$$E(X_n^3) = \frac{1}{8} - \frac{3}{4n} + O\left(\frac{\log n}{n^2}\right)\,. \qquad (3.18)$$

This establishes the lemma.

Using the lemmas above, we now establish the following asymptotics for $E(Z_n^2)$.

LEMMA 3.5. *The second moment of Z_n^2 decays as $E(Z_n^2) \sim \frac{\alpha}{n}$ where $\alpha = \frac{\sqrt{7}-1}{2} \approx .82$.*

Proof. As in several of the above lemmas, we observe that

$$\begin{aligned}E(X_{n+1}^4) &= \frac{1}{2}E((X_n' X_n'')^4) \\ &\quad + \frac{1}{2}E((1 - (1 - X_n')(1 - X_n''))^4)\,. \qquad (3.19)\end{aligned}$$

Simplifying via the same method as in the lemmas and using the results established in Lemmas 3.3 and 3.4, it follows that

$$E(X_{n+1}^4) = E(X_n^4)^2 - \frac{3}{2}E(X_n^2)^2 + \frac{3}{2}E(X_n^2) - \frac{1}{8}\,. \qquad (3.20)$$

For ease of notation, we define

$$h_n := \frac{1}{2} - E(X_n^4) \qquad (3.21)$$

and

$$d_n := \frac{1}{2} - E(X_n^2). \tag{3.22}$$

It follows from (3.20) that

$$h_n = h_{n-1} - h_{n-1}^2 + \frac{3}{2}d_n^2. \tag{3.23}$$

The proof of the lemma is finished by the following identity.

LEMMA 3.6. *If $d_n \sim 1/n$ and h_n are positive numbers satisfying (3.23) then $h_n \sim \alpha/n$ for $\alpha = (\sqrt{7}-1)/2$.*

Proof. Let $u_n = nh_n$. The recursion (3.23) becomes

$$\begin{aligned} u_n &= \frac{n}{n-1}u_{n-1} - \frac{n}{(n-1)^2}u_{n-1}^2 + \frac{3/2+o(1)}{n} \\ &= u_{n-1} + \frac{\left[u_{n-1} - u_{n-1}^2 + \frac{3}{2} + O\left(\frac{u_{n-1}}{n}\right)\right]}{n-1}. \end{aligned}$$

One may easily verify that $u_n/n \to 0$, which implies

$$u_n - u_{n-1} = n^{-1}[f(u_{n-1}) + o(1)]. \tag{3.24}$$

Checking that $x - x^2 + 3/2$ is positive on $(0,\alpha)$ and negative on (α,∞), we then see that $u_{n+1} > u_n$ when $u_n \in (0, \alpha - o(1))$ and $u_{n+1} < u_n$ when $u_n \in (\alpha + o(1), \infty)$, so u_n converges. Convergence to something other than α is not possible because in that case eventually $|f(u_n)| > \varepsilon$ and divergence of the harmonic sum in (3.24) would contradict convergence of u_n.

We complete the moment computations with:

COROLLARY 3.3.

$$E(X_n^4) = \frac{1}{2} + \frac{\alpha - 2}{n} + O\left(\frac{\log n}{n^2}\right)$$

where $\alpha = \frac{\sqrt{7}-1}{2}$.

Proof. We note that $E(X_n^4) = E(Z_n^2) - E(X_n^2) + 2E(X_n^3)$, and then the corollary follows immediately from Lemmas 3.3, 3.4, and 3.5.

Finally, to derive the last statement in Theorem 2.1, observe that greatest possible second moment for a random variable in $[0,a]$ with mean μ is $a\mu$. Thus,

$$\begin{aligned} \frac{\alpha + o(1)}{n} &= E(Z_n^2) \\ &= E(Z_n^2)[\![Z_n \ge a]\!] + E(Z_n[\![Z_n < a]\!])^2 \\ &\le (\sup_n Z_n^2)P(Z_n \ge a) + aEZ_n[\![Z_n < a]\!] \\ &\le \frac{1}{16}P(Z_n \ge a) + \frac{a + o(1)}{n}, \end{aligned}$$

and hence

$$P(Z_n > a) \ge 16\frac{\alpha - a + o(1)}{n}.$$

This finishes the proof of Theorem 2.1 for any value $a < \alpha$.

Now we prove Proposition 2.1, namely, there are $c, c' > 0$ such that $P(Z_n < \exp(-cn^2)) > c'/n$.

Proof of Proposition 2.1. Let $T' \subseteq T$ be the subtree whose vertices are the vertices of T that can be reached from the root by a path not containing any OR gate (if there is an OR gate at the root then T' is empty). Denote the size and depth of T' by $|T'|$ and $d(T')$, respectively. The event A_n that $d(T') < n$ and $|T'| > n^2$ is well known to have probability asymptotic to Cn^{-1} for some constant $C > 0$; this follows, for example, from the convergence of n times the law of the path that circumnavigates the tree to Brownian excursion measure [9].

Let $\mathcal{F} = \sigma(T')$ be the information contained in the value of the random tree T'. Let S_n be the set of vertices in T_n adjacent to T' but not in T'. Since a subtree of T with k vertices has $k+1$ neighbors, we see that on A_n, the set S_n satisfies $s := |S_n| > n^2$. Conditional on $\mathcal{F}$, the s subtrees from vertices in S_n are independent and distributed exactly as the gates of T except that the root is always an OR. Consequently, on A_n, the output at any vertex of S_n has conditional mean $3/4$ given $\mathcal{F}$, since all we know about the gates of this subtree is that there is an OR at the root. Furthermore, $E(X_n \,|\, \mathcal{F}) = (3/4)^s$ on the event A_n, since the root outputs a 1 if and only if all vertices in S_n output a 1. The result now follows from $P(A_n) \sim Cn^{-1}$ and $E(X_n \,|\, \mathcal{F}) < (3/4)^{n^2}$ on A_n, with any $c < \log(4/3)$ and $c' < C$.

4 Further Discussion and Experimental Data

The purpose of the sensitivity-first sampling scheme is to sample approximately from the distribution of X_n by sampling the variable $Y_k := X_{W_k}$ in the sampling scheme $y_1, y_2, \ldots$ which always chooses the leaf of greatest sensitivity. More precisely, when trying to sample from X_n, we partition the leaves of W_k into $A_k \cup B_k$ where A_k is the set of leaves of W_k at level n and B_k is the set of leaves of W_k at levels less than n; our sampling scheme then designates y_{k+1} to maximize $S(y_{k+1}, W_k)$ over $y_{k+1} \in B_k$. This sampling scheme will halt when $k = 2^n$ and produce $Y_k = X_n$, but our hope is that Y_k is close to X_n for k much less than n, for example a polynomial in $\log n$.

We cannot prove this, though we have some shaky evidence. The reason the evidence is shaky is that we have tabulated how great k must be in order to satisfy

criteria appearing to give Y_k near to X_n but cannot prove that Y_k is actually close to X_n. We conclude the theoretical discussion with some results giving bounds on the distance from Y_k to X_n.

We will be applying these bounds knowing $\mathcal{F}^{(k)}$ but not $\mathcal{F}_n$, so we want bounds measurable with respect to $\mathcal{F}^{(k)}$. A crude upper bound is:

PROPOSITION 4.1.

$$E(|Y_k - X_n| \,|\, \mathcal{F}^{(k)}) \le \sum_{y \in B_k} S(y, T_k)\,. \tag{4.25}$$

Proof. Let X_n^* be the (random) mean of the random Boolean function obtained from $\vec{g}_n$ by fixing inputs (at random) at vertices in B_k. Then X_n is a conditional expectation of X_n^* so, conditional on $\mathcal{F}^{(k)}$, we know that $E(|X_n - Y_k| \,|\, \mathcal{F}^{(k)}) \le E(|X_n^* - Y_k| \,|\, \mathcal{F}^{(k)})$.

For $y \in B_k$, fixing the inputs at leaves of T_n below y as independent fair coin flips produces an output at y; for purposes of computing X_n^*, we may as well simply fix that output, which will be a 0 half the time and a 1 half the time, independent of nodes not in $T(y)$. Thus another way to compute X_n^* is to fix inputs (independent fair coin flips) at all the leaves of T_k not already at level n. Enumerate these as $z_1, \ldots, z_r$, and let M_j denote the mean of the Boolean function obtained from $\vec{g}_{T_k}$ by fixing inputs at $z_1, \ldots, z_k$. The triangle inequality gives

$$E(|X_n^* - Y_k| \,|\, \mathcal{F}^{(k)}) \le \sum_{j=1}^{r} E(|M_j - M_{j-1}| \,|\, \mathcal{F}^{(k)})$$

which is equal to $\sum_{j=1}^{r} (1/2) S(z_j, T_k)$ because the sensitivity at z_j as $z_1, \ldots, z_{j-1}$ is revealed is itself a martingale. This proves (4.25).

Based on this, a reasonable time to halt the algorithm and output Y_k would be when the right-hand side of (4.25) is much smaller than the same sum over leaves of T_k that are at level n. Unfortunately, based on our preliminary experiments growing trees with **C++** according to the sensitivities of the leaves, this does not seem to happen until too much of T_n is explored to be efficient. However, the L^1 sum in (4.25) is probably an overestimate of how much Y_k will change on the way to evaluating X_n. In particular, since $\{Y_k\}$ is a martingale, one might expect that summing in L^2 yields sharper estimates.

The incremental variance of the martingale $\{Y_k\}$ is given by the squared sensitivities:

$$E((Y_{k+1} - Y_k)^2 \,|\, \mathcal{F}^{(k)}) = (1/4) S(y_{n+1}, T_k)^2\,.$$

We would like to conclude that the L^2 difference between Y_k and X_n is given by the sum of $S(y, T_k)^2$ over leaves y of T_k that are not already at level n, but the problem is that, with z_j, W_j as above, it is no longer true that $E(W_{j+1} - W_j)^2 = (1/4) S(z_j, T_k)^2$. This is because the sensitivity at z_j is a submartingale as the gates at $z_1, \ldots, z_{j-1}$ are revealed. We conjecture, however, that

$$E((Y_k - X_n)^2 \,|\, \mathcal{F}^{(k)}) \le C \sum_{y} S(y, T_k)^2$$

for some constant C.

Since we cannot rigorously prove a fast stopping rule that yields reasonable sample values of X_n, we have reliable samples of X_n only for $n \le 20$. Various graphs concerning the distribution of X_{15} and X_{20} are given below.

If we write $p = P(X_{15} \le x)$, then the following chart gives the values of p and the analogous x value. The data is based upon four million samples of X_{15}.

We also give similar data for X_{20}, based on 800,000 samples.

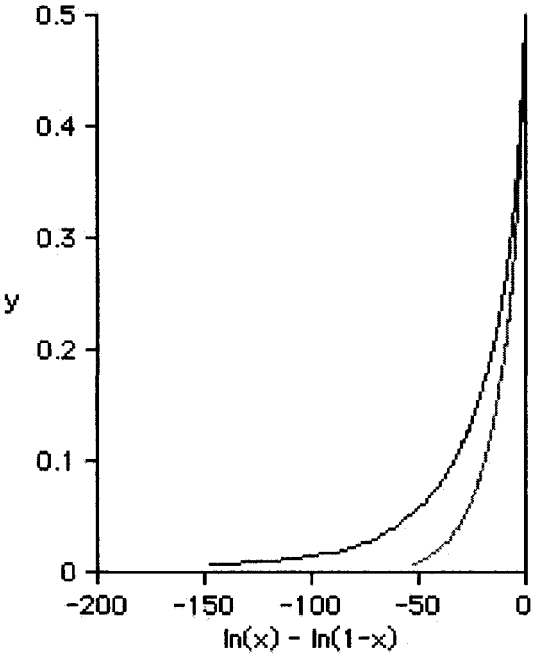
We emphasize that each sample of X_{15} and X_{20} was produced by computing a complete Boolean binary tree of depth 15 and 20, respectively.

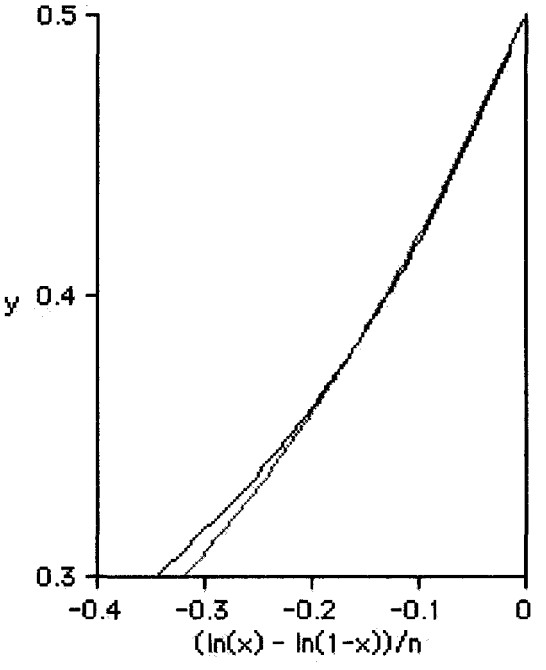
When simulating X_n for larger n, such as 100, we used an interactive **C++** program that allows the user to sample values from X_{100}, for instance, with interactions concerning when to stop the simulation. The **C++** program is trained to stop the simulation itself if it detects that the sensitivities of the leaf nodes, collectively, are sufficiently small. The stopping condition is easily modified by the user, so we continue to experiment with a variety of stopping conditions.

