1st Symposium on Simplicity in Algorithms

SOSA 2018, January 7–10, 2018, New Orleans LA, USA
Co-located with *The 29th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*

Edited by

Raimund Seidel

OASICS

Editors

Raimund Seidel
FR Informatik, Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
rseidel@cs.uni-saarland.de

ACM Classification 1998
G.4.1 Algorithm design and analysis

ISBN 978-3-95977-064-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-064-4.

Publication date
January, 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.SOSA.2018.0

OASIcs – OpenAccess Series in Informatics

ISSN 2190-6807

http://www.dagstuhl.de/oasics

Contents

Preface

Although simplicity in algorithms is usually appreciated it frequently does not find the recognition it deserves. Typically program committees prefer papers pursuing new domains or papers reporting improvements in various performance guarantees over papers that "just" simplify known results or that report algorithms that are simpler but slower. The present workshop was set up to counteract this attitude making simplicity and elegance in the design and analysis of algorithms its main objectives.

The response from the community was overwhelming. In spite of rather short notice we received 81 submissions. It quickly became clear that simplicity and elegance are not absolute notions but depend on the field, the background, and the sophistication of the researchers. The work of the program committee was therefore not easy. In the end we selected 15 papers, which you find in these proceedings and which we hope you will enjoy.

I would like to thank members of the program committee for putting this much work in this workshop and I would like to thank the members of the steering committee for conceiving this workshop and making it happen.

Raimund Seidel
Program Committee Chair

1st Symposium on Simplicity in Algorithms (SOSA 2018).
Editor: Raimund Seidel
OASICS Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organisation

Program Committee

Keren Censor-Hillel, *Technion|*
Edith Cohen, *Google, Mountain View*
Edith Elkind, *University of Oxford*
Jeremy Fineman, *Georgetown University*
Mohsen Ghaffari, *ETH Zürich*
David Karger, *MIT*
Richard Karp, *University of California, Berkeley*
Valerie King, *University of Victoria*
Dániel Marx, *Hungarian Academy of Sciences*
Moni Naor, *Weizmann Institute of Science*
Raimund Seidel (Chair), *Universität des Saarlandes*
Robert Tarjan, *Princeton University*
Virginia Vassilevska Williams, *MIT*
David Williamson, *Cornell University*
David Woodruff, *Carnegie Mellon University*
Uri Zwick, *Tel Aviv University*

Steering Committee

Michael A. Bender, *Stony Brook University*
David Karger, *MIT*
Tsvi Kopelowitz, *University of Waterloo*
Seth Pettie, *University of Michigan*
Robert Tarjan, *Princeton University*
Mikkel Thorup, *University of Copenhagen*

List of Authors

Divesh Aggarwal
CQT and Department of Computer Science, NUS
Singapore
dcsdiva@nus.edu.sg

Petra Berenbrink
Fachbereich Informatik, Universität Hamburg
Hamburg, Germany
berenbrink@informatik.uni-hamburg.de

Allan Borodin
Department of Computer Science, University of Toronto
Toronto, Canada
bor@cs.toronto.edu

Yixin Cao
Department of Computing, Hong Kong Polytechnic University
Hong Kong, China
yixin.cao@polyu.edu.hk

Deeparnab Chakrabarty
Department of Computer Science, Dartmouth College
Hanover NH, USA
deeparnab@dartmouth.edu

Timothy M. Chan
Dept. of Computer Science, University of Illinois
Urban IL 61801, USA
tmc@illinois.edu

Chandra Chekuri
Dept. of Computer Science, University of Illinois
Urban IL 61801, USA
chekuri@illinois.edu

Michael B. Cohen
MIT
32 Vassar Street, Cambridge, MA 02139, USA
micohen@mit.edu

Talya Eden
School of Electrical Engineering, Tel Aviv University
Tel Aviv, Israel
talyaa01@gmail.com

Mark Idleman
Dept. of Computer Science, University of Illinois
Urban IL 61801, USA
midleman2@illinois.edu

Klaus Jansen
Christian-Albrechts-Universität
Kiel, Germany
kj@informatik.uni-kiel.de

T.S. Jayram
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120, USA
jayram@us.ibm.com

Dominik Kaaser
Fachbereich Informatik, Universität Hamburg
Hamburg, Germany
dominik.kaaser@uni-hamburg.de

Sanjeev Khanna
Department of Computer and Information Science, University of Pennsylvania
Philadelphia PA, USA
sanjeev@cis.upenn.edu

Peter Kling
Fachbereich Informatik, Universität Hamburg
Hamburg, Germany
peter.kling@uni-hamburg.de

Tsvi Kopelowitz
University of Waterloo
Waterloo, Canada
kopelot@gmail.com

Andrew McGregor
College of Computer and Information Sciences, University of Massachusetts
Amherst MA, USA
mcgregor@cs.umass.edu

Jelani Nelson
Harvard John A. Paulson School of Engineering and Applied Sciences
29 Oxford Street, Cambridge, MA 02138, USA
minilek@seas.harvard.edu

Alantha Newman
CNRS–Université Grenoble Alpes
F-38000, Grenoble, France
alantha.newman@grenoble-inp.fr

Lena Otterbach
Fachbereich Informatik, Universität Hamburg
Hamburg, Germany
otterbach@informatik.uni-hamburg.de

Denis Pankratov
Department of Computer Science, University of Toronto
Toronto, Canada
denisp@cs.toronto.edu

Ely Porat
University of Bar-Ilan
Ramat Gan, Israel
porately@cs.biu.ac.il

Lars Rohwedder
Christian-Albrechts-Universität
Kiel, Germany
lro@informatik.uni-kiel.de

Will Rosenbaum
School of Electrical Engineering, Tel Aviv University
Tel Aviv, Israel
will.rosenbaum@gmail.com

Thapanapong Rukkanchanunt
Chiang Mai University
239 Huay Kaew Road, Chiang Mai 50200, Thailand
thapanapong.r@cmu.ac.th

Amirali Salehi-Abari
Faculty of Business and IT, University of Ontario Institute of Technology
Oshawa, Canada
abari@uoit.ca

Noah Stephens-Davidowitz
New York University
New York NY, USA
noahsd@gmail.com

Sofya Vorotnikova
College of Computer and Information Sciences, University of Massachusetts
Amherst MA, USA
svorotni@cs.umass.edu

R. Ryan Williams
CSAIL & EECS, MIT
Cambridge, MA, USA
rrw@mit.edu

A Naive Algorithm for Feedback Vertex Set*

Yixin Cao

Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
yixin.cao@polyu.edu.hk

Abstract

Given a graph on n vertices and an integer k, the feedback vertex set problem asks for the deletion of at most k vertices to make the graph acyclic. We show that a greedy branching algorithm, which always branches on an undecided vertex with the largest degree, runs in single-exponential time, i.e