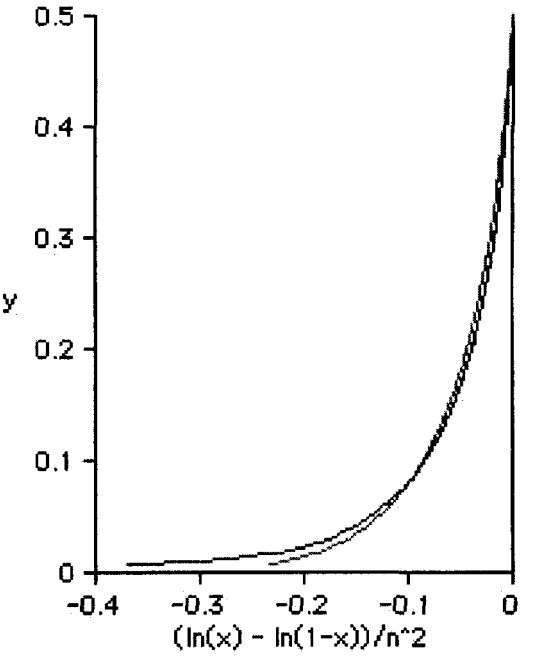
The **C++** program has several other features. For instance, it lets us visualize the data by examining the profile as the tree grows. The evolution of the profile as the most sensitive nodes are selected within the tree is an intriguing phenomenon. Besides further studying the profile, we also plan to continue investigating stopping criteria for the growth of large Boolean binary tree when simulating X_n for large n, for example, $n = 100$.

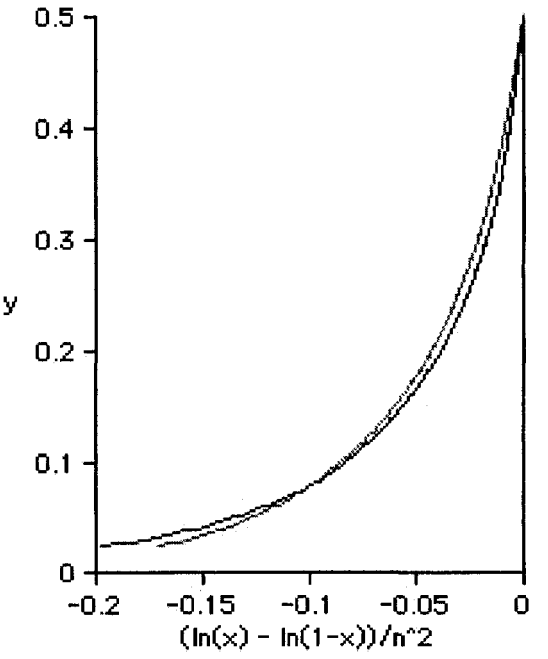
All of the graphs below are for the pairs (x, y) where $y = P(X_{15} \leq x)$ (based on 4,000,000 samples of X_{15}) and $y = P(X_{20} \leq x)$ (based on 800,000 samples of X_{20}).

We rescale the x-axis in a variety of ways.









Acknowledgments

We appreciate the input of Svante Janson, who simultaneously derived several of the observations presented here. We also acknowledge Bob Sedgewick's insightful advice about using randomization in the implementation of data structures (*Finding Paths in Graphs*, A of A 2005).

References

[1] C. Banderier, M. Bousquet-Mélou, P. Flajolet A. Denise, D. Gardy, and D. Gouyou-Beauchamps. Generating functions for generating trees. *Discrete Mathematics*, 246:29–55, 2002.

[2] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288:21–43, 2002.

[3] B. Chauvin, P. Flajolet, D. Gardy, and B. Gittenberger. And/or tree revisited. *Combinatorics, Probability and Computing*, 13(4–5):475–497, 2004.

[4] R. Durrett. *Probability: Theory and Examples.* Duxbury, Belmont, CA, 3rd edition, 2005.

[5] W. Feller. *An Introduction to Probability Theory and Its Applications.* Wiley, New York, 1968, 1971.

[6] J. Fill, P. Flajolet, and N. Kapur. Singularity analysis, hadamard products, and tree recurrences. *Journal of Computational and Applied Mathematics*, 174:271–313, February 2005.

[7] D. Gardy and A. Woods. And/or tree probabilities of boolean functions. In Conrado Martínez, editor, *2005 International Conference on Analysis of Algorithms*, volume AD of *DMTCS Proceedings*, pages 139–146. Discrete Mathematics and Theoretical Computer Science, 2005.

[8] H. Lefmann and P. Savický. Some typical properties of large and/or Boolean formulas. *Random Structures and Algorithms*, 10:337–351, 1997.

[9] J. Neveu and J. Pitman. The branching processes in a brownian excursion. In *Séminaire de Probabilités, XXIII*, volume 1372 of *Lecture Notes in Mathematics*, pages 248–257. Springer-Verlag, New York, 1989.

[10] P. Savický. Bent functions and random Boolean formulas. *Discrete Mathematics*, 147:211–237, 1995.

[11] P. Savický. Complexity and probability of some Boolean formulas. *Combinatorics, Probability and Computing*, 7(4):451–463, 1998.

[12] P. Savický and A. Woods. The number of Boolean functions computed by formulas of a given size. *Random Structures and Algorithms*, 13(3–4):349–382, 1998.

[13] I. Wegener. *The complexity of Boolean functions.* Teubner, Stuttgart, 1987.

Permanents of Circulants: a Transfer Matrix Approach* (Extended Abstract)

Mordecai J. Golin Yiu Cho Leung Yajun Wang

Abstract

Calculating the permanent of a $(0,1)$ matrix is a $\#P$-complete problem but there are some classes of *structured* matrices for which the permanent is calculable in polynomial time. The most well-known example is the *fixed-jump* $(0,1)$ circulant matrix which, using algebraic techniques, was shown by Minc to satisfy a constant-coefficient fixed-order recurrence relation.

In this note we show how, by interpreting the problem as calculating the number of cycle-covers in a directed circulant *graph,* it is straightforward to reprove Minc's result using combinatorial methods. This is a two step process: the first step is to show that the cycle-covers of directed circulant graphs can be evaluated using a *transfer matrix* argument. The second is to show that the associated transfer matrices, while very large, actually have much smaller characteristic polynomials than would a-priori be expected.

An important consequence of this new viewpoint is that, in combination with a new recursive decomposition of circulant-graphs, it permits extending Minc's result to calculating the permanent of the much larger class of circulant matrices with *non-fixed* (but linear) jumps.

1 Introduction

DEFINITION 1.1. *Let $A = (a_{i,j})$ be an $n \times n$ matrix. Let S_n be the set of permutations of the integers $[1,\ldots,n]$. The* permanent *of A is*

$$\mathrm{Perm}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)} \quad \textit{where} \quad \pi = [\pi(1),\ldots,\pi(n)].$$

If A is a $(0,1)$ matrix, then A can be interpreted as the adjacency matrix of some directed graph G and $\mathrm{Perm}(A)$ is the number of *directed cycle-covers* in G, where a directed cycle-cover is a collection of disjoint cycles that cover all of the vertices in the graph. Alternatively, A can be interpreted as the adjacency matrix of a bipartite graph $\bar{G}$, in which case $\mathrm{Perm}(A)$ is the number of *perfect-matchings* in $\bar{G}$. The permanent is a classic well-studied combinatorial object (see the book and later survey by Minc[13, 16]).

Calculating the permanent of a $(0,1)$ matrix is a $\#P$-Complete problem [19] even when A is restricted to have only 3 non-zero entries per row [8]. The best known algorithm for calculating a general permanent is a straightforward inclusion-exclusion technique due to Ryser [13] running in $\Theta(n2^n)$ time and polynomial space. By allowing super-polynomial space, Bax and Franklin [1] developed a slightly faster (although still exponential) algorithm for the $(0,1)$ case. We point out, in another direction, that just recently, Jerrum, Sinclair and Vigoda [11] developed a fully polynomial approximation scheme for *approximating* the permanent of nonnegative matrices.

On the other hand, for certain special structured classes of matrices one can *exactly* calculate the permanent in "polynomial time". The most studied example of such a class is probably the *circulant matrices,* which, as discussed in [7], can be thought of as the borderline between the easy and hard cases.

An $n \times n$ circulant matrix $A = (a_{i,j})$ is defined by specifying its first row; the $(i+1)^{\mathrm{st}}$ row is a cyclic shift i units to the right of the first row, i.e., $a_{i,j} = a_{1,1+(n+j-i) \bmod n}$. Let P_n denote the $(0,1)$ $n \times n$ matrix with **1**s in positions $(i, i+1)$, $i = 1,\ldots,n-1$, and $(n,1)$ and **0**s everywhere else. Many of the early papers on this topic express circulant matrices in the form

$$A_n = a_1 P_n^{s_1} + a_2 P_n^{s_2} + \cdots + a_k P_n^{s_k} \tag{1.1}$$

where $0 \le s_1 < s_2 < \cdots < s_k < n$ and $a_i = a_{1,s_i+1}$.

The first major result on permanents of $(0,1)$ circulants is due to Metropolis, Stein and Stein [12]. Let $k > 0$ be fixed and $A_{n,k} = \sum_{i=0}^{k-1} P_n^i$, be the $n \times n$ circulant matrix whose first row is composed of **1**s in its first k columns and **0**s everywhere else. Then [12] showed that, as a function of n, $\mathrm{Perm}(A_{n,k})$ satisfies a fixed order constant-coefficient recurrence relation in n

*Partially supported by HK CERG grants HKUST6162/00E, HKUST6082/01E and HKUST6206/02E.
Dept. of Computer Science, Hong Kong U.S.T., Clear Water Bay, Kowloon, Hong Kong. Email addresses are {golin,cscho,yalding}@cs.ust.hk.

and therefore, could be calculated in polynomial time in n (after a superpolynomial "start-up cost" in k for deriving the recurrence relation).

This result was greatly improved by Minc who showed that it was only a very special case of a general rule. Let $0 \leq s_1 < s_2 < \cdots < s_k < n$ be *any* fixed sequence and set $A_n = A_n(s_1, \ldots, s_k) = P_n^{s_1} + P_n^{s_2} + \cdots + P_n^{s_k}$. In [14, 15] Minc proved that $\mathrm{Perm}(A_n)$ always satisfies a constant-coefficient recurrence relation in n of order $2^{s_k} - 1$. Minc's theorem was proven by manipulating algebraic properties of A_n. Note, that as mentioned by Minc, this result is difficult to apply for large s_k since, in order to derive the coefficients of the recurrence relation it is first necessary to evaluate $\mathrm{Perm}(A_n)$ for $n \leq 2(2^{s_k} - 1)$ and, using, Ryser's algorithm, this requires $\Omega\left(2^{2^{s_k}}\right)$ time.

Later Codenotti, Resta and various coauthors improved these results in various ways; e.g. in [2] showing how to evaluate *sparse* circulant matrices of size ≤ 200; in [4, 5] showing that the permanents of circulants with only three **1**s per row can be evaluated in polynomial time; in [6] showing how the permanents of some special sparse circulants can be expressed in terms of determinants and are therefore solvable in polynomial time; in [2] showing that the permanents of *dense* circulants are hard to calculate and in [7] that even approximating the permanent of an arbitrary circulant modulo a prime p is "hard" unless $\mathbf{P}^{\#\mathbf{P}} = \mathbf{BPP}$.

In this paper we return to the original problem of Minc. Our first main result will be to show that if circulant *matrix* $A_n(s_1, \ldots, s_k)$ is interpreted as the adjacency matrix of a directed circulant *graph* C_n, then counting the number of cycle-covers of C_n using a *transfer matrix* approach immediately reproves Minc's result. As well as reproving Minc's original result this new technique will then permit us extend the result to a much larger set of circulant graphs as well as address other related problems. To explain, we first need to introduce some notation.

DEFINITION 1.2. *See Figure 1. Let* $C_n^{s_1, s_2, \cdots, s_k}$. *be the* n*-node* directed circulant graph *with jumps* $S = \{s_1, s_2, \ldots s_k\}$. *(Note that in this definition we allow negative* s_i*.) Formally,*

$$C_n^{s_1, s_2, \ldots, s_k} = (V(n), E_C(n))$$

where

$$V(n) = \{0, 1, \ldots, n-1\}$$

and

$$E_C(n) = \Big\{(i,j) : (j-i) \bmod n \in S\Big\}.$$

Note: we will assume that S contains at least one non-negative s_i since if all the s_i were negative we could multiply them by -1 and get an isomorphic graph. Also, we will often write C_n as shorthand for $C_n^{s_1, s_2, \cdots, s_k}$.

Let $G = (V, E)$ be a graph, $T \subseteq E$ and $v \in V$. Define $\mathtt{ID}_T(v)$ to be the *indegree* of v in graph (V, T) and $\mathtt{OD}_T(v)$ to be the *outdegree* of v in (V, T). $T \subseteq E$ is a *cycle-cover* of G if

$$\forall v \in V, \quad \mathtt{ID}_T(v) = \mathtt{OD}_T(v) = 1.$$

DEFINITION 1.3. *Let* $S = \{s_1, s_2, \ldots s_k\}$ *be given. Set*

$$\mathcal{CC}(n) = \{T \subseteq C_n : T \text{ is a cycle-cover of } C_n\}$$

and

$$T(n) = |\mathcal{CC}(n)| = \text{No. of cycle-covers of } C_n.$$

Note that, by the standard correspondence mentioned before, $A_n(s_1, \ldots, s_k)$ is the adjacency matrix of $C_n^{s_1, s_2, \cdots, s_k}$ and $T(n) = \mathrm{Perm}(A_n(s_1, \ldots, s_k))$. So, calculating $T(n)$ is equivalent to calculating permanents of $A_n(s_1, \ldots, s_k)$.

In [9, 10] the authors of this paper were interested in counting spanning trees and other structures in *undirected circulant* graphs. The main tool introduced there was a recursive decomposition of such graphs. In Section 2 we describe a related recursive decomposition of *directed circulant* graphs. Our technique will be to use this decomposition to show that for some constant m there is a $m \times 1$ (column) vector function $\bar{T}(n)$ such that

$$\forall n \geq 2\bar{s}, \quad T(n) = \beta \bar{T}(n) \quad \text{and} \quad \bar{T}(n+1) = A\bar{T}(n) \tag{1.2}$$

where $\bar{s}$ is a constant to be defined later (but reduces to $\bar{s} = s_k$ for the Minc formulation described previously), β is a $1 \times m$ constant row-vector and A is a constant $m \times m$ matrix. Such an A is known as a *transfer-matrix* see, e.g., [18].

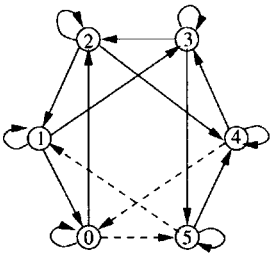
Let $P(x) = \sum_{i=0}^{t} p_i x^i$ be any polynomial that annihilates A, i.e., $P(A) = 0$. Then it is easy to see that $\forall n \geq 2\bar{s}$,

$$\begin{aligned}
\sum_{i=0}^{t} p_i T(n+i) &= \beta \left(\sum_{i=0}^{t} p_i A^{n+i-2\bar{s}} \right) \bar{T}(2\bar{s}) \\
&= \beta A^{n-2\bar{s}} \left(\sum_{i=0}^{t} p_i A^i \right) \bar{T}(2\bar{s}) \\
&= \beta A^{n-2\bar{s}} \mathbf{0} \bar{T}(2\bar{s}) \\
&= 0
\end{aligned}$$

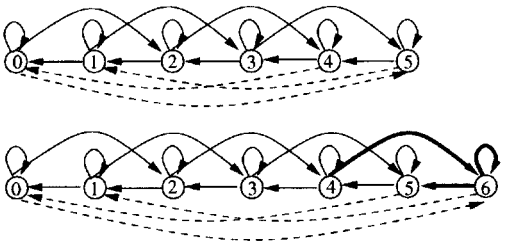
where $\mathbf{0}$ denotes the $m \times m$ zero matrix and 0 a scalar; $T(n)$ thus satisfies the degree-t constant coefficient recurrence relation $T(n+t) = \sum_{i=0}^{t-1} -\frac{p_i}{p_t} T(n+i)$ in n. By

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

(a) $P_n^{-1} + I + P_n^2$ $(n = 6)$



(b) $C_n^{-1,0,2}$ $(n = 6)$



(c) $C_n^{-1,0,2}$ $(n = 6, 7)$

Figure 1: $C_n^{-1,0,2}$: Circulant matrix (a) is the adjacency matrix of circulant graph $C_6^{-1,0,2}$ in (b). In (b), the solid edges are L_n and the dashed edges are $\texttt{Hook}(n)$. (c) illustrates $C_n^{-1,0,2}$ $(n = 6, 7)$ drawn in lattice graph representation. The bold edges in $C_7^{-1,0,2}$ are $\texttt{New}(n)$. Note that the $\texttt{Hook}(n)$ edges for $n = 6, 7$ are "independent" of n.

the Cayley-Hamilton theorem, the characteristic polynomial of A, which has degree $\leq m$, must annihilate A, so such a polynomial exists and $T(n)$ satisfies a recurrence relation of at most degree m. In our notation, Minc's theorem is that $T(n)$ satisfies a recurrence relation of degree $2^{\bar{s}} - 1$. Unfortunately, in our construction, $m = 2^{2\bar{s}}$ so the characteristic polynomial does not suffice for our purposes. Our next step will involve showing that even though A is of size $2^{2\bar{s}} \times 2^{2\bar{s}}$, there is a much smaller P, of degree $2^{\bar{s}} - 1$, that annihilates A, thus reproving Minc's theorem. We point out that this degree reduction of the transfer matrix (to the square-root of the original size) is, a-priori, quite unexpected, and does not occur in the undirected-circulant counting problems analyzed in [9, 10].

One interesting consequence of this new derivation is that, unlike in Minc's proof, to derive the recurrence relation it is no longer necessary to start by spending $\Omega\left(2^{2^{\bar{s}}}\right)$ time calculating the first $2^{\bar{s}}$ values of $T(n)$ using Ryser's method. Instead one only has to calculate A, β, the polynomial P and the first $2^{\bar{s}}$ values of $\bar{T}(n)$ which, as we will see later, can all be done in $O(\bar{s}2^{5\bar{s}})$ time, reducing the start-up complexity from doubly-exponential in $\bar{s}$ to singularly exponential.

Another, albeit minor, consequence of this new derivation is that it can also handle non $(0,1)$ circulants. That is, given *any* matrix A_n of the form (1), even when the a_i are not restricted to be in $\{0,1\}$ the technique shows that $\text{Perm}(A_n)$ satisfies a recurrence relation of degree $2^{\bar{s}} - 1$. This is only a minor consequence, though, since working through the details of Minc's original proof it is possible to modify it to get the same result.

A much more important new consequence, and the major motivation for this paper, is the fact that the proof can be extended to evaluate the permanents of *non-constant* jump circulant matrices, something which has not been addressed before. To explain this, we generalize Definition 1.2 to

DEFINITION 1.4. *See Figure 2(a). Let* p, s, $p_1, p_2, \ldots, p_k$ *and* $s_1, s_2, \ldots, s_k$ *be fixed integral constants with such that* $\forall i$, $0 \leq p_i < p$. *Set* $S = \{p_1 n + s_1, p_2 n + s_2, \cdots, p_k n + s_k\}$. *Denote the* $(pn + s)$*-node* directed circulant graph *with jumps* S *by*

$$C_n = C_{pn+s}^{p_1 n+s_1, p_2 n+s_2, \cdots, p_k n+s_k} = (V(n), E_C(n))$$

where

$$V(n) = \{0, 1, \ldots, pn + s - 1\}$$

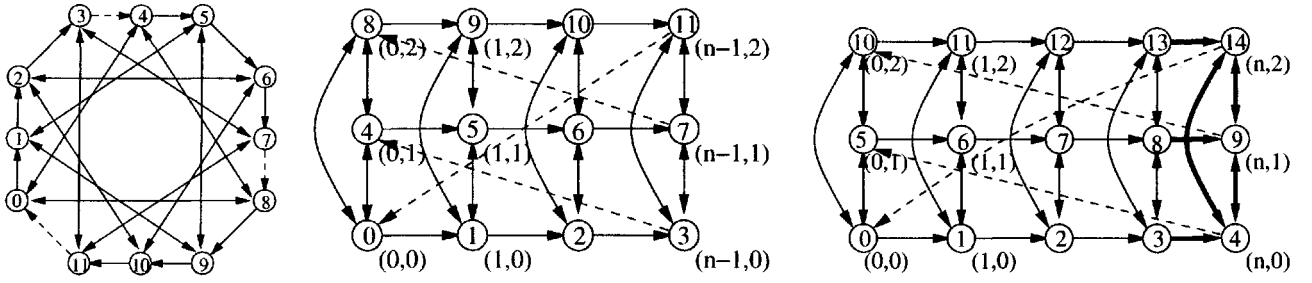
and

$$E_C(n) = \Big\{(i,j) : (j-i) \bmod (pn+s) \in S\Big\}.$$

Note that $A_{pn+s}(p_1 n + s_1, p_2 n + s_2, \cdots, p_k n + s_k)$ is the adjacency matrix of C_n so, counting the cycle-covers in C_n is equivalent to evaluating $\text{Perm}(A_{pn+s}(p_1 n + s_1, p_2 n + s_2, \cdots, p_k n + s_k))$. Our method of counting the cycle covers in C_n will be to derive a new recursive decomposition of C_n (which might be of independent interest) and use it to show that an analogue of (1.2) holds in the non-constant jump case as well; thus $T(n)$ still satisfies a constant-coefficient recurrence relation in n. For example, in Table 1, we show the recurrence relation for the number of cycle covers in $C_{3n}^{1,n+1,2n}$ and $C_{3n}^{0,n,2n-1}$.

In the next section we describe the new recursive decompositions of C_n, for both constant and non-constant jumps, upon which our technique is based. In Section 3 we show how this permits easily reproving Minc's result for non-constant circulants. We then describe the minor modifications that are needed to extend the proof to non-constant circulants.

Note: Due to space limitations in this extended abstract only the proof skeleton is given, with many



(a) $C_{3n}^{1,n,2n}$ $(n=4)$ (b) $C_{3n}^{1,n,2n}$ $(n=4)$ (c) $C_{3n}^{1,n,2n}$ $(n=5)$

Figure 2: $C_{3n}^{1,n,2n}$, a non-constant jump circulant: Dashed edges are $\texttt{Hook}(n)$. Solid edges are L_n. (a) and (b) are two representations of the graph when $n=4$. Note the lattice representation in (b). (c) is is the case $n=5$. The bold solid edges on the right are $\texttt{New}(n)$. The 3 vertices on the right are $VN(n)$. Note that the dashed $\texttt{Hook}(n)$ edges for both $n=4,5$ are "independent" of n.

$C_n^{-1,0,1}$ $C_n^{0,1,2}$	$T(n)=2T(n-1)-T(n-3)$ initial values $9,13,12$ for $n=4,5,6$	$T(n)\sim\phi^n$ $\phi=(1+\sqrt{5})/2$
$C_{3n}^{1,n+1,2n}$ $C_{3n}^{0,n,2n-1}$	$T(n)=5T(n-1)-5T(n-2)-5T(n-3)+6T(n-4)$ initial values $17,45,113,309$ for $n=2,3,4,5$	$T(n)\sim 3^n$

Table 1: The number of cycle-covers $T(n)$ in directed circulant graphs with constant jumps $C_n^{-1,0,1}$ and $C_n^{0,1,2}$, and with non-constant jumps $C_{3n}^{1,n+1,2n}$ and $C_{3n}^{0,n,2n-1}$, as derived by the technique in this paper. Note that inside each pair of graphs, the number of cycle covers is the same. This is to be expected, since their adjacency matrices are just linear circular shifts of each other so the permanents of their adjacency matrices are the same.

of the details omitted. Also, when reproving Minc's result, we only prove that $\text{Perm}(A_n)$ *satisfies a degree* $2^{\bar{s}}$ *recurrence relation and not a* $2^{\bar{s}}-1$ *one.*

2 A Recursive Decomposition of Directed Circulant Graphs

The main conceptual difficulty with deriving a recurrence relation for $T(C_n)$ is that larger circulant graphs can not be built recursively out of smaller ones. The crucial observation, though, is that, there is *another graph*, L_n, the *lattice graph*, that *can* be built recursively, and C_n can then be constructed from L_n through the addition of a constant number of edges[1]. In [9, 10] the authors of this paper developed such a recursive decomposition for *undirected* circulant graphs as a tool for counting the number of spanning trees in such graphs. In what follows we develop a corresponding decomposition for *directed circulants* that will permit counting cycle-covers.

We first show this for the restricted case in which S, the set of jumps, is constant (independent of n), where it is easy to visualize. After deriving the relevant properties we extend the decomposition to the more complicated case in which the set of jumps can depend upon n.

DEFINITION 2.1. *See Figure 1. Let* $S=\{s_1,s_2,\ldots s_k\}$, *where the* s_i *are fixed integers. Define the* n*-node* lattice graph *with jumps* S

$$L_n^{s_1,s_2,\ldots,s_k}=(V(n),E_L(n))$$

where

$$E_L(n)=\Big\{(i,j):j-i\in S\Big\}.$$

Now set

$$\texttt{Hook}(n)=E_C(n)-E_L(n)$$

and

$$\texttt{New}(n)=E_L(n+1)-E_L(n).$$

Note that this implies

$$(2.3)\; L_{n+1}=L_n\cup\texttt{New}(n)\quad\textit{and}\quad C_n=L_n\cup\texttt{Hook}(n).$$

The simple but important observation is that, *when* n *is viewed as a label rather than as a number,* $\texttt{Hook}(n)$ and $\texttt{New}(n)$ are *independent* of the actual *value* of n.

[1] To put this into context, this is very similar to the definition of *Recursive families* for undirected graphs [3, 17].

LEMMA 2.1. *Set* $S^+ = \{s \in S : s \geq 0\}$ *and* $S^- = \{s \in S : s < 0\}$. *Then*

$$\begin{aligned}
\texttt{Hook}(n) &= \left(\bigcup_{s\in S^+} \{(n-j, s-j) : 1 \leq j \leq s\}\right) \\
&\quad \cup \left(\bigcup_{s\in S^-} \{(j, n+s+j) : 0 \leq j < |s|\}\right), \\
\texttt{New}(n) &= \left(\bigcup_{s\in S^+} \{(n-s, n)\}\right) \cup \left(\bigcup_{s\in S^-} \{(n, n+s)\}\right).
\end{aligned}$$

Further set $s^+ = \max_{s\in S^+} s$, *and* $s^- = \max_{s\in S^-} |s|$ *(if* $S^- = \emptyset$ *set* $s^- = 0$*). For later use we define* $\bar{s} = s^+ + s^-$. *Now define*

$$\begin{aligned}
L^+(n) &= \{0, \ldots s^+ - 1\}, \\
R^+(n) &= \{n - s^+, \ldots, n-1\}, \\
L^-(n) &= \{0, \ldots s^- - 1\}, \\
R^-(n) &= \{n - s^-, \ldots, n-1\}.
\end{aligned}$$

Then

$$\begin{aligned}
\texttt{Hook}(n) &\subseteq (R^+(n) \times L^+(n)) \cup (L^-(n) \times R^-(n)) \\
\texttt{New}(n) &\subseteq (R^+(n) \times \{n\}) \cup (\{n\} \times R^-(n)) \\
&\quad \cup \{(n,n)\} \qquad (2.4)
\end{aligned}$$

Important Note: In this section and the next we will always assume that $n \geq 2\bar{s}$ *since this will guarantee that* $(L^+(n) \cup L^-(n)) \cap (R^+(n) \cup R^-(n)) = \emptyset$. *Without this assumption some of our proofs would fail. Also note that the* $\{(n,n)\}$ *term in* $\texttt{New}(n)$ *is only needed when* $0 \in S$.

We now extend the above definitions and lemmas to the case of non-constant circulants. This will require a change in the way that we visualize the nodes of C_n; until, now, as in Figure 1(c), we visualized them as points on a line with the edges in $\texttt{Hook}(n)$ connecting the left and right endpoints of the line. In the non-constant jump case it will be convenient to visualize them as points on a bounded-height *lattice*, where $\texttt{Hook}(n)$ connects the left and right boundaries of the lattice. We start by introducing a new graph:

DEFINITION 2.2. *See Figure 2. Let* $p, s,\ p_1, p_2, \ldots, p_k$ *and* $s_1, s_2, \ldots, s_k$ *be given integral constants such that* $\forall i, 0 \leq p_i < p$. *Set* $S = \{p_1 n + s_1, p_2 n + s_2, \cdots, p_k n + s_k\}$. *For* u, v *and integer* n, *set* $f(n; u, v) = un + v$. *Define*

$$\widehat{C}_n = \left(\widehat{V}_C(n),\ \widehat{E}_C(n)\right)$$

where

$$\begin{aligned}
\widehat{V}(n) &= \{(u,v) : 0 \leq u \leq p-1,\ 0 \leq v \leq n-1\} \\
&\quad \cup \{(p-1, v) : n \leq v \leq n+s-1\}
\end{aligned}$$

and $\widehat{E}_C(n)$ *is the set of the union of all edges*

$$((u_1, v_1), (u_2, v_2)),$$

where the union is taken over all

$$(u_1, v_1),\ (u_2, v_2) \in \widehat{V}(n)$$

such that

$$f(n; u_2, v_2) - f(n; u_1, v_1) \bmod (pn + s) \in S.$$

Directly from the definition we see $\widehat{C}_n$ is *isomorphic* to $C_n = C_{pn+s}^{p_1 n + s_1, p_2 n + s_2, \cdots, p_k n + s_k}$. In particular, cycle-covers of $\widehat{C}_n$ are in 1-1 correspondence with cycle covers of C_n so we can restrict ourselves to counting cycle covers of $\widehat{C}_n$. We now introduce the generalization of Definition 2.1.

DEFINITION 2.3. *Let* $p, s,\ p_1, p_2, \ldots, p_k$ *and* $s_1, s_2, \ldots, s_k$ *and* S, f *be as in Definition 2.2. Define the* $pn + s$*-node* lattice graph *with jumps* S

$$L_n = (\widehat{V}(n), \widehat{E}_L(n))$$

where $\widehat{E}_L(n)$ *is the set of the union of all edges*

$$((u_1, v_1), (u_2, v_2))$$

where the union is taken over all

$$(u_1, v_1),\ (u_2, v_2) \in \widehat{V}(n),$$

such that

$$f(n; u_2, v_2) - f(n; u_1, v_1) = p_i n + s_i \bmod (pn + s)$$

and

$$u_2 - u_1 = p_i \bmod p.$$

Now set

$$\texttt{Hook}(n) = \widehat{E}_C(n) - \widehat{E}_L(n)$$

and

$$\texttt{New}(n) = \widehat{E}_L(n+1) - \widehat{E}_L(n).$$

Note that this implies

$$\begin{aligned}
L_{n+1} &= L_n \cup \texttt{New}(n) \\
&\quad \textit{and} \qquad (2.5) \\
\widehat{C}_n &= L_n \cup \texttt{Hook}(n).
\end{aligned}$$

It is now straightforward to derive an analogue of Lemma 2.1 showing that $\texttt{Hook}(n)$ and $\texttt{New}(n)$ are *independent* of the actual *value* of n. Let $NV(n) = V_L(n+1) - V_L(n)$. $NV(n)$ will be the *new* vertices in $V_L(n+1)$. Note that we did not define this for *fixed-jump* circulant graphs since in the fixed-jump case there is only the one new vertex $V_L(n+1) - V_L(n) = \{n\}$ and $NV(n)$ would be constant.

LEMMA 2.2. *Set* $S^+ = \{s_i \in S : s_i \geq 0\}$, $S^- = \{s_i \in S : s_i < 0\}$, *and* $s^+ = \max_{s_i \in S^+} s_i$, $s^- = \max_{s_i \in S^-} |s_i|$ *(if* $S^- = \emptyset$ *set* $s^- = 0$*).*
Let $t_1 = s^+ - 1$, $t_2 = s^- - 1$ *and* $r = \min\{-s, 0\}$.
Now let $0 \leq u \leq p-1$ *and define*

$$\begin{aligned} L^+(n) &= \{(u,v) : 0 \leq v \leq \max\{t_1, t_1+s\}\}, \\ R^+(n) &= \{(u, n-1-v) : r \leq v \leq \max\{t_1, t_1-s\}\}, \\ L^-(n) &= \{(u,v) : 0 \leq v \leq \max\{t_2+s, t_2+2s\}\}, \\ R^-(n) &= \{(u, n-1-v) : r \leq v \leq \max\{t_2, t_2+s\}\}. \end{aligned}$$

Then

$$\begin{aligned} \texttt{Hook}(n) &\subseteq (R^+(n) \times L^+(n)) \\ &\quad \cup (L^-(n) \times R^-(n)), \\ \texttt{New}(n) &\subseteq (R^+(n) \times NV(n)) \\ &\quad \cup (NV(n) \times R^-(n)) \\ &\quad \cup (NV(n) \times NV(n)). \end{aligned} \tag{2.6}$$

3 A New Proof of Minc's result

Let CC be a cycle-cover of C_n. Then, in $T = CC - \texttt{Hook}(n)$, almost all vertices v except possibly those that have an edge of $\texttt{Hook}(n)$ hanging off of them, have $\texttt{ID}_T(v) = \texttt{OD}_T(v) = 1$. Referring to (2.4) this motivates

DEFINITION 3.1. $T \subseteq E_L(n)$ *is a* legal cover of L_n *if*

- $\forall v \in V$, $\texttt{ID}_T(v) \leq 1$ *and* $\texttt{OD}_T(v) \leq 1$.
- $\forall v \in V - (L^+(n) \cup R^-(n))$, $\texttt{ID}_T(v) = 1$.
- $\forall v \in V - (L^-(n) \cup R^+(n))$, $\texttt{OD}_T(v) = 1$.

Then, from (2.4) we have

LEMMA 3.1.
(a) If $T \subseteq E_C(n)$ *is a cycle-cover of* C_n, *then* $T - \texttt{Hook}(n)$ *is a legal-cover of* L_n.

(b) If $T \subseteq E_L(n+1)$ *is a legal-cover of* L_{n+1}, *then* $T - \texttt{New}(n)$ *is a legal-cover of* L_n.

From the definition of legal covers we can classify and partition legal covers by the appropriate in/out degrees of their vertices in $L^+(n), L^-(n), R^+(n), R^-(n)$.

DEFINITION 3.2. A *is a* binary r-tuple *if* $A = (A(0), A(1), \ldots, A(r-1))$ *where* $\forall i, A(i) \in \{0,1\}$.

Let $\mathcal{P}$ *be the set of* $2^{2\bar{s}}$ *tuples* (L_+, L_-, R_+, R_-) *where* L_+, L_-, R_+, R_- *are, respectively, binary* s^+, s^-, s^+, s^- *tuples.*

Let T *be a* legal-cover *of* L_n. *The* classification of T *will be* $C(T) = (L_+^T, L_-^T, R_+^T, R_-^T) \in \mathcal{P}$ *where*

$$\begin{aligned} \forall 0 \leq i < s^+, \quad L_+^T(i) &= \texttt{ID}_T(i) \\ R_+^T(i) &= \texttt{OD}_T(n-1-i), \\ \forall 0 \leq i < s^-, \quad R_-^T(i) &= \texttt{ID}_T(n-1-i), \\ L_-^T(i) &= \texttt{OD}_T(i). \end{aligned}$$

If T *is* not *a legal-cover then we will use the convention that* $C(T) = \emptyset$. *Finally, set*

$$\begin{aligned} \mathcal{L}(n) &= \{T \subseteq E_L(n) : T \text{ is a legal cover of } L_n\} \\ \mathcal{L}_X(n) &= \{T \in \mathcal{L}(n) : C(T) = X\} \\ T_X(n) &= |\mathcal{L}_X(n)| \end{aligned}$$

so $T_X(n)$ *is the number of legal-covers of* L_n *with classification* X.

The main reason for introducing these definitions is that checking whether a legal cover T of L_n can be completed to a cycle-cover of C_n doesn't depend upon all of T but only on its classification $C(T)$. Furthermore, how a legal-cover in L_n expands to a legal cover in L_{n+1} will also only depend upon $C(T)$.

LEMMA 3.2. *See Figures 3 and 4.*
(a) Let $X = (L_+^X, L_-^X, R_+^X, R_-^X) \in \mathcal{P}$ *and* $S \subseteq \texttt{Hook}(n)$.
Let T *be a legal cover in* L_n *with* $C(T) = X$.

Then whether $T \cup S$ *is a cycle cover of* C_n *depends only upon* X *and* S *(and* not at all on n*). In particular, if* T *is a legal-cover of* L_n *and* T' *is a legal cover of* $L_{n'}$ *with* $C(T) = C(T')$ *then*

$T \cup S$ *is a cycle-cover of* C_n
iff
$T' \cup S$ *is a cycle-cover of* $C_{n'}$

Note: We will write $X \cup S$ *is a cycle cover* to denote that $T \cup S$, with $C(T) = X$, is a cycle cover.

(b)Let T' *be a legal cover in* L_n *with* $C(T') = X' \in \mathcal{P}$. *and* $S \subseteq \texttt{New}(n)$.

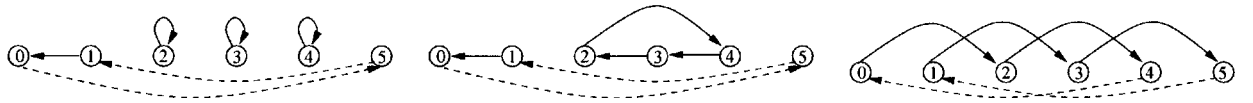
Then whether $C(T' \cup S) = X$ *depends only upon* X' *and* S *(and* not at all on n*). In particular, if* T' *is a legal-cover of* L_n *and* T'' *is a legal cover of* $L_{n'}$ *with* $C(T') = C(T'')$ *then*

$$C(T' \cup S) = C(T'' \cup S)$$

Note: We will write $(X' \cup S) = X$ to denote that, when $C(T') = X'$, $C(T' \cup S) = X$.

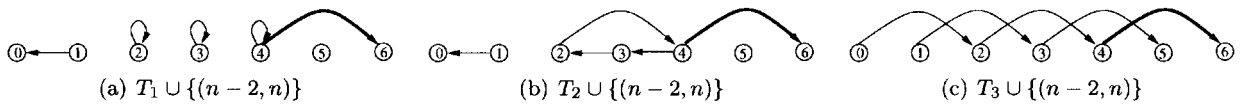
Proof. To prove (a) recall that $T \cup S$ is a legal-cover of L_n if and only if,
$\forall v \in V$, $\texttt{ID}_{T \cup S}(v) = \texttt{OD}_{T \cup S}(v) = 1$ or

$$\forall v \in V, \texttt{ID}_S(v) = 1 - \texttt{ID}_T(v) \quad \text{and} \quad \texttt{OD}_S(v) = 1 - \texttt{OD}_T(v) \tag{3.7}$$



(a) CC_1 and T_1 (b) CC_2 and T_2 (c) CC_3 and T_3

Figure 3: All of the figures are in $C_6^{-1,0,2}$. Dashed edges are $\texttt{Hook}(n)$. The solid plus dashed edges are three different cycle covers CC_i, $i = 1,2,3$ in C_6. Removing the dashed $\texttt{Hook}(n)$ edges leaves three *legal* covers T_i, $i = 1,2,3$, in L_6. Note that $s^+ = 2$ and $s^- = 1$ so classifications are of the form $(L_+^T, L_-^T, R_+^T, R_-^T)$ where L_+^T and R_+^T are pairs and L_-^T and R_-^T are singletons. Calculation gives $C(T_1) = C(T_2) = X_1' = (\,(1,0),\,(0),\,(0,1),\,(0)\,)$ and $C(T_3) = X_3' = (\,(0,0),\,(1),\,(0,0),\,(1)\,)$.



(a) $T_1 \cup \{(n-2,n)\}$ (b) $T_2 \cup \{(n-2,n)\}$ (c) $T_3 \cup \{(n-2,n)\}$

Figure 4: n was increased from 6 to 7 and $S = \{(4,6)\} \subseteq \text{New}(6)$ was added to the T_i of the previous figure. Note that, in L_7, $C(T_1 \cup S) = C(T_2 \cup S) = \emptyset$ since they are no longer legal covers. Also, $C(T_3 \cup S) = X_3(= X_3') = (\,(0,0),\,(1),\,(0,0),\,(1)\,)$. Thus, $C(X_1' \cup S) = \emptyset$ and $C(X_3' \cup S) = X_3'$.

From Lemma 2.1 and the definition of a legal cover we have that this is true if and only if

$$\begin{aligned} \forall i \le s^+, \quad \texttt{ID}_S(i) &= 1 - L_+^X(i), \\ \texttt{OD}_S(i) &= 1 - L_-^X(i), \\ \forall i \le s^-, \quad \texttt{ID}_S(n-1-i) &= 1 - L_+^X(i,), \\ \texttt{OD}_S(n-1-i) &= 1 - L_-^X(i). \end{aligned}$$

and this is only dependent upon X and S and not upon n or any other properties of T.

The proof of (b) is similar and omitted here. □

DEFINITION 3.3. *For $X, X' \in \mathcal{P}$, $S \subseteq \texttt{Hook}(n)$ and $S' \subseteq \texttt{New}(n)$ set*

$$\beta_{X,S} = \begin{cases} 1 & \text{if } X \cup S \text{ is a cycle cover} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\alpha_{X,X',S'} = \begin{cases} 1 & \text{if } C(X' \cup S') = X \\ 0 & \text{otherwise} \end{cases}.$$

Now set

$$\beta_X = \sum_{S \subseteq \texttt{Hook}(n)} \beta_{X,S}$$

and

$$\alpha_{X,X'} = \sum_{S' \subseteq \texttt{New}(n)} \alpha_{X,X',S'}.$$

Note that β_X and $\alpha_{X,X'}$ are constants that can be mechanically calculated. Then Lemmas 3.1 and 3.2 immediately imply our main technical result, which is equivalent to (1.2).

LEMMA 3.3.

$$T(n) = \sum_{X \in \mathcal{P}} \beta_X T_X(n)$$

and

$$T_X(n+1) = \sum_{X' \in \mathcal{P}} \alpha_{X,X'} T_X(n).$$

Let $m = |\mathcal{P}| = 2^{2\bar{s}}$. Take any arbitrary ordering of $\mathcal{P}$ and define the $1 \times m$ constant vector $\beta = (\beta_X)_{X \in \mathcal{P}}$ and $m \times m$ constant matrix $A = (\alpha_{X,X'})_{X,X' \in \mathcal{P}}$. Finally, set $\bar{T}(n) = \text{col}(T_X(n))_{X \in \mathcal{P}}$ to be a $m \times 1$ column vector. Then, Lemma 3.3 is exactly equation (1.2) which immediately implies that $T(n)$ satisfies a fixed-degree constant coefficient recurrence relation where the degree of the recurrence is at most the degree of any polynomial $P(x)$ such that $P(A) = 0$. By the Cayley-Hamilton theorem, $Q(A) = 0$, $Q(x)$ is the degree $m = 2^{2\bar{s}}$ characteristic polynomial $Q(x) = \det(IX - A)$.

We will now see that it is possible to reduce this degree from $2^{2\bar{s}}$ down to below $2^{\bar{s}}$.

LEMMA 3.4. *Let $A = (\alpha_{X,X'})$. Then there is a degree $2^{\bar{s}} - 1$ polynomial $P(x)$ such that $P(A) = 0$.*

Proof. Recall that $\bar{s} = s^+ + s^-$. Suppose $X = (L_+^X, L_-^X, R_+^X, R_-^X)$ and $X' = (L_+^{X'}, L_-^{X'}, R_+^{X'}, R_-^{X'})$.

Recall that $\alpha_{X,X'} = \sum_{S \subseteq \texttt{New}(n)} \alpha_{X,X',S}$ where $\alpha_{X,X',S} = 1$ if and only if $C(X' \cup S) = X$, and is otherwise 0.

Now let L_+, L_- be any 2^{s^+} and 2^{s^-} binary tuples and partition $\mathcal{P}$ up into $2^{\bar{s}}$ sets of size $2^{\bar{s}}$, $\mathcal{P}_{L_+,L_-} = \{X \in \mathcal{P} : L_+^X = L_+,\, L_-^X = L_-\}$.

Note that, if $S \subseteq \texttt{New}(n)$, none of S's edges have endpoints in $L^+(n)$ or $L^-(n)$. Intuitively, this is because edges in $\texttt{New}(n)$ only connect vertices near the *right* side of the lattice and do not touch any vertices on the *left* side of the lattice.

Thus, if $\alpha_{X,X',S} = 1$, then $L^X_+ = L^{X'}_+$ and $L^X_- = L^{X'}_-$. In particular this means that if $\alpha_{X,X',S} = 1$ then X, X' are both in the same partition set $\mathcal{P}_{L_+,L_-}$.

Now suppose that $\alpha_{X,X',S} = 1$. Let $\bar{L}_+$, $\bar{L}_-$ be *any* other 2^{s^+} and 2^{s^-} binary tuples and set

$$\bar{X} = (\bar{L}_+, \bar{L}_-, R^X_+, R^X_-) \quad \text{and} \quad \bar{X}' = (\bar{L}_+, \bar{L}_-, R^{X'}_+, R^{X'}_-). \tag{3.8}$$

Then, again using the fact that none of the endpoints of S are in $L^+(n)$ or $L^-(n)$ we have that $C(X' \cup S) = X$ if and only if $C(\bar{X}' \cup S) = \bar{X}$ so $\alpha_{X,X'} = \alpha_{\bar{X},\bar{X}'}$.

When constructing matrix $A = (\alpha_{X,X'})_{X,X' \in \mathcal{P}}$ we previously allowed any arbitrary ordering of $\mathcal{P}$. Now order the $X \in \mathcal{P}$ lexicographically; this groups all of the X in a particular $\mathcal{P}_{L_+,L_-}$ consecutively. The observations above imply that A is partitioned into $2^{\bar{s}} \times 2^{\bar{s}}$ blocks where each block is of size $2^{\bar{s}} \times 2^{\bar{s}}$. The non-diagonal blocks correspond to $\alpha_{X,X'}$ where X, X' are in different partitions so all of the non-diagonal blocks are 0. On the other hand, the fact that $\alpha_{X,X'} = \alpha_{\bar{X},\bar{X}'}$ for the $\bar{X}, \bar{X}'$ defined in (3.8), tells us that all the diagonal blocks are copies of each other.

Let $\bar{A}$ be one of the $2^{\bar{s}} \times 2^{\bar{s}}$ diagonal blocks in A. A can then be denoted as $A = \text{diag}(\bar{A}, \bar{A}, \ldots, \bar{A})$ where A contains $2^{\bar{s}}$ copies of $\bar{A}$ on its diagonal. Thus, $\forall i$, $A^i = \text{diag}(\bar{A}^i, \bar{A}^i, \ldots, \bar{A}^i)$. In particular, this means that any polynomial $P(x)$ that annihilates $\bar{A}$ also annihilates A. Since $\bar{A}$ is a $2^{\bar{s}} \times 2^{\bar{s}}$ matrix, the Cayley-Hamilton theorem says that the characteristic polynomial $\bar{P}(x)$ of $\bar{A}$, which is of degree $2^{\bar{s}}$, annihilates $\bar{A}$.

By a more careful analysis of the structure of $\bar{A}$ it is possible to show that $\bar{P}(x)$ actually has degree $2^{\bar{s}}-1$ but, as mentioned in the introduction, that further analysis will be omitted here. □

Lemma 3.3 tells us that (1.2) holds while Lemma 3.4 tells us that matrix A is annihilated by polynomial $P(x)$ of degree $2^{\bar{s}} - 1$. Combining them gives that $T(n)$ satisfies a degree-$(2^{\bar{s}} - 1)$ constant coefficient recurrence relation. In order to actually *derive* the recurrence relation, though, it is necessary to calculate the $\alpha_{X,X'}$, β_X, $\bar{T}$ and P as well as the first $2^{\bar{s}} - 1$ values of $T(n) = \beta\bar{T}(n)$. It is relatively straightforward (but omitted in this extended abstract) to see how to evaluate all of these in $O(\bar{s}2^{5\bar{s}})$ time by evaluating $O(2^{2\bar{s}})$ permanents of size $2\bar{s}$ and $O(2^{4\bar{s}})$ of size $\bar{s}$.

We just saw how to calculate the number of cycle-covers in constant-jump circulant graphs. Reviewing the proof, everything followed directly as a consequence from the recursive decomposition of circulant graphs in (2.3) combined with the structural properties of the decomposition given in Lemma 2.1. But, as also derived in Section 2, non-constant jump circulants have exactly the *same* structural properties, given in (2.5) and Lemma 2.2. Therefore, the entire proof developed in Section 3 can be rewritten for non-constant jump circulants. The only difference is in the degree of the recurrence relation for the number of cycle-covers. Reviewing the proof for the constant-jump case we can see that the order of the recurrence relation is really $2^{|R^+(n)|+|R^-(n)|}$ which worked out to $2^{\bar{s}} - 1$. In the non-constant case, from Lemma 2.2, we can calculate that $|R^+(n)| + |R^-(n)| = p(|s| + s^+ + s^-) + 2s$ so the order of the recurrence relation will then be $2^{p(|s|+s^+ + s^-)+2s}$. Note that in the constant jump case we had $p = s = 0$ so this collapses down to $2^{s^+ + s^-} = 2^{\bar{s}}$ which is what we had previously derived. For an example of such a recurrence relation, see the second set of graphs in Table 1.

References

[1] E. Bax and J. Franklin. A permanent algorithm with $\exp[\Omega(n^{1/3}/2 \ln n)]$ expected speedup for 0-1 matrices. *Algorithmica*, 32:157–162, 2002.

[2] A. Bernasconi, B. Codenotti, V. Crespi, and G. Resta. How fast can one compute the permanent of circulant matrices? *Linear Algebra and its Applications*, 292(1-3):15–37, 1999.

[3] N. L. Biggs, R. M. Damerell, and D. A. Sands. Recursive families of graphs. *J. Combin. Theory Ser. B*, 12:123–131, 1972.

[4] B. Codenotti, V. Crespi, and G. Resta. On the permanent of certain (0, 1) toeplitz matrices. *Linear Algebra and its Applications*, 267:65–100, 1997.

[5] B. Codenotti and G. Resta. On the permanent of certain circulant matrices. *Algebraic combinatorics and computer science*, pages 513–532, 2001.

[6] B. Codenotti and G. Resta. Computation of sparse circulant permanents via determinants. *Linear Algebra and its Applications*, 355(1-3):15–34, 2002.

[7] B. Codenotti, J. D. Shparlinski, and A. Winterhof. On the hardness of approximating the permanent of structured matrices. *Computational Complexity*, 11(3-4):158–170, 2002.

[8] P. Dagum, M. Luby, M. Mihail, and U. Vazirani. Polytopes, permanents and graphs with large factors. In *Proceedings of the Twentyninth IEEE Symposium*

on Foundations of Computer Science, pages 412–421, 1988.

[9] M. J. Golin and Y. C. Leung. Unhooking circulant graphs: A combinatorial method for counting spanning trees and other parameters. In *Proceedings of the* 30'th International Workshop on Graph-Theoretic Concepts in Computer Science, pages 296–307, 2004.

[10] M. J. Golin, Y. C. Leung, and Y. J. Wang. Counting spanning trees and other structures in non constant-jump circulant graphs. In *The 15th Annual International Symposium on Algorithms and Computation*, pages 508–521, 2004.

[11] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004.

[12] N. Metropolis, M. L. Stein, and P. R. Stein. Permanents of cyclic (0, 1) matrices. *J. Combin. Theory Ser. B*, 7:291–321, 1969.

[13] H. Minc. *Permanents*, volume 6 of *Encyclopedia of mathematics and its applications*. Addison-Wesley Pub. Co., 1978.

[14] H. Minc. Recurrence formulas for permanents of (0,1)-circulants. *Linear Algebra and its Applications*, 71:241–265, 1985.

[15] H. Minc. Permanental compounds and permanents of (0,1)-circulants. *Linear Algebra and its Applications*, 86:11–42, 1987.

[16] H. Minc. Theory of permanents. *Linear and Multilinear Algebra*, 21(2):109–148, 1987.

[17] M. Noy and A. Ribó. Recursively constructible families of graphs. *Advances in Applied Mathematics*, 32:350–363, 2004.

[18] R. P. Stanley. *Enumerative combinatorics*. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, Calif., 1986.

[19] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput*, 8:410–421, 1979.

[20] Q. F. Yang, R. E. Burkard, E. Cela, and G. Woeginger. Hamiltonian cycles in circulant digraphs with two stripes. *Discrete Math*, 176:233–254, 1997.

Appendices

A Worked example for $C_n^{0,1,2}$

As discussed in the paper we have that $T(n)$, the number of cycle covers in $C_n^{0,1,2}$, satisfies

$$\forall n \geq 2\bar{s}, \quad T(n) = \beta \bar{T}(n) \quad \text{and} \quad \bar{T}(n+1) = A\bar{T}(n)$$

where $\beta = (\beta_X)_{X\in\mathcal{P}}$ and $A = (\alpha_{X,X'})_{X,X'\in\mathcal{P}}$.

For $C_n^{0,1,2}$ we have $s^+ = 2$, $s^- = 0$ so $\bar{s} = s^+ + s^- = 2$. Definition 3.2 then says that every $X \in \mathcal{P}$ is in the form $X = (L_+^X, L_-^X, R_+^X, R_-^X)$ where $L_+^X, R_+^X \in \{0,1\}^2$ and L_-^X, R_-^X are empty. We can therefore represent every X by a four-bit binary vector in which the first two bits represent L_+^X and the last two R_+^X; there are 16 such $X \in \mathcal{P}$. Ordering the X lexicographically we calculate that β is

$$(\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ ),$$

$\bar{T}(4)$ is

$$(\ 1\ 0\ 0\ 0\ 0\ 2\ 1\ 0\ 0\ 3\ 2\ 0\ 0\ 0\ 0\ 1\ )^t$$

(where the t denotes taking the transpose), and *Transfer matrix* $A(X)$ is

$$\left(\begin{array}{cccc|cccc|cccc|cccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right)$$

As predicted by Lemma 3.4, A is partitioned into 16 4×4 blocks where all but the diagonal blocks are 0 and all of the diagonal blocks are equal to some 4×4 matrix $\bar{A}$ which in this case is

$$\bar{A} = \begin{pmatrix} 1&0&0&0\\ 0&1&1&0\\ 0&1&0&0\\ 0&0&0&1 \end{pmatrix}$$

This means that

$$Q(x) = det(Ix - \bar{A}) = (x^2 - x - 1)(x-1)^2,$$

the characteristic polynomial of $\bar{A}$, annihilates A.

Working through the details we can then solve to find that $C_n^{0,1,2} = 2T(n-1) - T(n-3)$ with initial values $T(4) = 9$, $T(5) = 13$, and $T(6) = 12$.

B Other Applications

In this appendix we quickly mention two other applications of the technique introduced in this paper.

The first is in analyzing the number of cycles in certain classes of random restricted permutations. Using the standard cycle-decomposition of a permutation there is 1-1 correspondence between permutations

$C_n^{-1,0,1}$	$T_1(n) = 3T(n-1) - T_1(n-2) - 3T_1(n-3)$ $+T_1(n-4) + T_1(n-5)$ initial values $22, 42, 80, 149, 274$ for $n = 4, 5, 6, 7, 8$	$T_1(n) \sim \frac{\phi^4}{\phi^2+\phi^4} n\phi^n$ $\sim .7236n\phi^n$	$\frac{T_1(n)}{T_0(n)} \sim .7236n$
$C_n^{0,1,2}$	$T_1(n) = 3T(n-1) - 6T_1(n-3) + 2T_1(n-4)$ $+4T_1(n-5) - T_1(n-6) - T_1(n-7)$ initial values $21, 32, 56, 93, 161, 275, 475$ for $n = 4, 5, \ldots, 10$	$T_1(n) \sim \frac{\phi^2}{\phi^2+\phi^4} n\phi^n$ $\sim .2764n\phi^n$	$\frac{T_1(n)}{T_0(n)} \sim .2764n$

Table 2: $T_1(n)$ is the number of cycles in the given graph with n vertices.

$\pi \in S_n$ and cycle-covers of the complete directed graph on n-vertices. For given parameters p, s, p_i, s_i and S as in Definition 1.4 define

$$S_{pn+s}(S) = \{\pi \in S_{pn+s} : \pi[i] - i \bmod (pn+s) \in S\} \quad (2.9)$$

to be the set of permutations in which $\pi[i]$ is restricted by (2.9). Now suppose that we pick a permutation π uniformly at random from $S_{pn+s}(S)$ and set $X =$ # of cycles in π. What can be said about the distribution of X?

By the 1-1 correspondence between permutations and cycle-covers, $\pi \in S_{pn+s}(S)$ if and only if the corresponding cycle-cover is in C_n. Thus, the number of such permutations satisfies $|S_{pn+s}(S)| = T(n)$ where $T(n)$ is the number of cycle-covers in C_n. Suppose now that for cycle cover $T \in \mathcal{CC}(n)$ we define $\#_C(T)$ to be the *number* of cycles composing cover T and set

$$TC_i(n) = \sum_{T \in \mathcal{CC}(n)} (\#_C(T))^i .$$

That is, $TC_0(n) = T(n)$ while $TC_1(n)$ is the total number of cycles summed over all cycle-covers in C_n. Then, again by the correspondence, we have that the moments of X are given by

$$\forall i \geq 0, \quad E(X^i) = \frac{TC_i(n)}{TC_0(n)}.$$

The interesting point is that the transfer matrix approach introduced in this paper can mechanically be extended to counting the total number of cycles in the cycle-covers, to show that for every i, $TC_i(n)$ satisfies a fixed-order constant coefficient recurrence relation. For given, p, s, $p_1, p_2, \ldots, p_k$ and $s_1, s_2, \ldots, s_k$ this permits, for example, calculating $E(X)$ and $Var(X)$.

As an illustration recall the results from Table 1 counting the number of cycle covers in $C_n^{-1,0,1}$ and $C_n^{0,1,2}$. Even though these two graphs are *not isomorphic* they had the same number of cycle-covers because the adjacency matrix of the second is just the adjacency matrix of the first with every row (cyclicly) shifted over one step. Since permanents are invariant under cyclic shifts both matrices have the same permanent which is $\sim \phi^n$ where $\phi = (1+\sqrt{5})/2$.

Using our technique we calculated $TC_1(n)$ for both cases with the results given in Table 2.

In both cases we have that $TC_1(n) \sim cn\phi^n$. This means that if a permutation on n items is chosen at random from the corresponding distribution then, on average, it will have $\frac{T_1(n)}{T_0(n)} \sim cn$ cycles. It is interesting to note that that c is different for the two cases.

The second application of the technique we note is that a minor modification permits using it to show that the number of *Hamiltonian Cycles* in a directed circulant graph C_n also satisfies a constant-coefficient recurrence relation in n. This fact was previously known for *undirected* circulant graphs [9, 10] but doesn't seem to have been known for directed circulants, with the exception of the special case of in(out)-degree 2 circulants [20].

Random partitions with parts in the range of a polynomial*

William M. Y. Goh† Paweł Hitczenko‡

Abstract

Let $\Omega(n,Q)$ be the set of partitions of n into summands that are elements of the set $\mathcal{A} = \{Q(k) : k \in Z^+\}$. Here $Q \in Z[x]$ is a fixed polynomial of degree $d > 1$ which is increasing on $\mathbf{R}^+$, and such that $Q(m)$ is a non-negative integer for every integer $m \geq 0$. For every $\lambda \in \Omega(n,Q)$, let $\mathbf{M}_n(\lambda)$ be the number of parts, with multiplicity, that λ has. Put a uniform probability distribution on $\Omega(n,Q)$, and regard $\mathbf{M}_n$ as a random variable. The limiting density of the random variable $\mathbf{M}_n$ (suitably normalized) is determined explicitly. For specific choices of Q, the limiting density has appeared before in rather different contexts such as Kingman's coalescent, and processes associated with the maxima of Brownian bridge and Brownian meander processes.

1 Introduction and statement of the result

In research on partitions, there have been great synergies between probabilistic, analytic, and combinatorial methods. The oldest literature on partition enumerations, dating back to Hardy and Ramanujan [15], has a purely analytic flavor. But Erdös and Lehner [11] introduced a probabilistic viewpoint that was quite fruitful. Random partitions were developed by Erdös, Szalay, Turan and others, [12, 27, 29, 30, 31, 32]. Some authors, e.g. [5, 16, 17, 22, 25], have studied random partitions with summands restricted to proper subsets of the set of positive integers. Increasingly sophisticated probabilistic ideas have been introduced [13, 2], and these ideas have led to remarkably strong theorems about the joint distribution of part sizes of random integer partitions [23].

In this abstract we concentrate on the limiting distribution of the number of parts in a random partition whose parts are restricted to the range of a polynomial. Specifically, let

$$Q(x) = a_d x^d + a_{d-1}x^{d-1} + \ldots + a_1 x + a_0 \tag{1.1}$$

be a fixed polynomial of degree $d \geq 2$ and we assume that $Q(x)$ is strictly increasing for $x > 0$ and that $Q(m)$ is a non-negative integer for an integer $m \geq 0$. Let $\Omega(n,Q)$ be the set of partitions of n into summands that are elements of the set $\mathcal{A} = \{Q(k) : k \in Z^+\}$. For every $\lambda \in \Omega(n,Q)$, let $\mathbf{M}_n(\lambda)$ be the number of parts, with multiplicity, that λ has. Put a uniform probability measure P_n on $\Omega(n,Q)$, and regard $\mathbf{M}_n$ as a random variable. Note that $\mathbf{M}_n(\lambda) = \sum_a M_a(\lambda)$, where $M_a(\lambda)$ is the multiplicity of the part size a in the P_n-random partition λ. These random variables M_a are clearly not independent since they must satisfy the condition $\sum_{a\in\mathcal{A}} aM_a = n$. Fristedt [13] used a conditioning device that enables one to cope with this dependence. It quickly proved to be a powerful tool and has been used by several authors in the past decade, see e.g. [1, 2, 5, 8, 23, 24, 26]. Given a parameter $q \in (0,1)$, let $\{G_a\}_{a\in\mathcal{A}}$ be mutually independent geometric random variables with respective parameters $1-q^a$, i.e. for all $a \in \mathcal{A}$, and for all non-negative integers k, we have $\mathsf{P}(G_a = k) = (1-q^a)q^{ak}$. As was observed by Fristedt [13] the joint distribution of the random variables $\{M_a\}_{a\in\mathcal{A}}$ (with respect to P_n) is exactly *equal* to the *conditional* distribution of the $\{G_a\}_{a\in\mathcal{A}}$, where the event conditioned on is that $\sum_{a\in\mathcal{A}} aG_a = n$. This is true for any choice of the parameter q. Hence the parameter $q = q_n$ can be chosen in such a way that asymptotic estimates as $n \to \infty$ are tractable. Analogous methods have been used (with Poisson distributions in place of geometric distributions) in the context of random permutations [28]. As a matter of fact, it is quite common that the distribution of the components of random combinatorial structures are independent random variables conditioned on the sum of the sizes being fixed (see [1] for more information and references).

We write E_n for expected values computed using P_n. We likewise write P_q and E_q for computations with the independent geometric variables. We use Fristedt's device, as well as some additional probabilistic and analytic arguments to derive a limit theorem for $\mathbf{M}_n$.

For $n \geq 1$ we choose the parameter $q = q_n = \exp[-C_Q n^{-d/d+1}]$, with a specific value of the constant,

*The second author was supported in part by NSA Grant MSPF-04G-054

†Department of Mathematics, Drexel University, Philadelphia, PA 19104, email: wgoh@math.drexel.edu

‡Department of Mathematics, Drexel University, Philadelphia, PA 19104, email: phitczenko@math.drexel.edu

namely C_Q is equal to

$$(1.2)\quad \frac{1}{a_d^{1/(d+1)}}\left(\frac{\zeta(1+d^{-1})\Gamma(1+d^{-1})}{d}\right)^{d/(d+1)}$$

(A reason for that particular choice will become clear in Section 2.) We also set the normalizing constants $\mu_n = n^{\frac{d}{d+1}}/C_Q$. Finally, we let $r_1, r_2, \ldots, r_d$ be the (complex) roots of $Q(z)$ and for $j = 1, 2\ldots$, $\alpha_1(j), \ldots, \alpha_{d-1}(j)$ be those (complex) roots of $Q(z) - Q(j)$ that are not equal to j.

Our aim is to sketch a proof of the following:

THEOREM 1.1. *For any positive real number x, we have*

$$(1.3)\quad \lim_{n\to\infty} P_n(\frac{\mathbf{M}_n}{\mu_n} \le x) = P(W_Q \le x),$$

where W_Q is a random variable whose characteristic function is

$$(1.4)\quad \phi_Q(t) = \prod_{k\ge 1}\frac{1}{1-it/Q(k)}$$

and whose density is

$$(1.5)\quad f_Q(x) = \sum_{j=1}^{\infty}(-1)^{j-1}e^{-Q(j)x}\frac{Q'(j)}{(j-1)!}\frac{\prod_{m=1}^{d-1}\Gamma(1-\alpha_m(j))}{\prod_{m=1}^{d}\Gamma(1-r_m)},$$

for $x > 0$.

Our argument will be broken into several steps. We will first show that the distribution of $\mathbf{M}_n$ is close to that of a sum of independent geometric random variables with suitably chosen parameters. It then follows that the limiting distribution has a characteristics function given by (1.4). This means that W_Q is equidistributed with the infinite sum of independent exponential random variables with parameters $Q(k)$, $k = 1, 2\ldots$. We will carry out the Fourier inversion and, after some simplifications, will show that the density is given by (1.5).

Finally, we will point out that specific choices of Q lead to distributions that have already appeared in several, quite different, contexts. We will briefly mention a few such instances in the last section of the abstract.

2 Reduction to the case of independent summands

We consider a doubly infinite array $\{G_{n,a} : a \in \mathcal{A},\ n \ge 1\}$, where $G_{n,a}$ is a geometric random variable with parameter $1 - q_n^a$, and for each $n \ge 1$, $\{G_{n,a}\ a \in \mathcal{A}\}$ are independent.

As follows from an observation by Fristedt [13] (see also [5]), for any $x > 0$, and $n \ge 1$ we have

$$\begin{aligned}\mathsf{P}_n(\frac{\mathbf{M}_n}{\mu_n} \le x) &= \mathsf{P}_q(\sum_{a\in\mathcal{A}} G_{n,a} \le \mu_n x | \sum_{a\in\mathcal{A}} aG_{n,a} = n)\\ &= \frac{\mathsf{P}_q(\sum_{a\in\mathcal{A}} G_{n,a} \le \mu_n x, \sum_{a\in\mathcal{A}} aG_{n,a} = n)}{\mathsf{P}_q(\sum_{a\in\mathcal{A}} aG_{n,a} = n)}. \quad (2.6)\end{aligned}$$

The argument, whose details we omit here, is to show that the events in the numerator of (2.6) are asymptotically independent. This is done by arguing that for a suitably chosen sequence (k_n), each of the sums

$$\sum_{a\in\mathcal{A}} aG_{n,a} = \sum_{j=1}^{\infty} Q(j)G_{n,Q(j)},$$

and

$$\sum_{a\in\mathcal{A}} G_{n,a} = \sum_{j=1}^{\infty} G_{n,Q(j)},$$

can be split in two pieces ($j \le k_n$ and $j > k_n$) so that the dominant contribution to the value of $\sum_{j\ge1} G_{n,Q(j)}$ comes from indices $j \le k_n$ while the dominant contribution to $\sum_{j\ge1} Q(j)G_{n,Q(j)}$ comes from $j > k_n$.

This can be seen by extending the line of argument that was originally developed by Fristedt. First, in order to asymptotically maximize the denominator in (2.6) we choose q_n so that $\mathsf{E}_q(\sum_{a\in\mathcal{A}} aG_{n,a}) \sim n$. Since G's are geometric this means that we want

$$\sum_{a\in\mathcal{A}} a\frac{q^a}{1-q^a} = \sum_{j=1}^{\infty} Q(j)\frac{q^{Q(j)}}{1-q^{Q(j)}} \sim n.$$

After calculations and change of variables $y = Q(x)\ln(1/q)$ we get (the same computations were carried out in the case $Q(x) = \binom{x+d}{d}$ in [13] for $d = 1$ and in [5] for $d \ge 2$)

$$\begin{aligned}\mathsf{E}_q\sum_{a\in\mathcal{A}} aG_{n,a} &= \sum_{\ell=1}^{\infty} Q(\ell)\frac{q^{Q(\ell)}}{1-q^{Q(\ell)}}\\ &\sim \int_0^{\infty} Q(x)\frac{e^{-Q(x)\ln(1/q)}}{1-e^{-Q(x)\ln(1/q)}}dx\\ &= \frac{1}{\ln^2(1/q)}\int_0^{\infty}\frac{y}{Q'(Q^{-1}(y/\ln(1/q)))}\frac{e^{-y}}{1-e^{-y}}dy\\ &\sim \frac{1}{da_d^{1/d}\ln^{1+1/d}(1/q)}\int_0^{\infty} y^{1/d}\frac{e^{-y}}{1-e^{-y}}dy,\end{aligned}$$

which, using [20, formula 3.411-7] leads to $q_n = \exp(-C_Q/n^{d/(d+1)})$, where C_Q is given by (1.2).

By the same argument, if $k_n = o(n^{1/(d+1)})$, then

$$\mathsf{E}_q \sum_{j\le k_n} Q(j)G_{n,Q(j)} = \sum_{j=1}^{k_n} Q(j)\frac{e^{-Q(j)\ln(1/q)}}{1-e^{-Q(j)\ln(1/q)}}$$

$$\sim \frac{1}{\ln^2(q^{-1})}\int_0^{Q(k_n)\ln(q^{-1})} \frac{y}{Q'\left(Q^{-1}\left(\frac{y}{\ln(q^{-1})}\right)\right)}\frac{e^{-y}dy}{1-e^{-y}}$$

$$\sim n\int_0^{k_n^d/n^{d/(d+1)}} \frac{y^{1/d}e^{-y}}{1-e^{-y}}dy$$

$$\sim cn\frac{k_n}{n^{1/(d+1)}}, \tag{2.7}$$

which is of lower order than the expected value of the full sum $\sum_{j=1}^{\infty} Q(j)G_{n,Q(j)}$. (Here and throughout the rest of this abstract $c = c_Q$ is an unspecified constant which depends on Q only. Its value is unimportant and may change from one use to another.)

Similar reasoning applied to $\sum G_{n,Q(j)}$ gives,

$$\mathsf{E}_q \sum_{j>k_n} G_{n,Q(j)} \sim \int_{k_n}^{\infty} \frac{e^{-Q(x)\ln(1/q)}}{1-e^{-Q(x)\ln(1/q)}}dx$$

$$\sim \frac{1}{\ln(q^{-1})}\int_{Q(k_n)\ln(q^{-1})}^{\infty} \frac{e^{-y}}{(1-e^{-y})}\frac{dy}{Q'\left(Q^{-1}\left(\frac{y}{\ln(q^{-1})}\right)\right)}$$

$$\sim cn^{1/(d+1)}\int_{\frac{ck_n^d}{n^{d/(d+1)}}}^{\infty} \frac{y^{\frac{1}{d}-1}e^{-y}}{1-e^{-y}}dy$$

$$\sim cn^{1/(d+1)}\cdot\frac{n^{(d-1)/(d+1)}}{k_n^{d-1}} \sim c\frac{n^{d/(d+1)}}{k_n^{d-1}},$$

and

$$\mathsf{E}_q\sum_{j=1}^{\infty} G_{n,Q(j)} \sim cn^{d/(d+1)},$$

Hence, as long as $k_n \to \infty$, the expected value of the sum restricted to $j > k_n$ is of smaller order than that of the full sum. Thus one expects the contribution of $\sum_{j>k_n} G_{n,Q(j)}$ to be negligible. Similarly, (2.7) suggests that the contribution of the truncated sum $\sum_{j\le k_n} Q(j)G_{n,Q(j)}$ to the full sum is negligible.

Of course, the very fact that the two pieces have expectation of lower order than the respective sums over all of natural numbers, does not by itself suffice to argue that they may be dropped from the sums without affecting their magnitude. But both of these expressions, being sums of independent random variables are heavily concentrated about their expected value. This can be quantified by using methods based on exponential inequalities. When these estimates are carried out, we are left with two truncated sums over the disjoint sets of indices, plus error terms that are negligible even when divided by the denominator of (2.6). The gain is that, unlike the original sums, the truncated sums are independent and thus can be handled with relative ease. Since the estimates are very explicit, it is easy to trace down conditions that k_n's need to satisfy and it turns out that one may choose

$$k_n = \Theta(n^{\alpha}), \quad \text{where} \quad 0 < \alpha < \frac{1}{2(d+1)}. \tag{2.8}$$

The upshot of all this is that, for any $x > 0$,

$$\lim_{n\to\infty} \mathsf{P}_n(\mathbf{M}_n/\mu_n \le x) = \lim_{n\to\infty} \mathsf{P}_q(\sum_{j=1}^{k_n} G_{n,Q(j)}/\mu_n \le x).$$

Since the jth summand on the right-hand side above is geometric with parameters $1-q^{Q(j)}$ its characteristic function is

$$\mathsf{E}_q e^{itG_{n,Q(j)}/\mu_n} = \frac{1-q^{Q(j)}}{1-e^{it/\mu_n}q^{Q(j)}}.$$

Since $j \le k_n = o(n^{1/(2(d+1))})$, $q = \exp(-C_Q/n^{d/(d+1)})$, and $\mu_n = n^{d/(d+1)}/C_Q$ using basic approximations we further have

$$\frac{1-q^{Q(j)}}{1-e^{it/\mu_n}q^{Q(j)}} = \frac{1-\exp(-Q(j)C_Q/n^{d/(d+1)})}{1-\exp(it/\mu_n - Q(j)C_Q/n^{d/(d+1)})}$$

$$= \frac{C_Q\frac{Q(j)}{n^{d/(d+1)}} + O\left(\frac{Q^2(k)}{n^{2d/(d+1)}}\right)}{C_Q\frac{Q(j)}{n^{d/(d+1)}} - \frac{it}{\mu_n} + O\left(\frac{Q^2(k)}{n^{2d/(d+1)}}\right)}$$

$$= \frac{1}{1-\frac{it}{Q(j)}}\left(1+O\left(\frac{Q^2(k)}{n^{2d/(d+1)}}\right)\right).$$

Hence, by independence of the summands, for j's in our range, we get

$$\begin{aligned}
\phi_n(t) &:= \mathsf{E}e^{\frac{t}{\mu_n}\sum_{j=1}^{k_n} G_{n,Q(j)}} \\
&= \prod_{j=1}^{k_n}\left(\frac{1}{1-\frac{it}{Q(j)}}\left(1+O\left(\frac{Q^2(j)}{n^{2d/(d+1)}}\right)\right)\right) \\
&= \left(\prod_{j=1}^{k_n}\frac{1}{1-\frac{it}{Q(j)}}\right)\left(1+O\left(\frac{Q^2(k_n)}{n^{2d/(d+1)}}\right)\right)^{k_n} \\
&= \left(\prod_{j=1}^{k_n}\frac{1}{1-\frac{it}{Q(j)}}\right)\left(1+O\left(\frac{k_nQ^2(k_n)}{n^{2d/(d+1)}}\right)\right). \qquad (2.9)
\end{aligned}$$

Since $k_nQ^2(k_n) = O(k_n^{2d+1})$, for k_n satisfying (2.8), the "big Oh" term in (2.9) goes to zero. Thus we conclude that $\phi_n(t)$ converge pointwise to $\phi_Q(t)$ given by (1.4).

3 Fourier Inversion

In this section we derive an explicit representation for the density of the limit distribution. By inversion formula, this density is given for $x > 0$ by

$$f_Q(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-itx} \phi_Q(t) dt.$$

If we regard t as a complex variable, then $\phi_Q(t)$ is a meromorphic function with simple poles at $-iQ(j)$, $j \geq 1$. One may then apply residue theory to evaluate the integral and deduce that, for $x > 0$,

$$f_Q(x) = \sum_{j=1}^{\infty} e^{-Q(j)x} Q(j) \prod_{\ell \neq j} \frac{Q(\ell)}{Q(\ell) - Q(j)}. \tag{3.10}$$

Specifically, for a large natural number n, we let

$$N = \frac{Q(n) + Q(n+1)}{2}$$

and we let C_N to be a clockwise oriented rectangular contour in the complex plane with vertices at $\pm N$, $\pm N - iN$. We consider the contour integral

$$\frac{1}{2\pi} \oint_{C_N} e^{-itx} \phi_Q(t) dt, \quad x > 0,$$

and we show that the integrals along three non-real sides of C_N approach zero as $n \to \infty$. Since the residue of

$$e^{-itx} \prod_{\ell \geq 1} \frac{1}{1 - \frac{it}{Q(\ell)}},$$

at $t = -iQ(j)$ is

$$iQ(j) e^{-xQ(j)} \prod_{\ell \neq j} \frac{Q(\ell)}{Q(\ell) - Q(j)},$$

using the residue theorem (taking into account the orientation of C_N) and passing to the limit with n we derive (3.10).

4 Simplification

The expression on the right–hand side of (3.10) may be further transformed by evaluating the product. Specifically, we will show that

$$\prod_{\ell \neq j} \frac{Q(\ell)}{Q(\ell) - Q(j)} = \frac{Q'(j)(-1)^{j+1}}{Q(j)(j-1)!} \frac{\prod_{t=1}^{d-1} \Gamma(1 - \alpha_t(j))}{\prod_{t=1}^{d} \Gamma(1 - r_t)}, \tag{4.11}$$

where $r_1, r_2, \ldots, r_d$ are the roots of $Q(z)$ and $\alpha_1(j), \ldots, \alpha_{d-1}(j)$ are those roots of $Q(z) - Q(j)$ that are not equal to j.

To this end we write

$$\prod_{\ell \neq j} \frac{Q(\ell)}{Q(\ell) - Q(j)} = \lim_{s \to j} \left(\prod_{\ell \neq j} \frac{Q(\ell)}{Q(\ell) - Q(s)} \right)$$

$$= \lim_{s \to j} \left(\frac{Q(j) - Q(s)}{Q(j)} \prod_{\ell \geq 1} \frac{Q(\ell)}{Q(\ell) - Q(s)} \right). \tag{4.12}$$

We factor both $Q(\ell)$ and $Q(\ell) - Q(s)$ as a product of linear terms

$$Q(\ell) = a_d \prod_{m=1}^{d} (\ell - r_m)$$

$$Q(\ell) - Q(s) = a_d \prod_{m=1}^{d} (\ell - \alpha_m(s)).$$

We now use the following formula [33, Chaptex XII, Sec. 12.13]: if $a_1 + \ldots + a_r = b_1 + \ldots + b_r$ then

$$\prod_{n=1}^{\infty} \frac{(n - a_1) \cdot \ldots \cdot (n - a_r)}{(n - b_1) \cdot \ldots \cdot (n - b_r)} = \prod_{m=1}^{r} \frac{\Gamma(1 - b_m)}{\Gamma(1 - a_m)}.$$

Applying this to the product in (4.12) we obtain

$$\prod_{\ell \neq j} \frac{Q(\ell)}{Q(\ell) - Q(j)} = \lim_{s \to j} \left(\frac{Q(j) - Q(s)}{Q(j)} \prod_{m=1}^{d} \frac{\Gamma(1 - \alpha_m(s))}{\Gamma(1 - r_m)} \right).$$

We know that exactly one of $\alpha_m(s)$'s is equal to s and we assume without loss of generality that $\alpha_d(s) = s$. Since

$$\frac{1}{Q(j)\Gamma(1 - r_d)} \prod_{m=1}^{d-1} \frac{\Gamma(1 - \alpha_m(s))}{\Gamma(1 - r_m)} \tag{4.13}$$

is continuous at $s = j$ we only need to be concerned with

$$\lim_{s \to j} ((Q(j) - Q(s))\Gamma(1 - \alpha_d(s)))$$

$$= \lim_{s \to j} \left(\frac{Q(j) - Q(s)}{j - s} (j - s)\Gamma(1 - s) \right)$$

$$= \lim_{s \to j} \left(\frac{Q(j) - Q(s)}{j - s} (j - s)(-s)\Gamma(-s) \right)$$

$$= jQ'(j) \lim_{s \to j} ((s - j)\Gamma(-s)).$$

Since the residue of $\Gamma(z)$ at $-j$ is $(-1)^j / j!$ this last limit is $(-1)^{j-1} / j!$ which combined with (4.13) and (4.12) proves (4.11).

5 Further remarks

In this section we briefly discuss a few cases that are of special interest.

(i) One such case, $Q(z) = \frac{z(z+1)}{2}$ arises naturally in the context of iterated functions and the coalescent [14], [18]. There the characteristic function is

$$\begin{aligned} \phi(t) &= \prod_{m=2}^{\infty} \frac{\binom{m}{2}}{\binom{m}{2} - it} \\ &= \prod_{m=1}^{\infty} \frac{1}{1 - it/\binom{m+1}{2}}. \end{aligned} \tag{5.14}$$

In (4.11) we have $r_1 = 0, r_2 = -1$, $\alpha_1(k) = -k - 1$, and consequently

$$\begin{aligned} &\prod_{\ell \neq k} \frac{Q(\ell)}{Q(\ell) - Q(k)} \\ &= \frac{\frac{2k+1}{2}}{\frac{k(k+1)}{2}} \frac{\Gamma(1+1+k)}{\Gamma(1-0)\Gamma(1+1)} \frac{(-1)^{k-1}}{(k-1)!} \\ &= (-1)^{k-1}(2k+1). \end{aligned}$$

Hence inversion of (5.14) yields the probability density function

$$f(x) = \sum_{k=2}^{\infty} e^{-\binom{k}{2}x} \binom{k}{2} (-1)^k (2k-1), \ x > 0.$$

This latter density is well-known in certain circles, and is generally attributed to Kingman [18], [19]. See the unpublished manuscript [14] for a derivation that is related to the arguments in this paper.

(ii) Similarly, for the special case $Q(x) = x^3$, we consider the number of parts of random partitions of n into parts that are cubes. For this particular class of partitions, Richmond [25] provided asymptotic estimates for the moments. Carleman's conditions are satisfied, therefore the limit distribution is uniquely determined. However Richmond did not invert, and we are not aware of any previous work in which the limiting density is calculated. In fact, the density has an interesting form: for $x > 0$,

$$f(x) = 3 \sum_{k=1}^{\infty} e^{-k^3 x} \frac{(-1)^{k+1} k^3 c_k}{k!},$$

where

$$\begin{aligned} c_k &= \Gamma(1 - ke^{2\pi i/3})\Gamma(1 - ke^{-2\pi i/3}) \\ &= |\Gamma(1 - ke^{2\pi i/3})|^2. \end{aligned}$$

(iii) The next case corresponds to $Q(x) = \binom{x+d}{d}$, for some fixed positive integer d. (Since $d = 1$ does not impose any restrictions we will assume $d \geq 2$. Also, $d = 2$ was a special case discussed in (i).) Such partitions are in bijection with partitions with dth differences non-negative. Some of their properties (although the limiting distribution of the number of parts was not one of them) were studied in [5]. We have $r_m = -m$, $m = 1, \ldots, d$ and thus

$$\prod_{m=1}^{d} \Gamma(1 - r_m) = \prod_{m=1}^{d} m!.$$

Further,

$$Q'(x) = \frac{1}{d!} \sum_{j=1}^{d} \prod_{\substack{1 \leq \ell \leq d \\ \ell \neq j}} (x + \ell) = Q(x) \sum_{j=1}^{d} \frac{1}{x+j},$$

so that

$$Q'(k) = Q(k) \left(H_{k+d} - H_k \right),$$

where H_n is the nth harmonic number. Although there does not seem to be a simple way of handling the roots of $Q(x) - Q(k)$ in the general case, the case $d = 3$ can be managed (as can be any other polynomial of degree 3 since it leads to a quadratic equation after factoring $(x - k)$) and gives the density

$$\sum_{k=1}^{\infty} (-1)^{k-1} e^{-\binom{k+3}{3}x} \binom{k+3}{3} \frac{(H_{k+3} - H_k) f_k}{2! \cdot 3! \cdot (k-1)!},$$

where $f_k = \left| \Gamma\left(4 + \frac{k}{2} + \frac{i}{2}\sqrt{3k^2 + 12k + 8}\right) \right|^2$ and $x > 0$.

If $d = 4$ then $\binom{x+4}{4} - \binom{k+4}{4}$ has a real root $-k-5$ (in addition to k, of course) and the limiting density for $x > 0$ is given by

$$\sum_{k=1}^{\infty} (-1)^{k-1} e^{-\binom{k+4}{4}x} \binom{k+4}{4} \frac{(H_{k+4} - H_k) g_k (k+5)!}{2! \cdot 3! \cdot 4! \cdot (k-1)!},$$

where $g_k = \left| \Gamma\left(\frac{7}{2} + \frac{i}{2}\sqrt{4k^2 + 20k + 15}\right) \right|^2$.

(iv) Finally, we would like to conclude by observing that the choice $Q(x) = x^2$ corresponds to yet another interesting situation that arises in quite a different context. In view of (1.5) and (4.11) the probability density function corresponding to this choice is

$$f(x) = 2 \sum_{k=1}^{\infty} (-1)^{k+1} k^2 e^{-k^2 x}, \quad x > 0.$$

Up to a scaling this is the density of the maximum of the Brownian bridge process or the Brownian meandering process (see [7, Section 3] and also [10, 9] for more details and information). Further interesting connections along with many more references to the literature are discussed in a relatively recent survey paper [3].

Distribution function corresponding to the last density is given by

$$\begin{aligned} F(x) &= 1 - 2\sum_{k=1}^{\infty}(-1)^{k+1}e^{-k^2x} \\ &= \sum_{k=-\infty}^{\infty}(-1)^k e^{-k^2x}. \end{aligned}$$

Changing variables, $x \to 2x^2$ and differentiating gives a density

$$4\sum_{k=-\infty}^{\infty}(-1)^{k-1}k^2xe^{-2k^2x^2},$$

which is the density of the Kolmogorov-Smirnov statistic used to measure the discrepancy between the true and empirical distribution functions. We refer the reader to [21] for the translation of the original work of Kolmogorov and to [4, Chapter 2, Sec. 13] for a detailed exposition.

Acknowledgment: We would like to thank Eric Schmutz for several helpful discussions, suggestions, and comments.

References

[1] R. Arratia, A. D. Barbour, and S. Tavare, *Logarithmic Combinatorial structures: A Probabilistic Approach*, EMS Monographs in Mathematics, EMS (2003).

[2] R. Arratia and S. Tavare, *Independent Poisson process approximations for random combinatorial structures*, Adv. in Math., 104 (1994), pp. 90–154.

[3] P. Biane, J. Pitman, and M. Yor, *Probability laws related to Jacobi theta and Riemann zeta functions, and Brownian excursions*, Bull. Amer. Math. Soc., 38 (2001), pp. 435–465.

[4] P. Billingsley, *Convergence of Probability Measures*, Wiley (1968).

[5] E. R. Canfield, S. Corteel, and P. Hitczenko, *Partitions with rth differences non-negative*, Adv. in Appl. Math., 27 (2001), pp. 298-317.

[6] N. R. Chaganty and J. Sethuraman, *Strong large deviation and local limit theorems*, Ann. Probab., 21 (1993), pp. 1671–1690.

[7] K. L. Chung, *Excursions in Brownian motion*, Ark. Mat., 14 (1976), pp. 155–177.

[8] S. Corteel, B. Pittel, C. D. Savage, and H. S. Wilf, *On the multiplicity of parts in a random partition*, Random Structures Algorithms, 14 (1999), pp. 185-197.

[9] R. T. Durrett and D. L. Iglehart, *Functionals of Brownian meander and Brownian excursion*, Ann. Probab., 5 (1977), pp. 129–135.

[10] R. T. Durrett, D. L. Iglehart, and D. R. Miller, *Weak convergence to Brownian meander and Brownian excursion*, Ann. Probab., 5 (1977), pp. 117–129.

[11] P. Erdös and J. Lehner, *The distribution of the number of summands in the partition of a positive integer*, Duke Math. J., 8 (1941), pp. 335–345.

[12] P. Erdös and P. Turan, *On some general problems in the statistical theory of partitions*, Acta Arith., 18 (1971), pp. 53–62.

[13] B. Fristedt, *The structure of random partitions of large integers*, Trans. Amer. Math. Soc., 337 (1993), pp. 703–735.

[14] W. M. Y. Goh, P. Hitczenko, and E. Schmutz, *Iterating random functions on a finite set*, unpublished manuscript available at *arxiv: math.CO/0207276.*

[15] G. H. Hardy and S. R. Ramanujan, *Asymptotic formulae in combinatory analysis*, Proc. London Math. Soc., 17 (1918), pp. 75–118.

[16] C. B. Haselgrove and H. N. V Temperly, *Asymptotic formulae in the theory of partitions*, Math. Proc. Cambridge Philos. Soc., 50 (1954), pp. 225–241.

[17] H. K. Hwang, *Limit theorems for the number of summands in integer partitions*, J. Combin. Theory Ser. A, 96 (2001), pp. 89–126.

[18] J. F. C. Kingman, *On the genealogy of large populations. Essays in statistical science.* J. Appl. Probab., 19A (1982), pp. 27–43.

[19] J. F. C. Kingman, *The Coalescent*, Stochastic Process. Appl., 13 (1982), pp. 235–248.

[20] I. S. Gradshteyn, I. M. Ryzhik, *Table of Integrals, Series, and Products*, 4th ed., Academic Press, New York, 1965.

[21] A. N. Kolmogorov, *On the empirical determination of a distribution, in: Breakthroughs in Statistics, vol. II*, S. Kotz. and N. L. Johnson, Eds. Springer, 1992, pp. 106 – 113.

[22] D. Lee, *The asymptotic distribution of the number of summands in unrestricted Λ partitions*, Acta Arith., 65 (1993), pp. 29–43.

[23] B. Pittel, *On the likely shape of the random Ferrers diagram*, Adv. in Appl. Math., 18 (1997), pp. 432–488.

[24] B. Pittel, *Confirming two conjectures about integer partitions*, J. Combin. Theory Ser. A, 88 (1999), pp. 123–135.

[25] L. B. Richmond, *The moments of partitions II*, Acta Arith., 28 (1975/76), pp. 229-243.

[26] D. Romik, *Partitions of n into $tn^{1/2}$ parts*, Europ. J. Combin., 26 (2005), pp. 1–17.

[27] K. F. Roth and G. Szekeres, *Some asymptotic formulae in the theory of partitions*, Quarterly J. Math. Oxford

Ser., 5 (1954), pp. 241–259.

[28] L. A. Shepp and S. P. Lloyd, *Ordered cycle lengths in a random permutation*, Trans. Amer. Math. Soc., 12 (1966), pp. 340–357.

[29] M. Szalay and P. Turan, *On some problems of the statistical theory of partitions with applications to characters of the symmetric group I*, Acta Math. Acad. Sci. Hungar., 29 (1977), pp. 361–379.

[30] M. Szalay and P. Turan, *On some problems of the statistical theory of partitions with applications to characters of the symmetric group II*, Acta Math. Acad. Sci. Hungar., 29 (1977), pp. 381–392.

[31] M. Szalay and P. Turan, *On some problems of the statistical theory of partitions with applications to characters of the symmetric group III*, Acta Math. Acad. Sci. Hungar., 32 (1978), pp. 129–155.

[32] G. Szekeres, *An asymptotic formula in the theory of partitions II*, Quarterly J. Math. Oxford Ser., 4 (1953), pp. 96–111.

[33] E. T. Whittaker and G. N. Watson, *A Course of Modern Analysis*, 4th Ed., reprinted. Cambridge University Press, 1963.

This page intentionally left blank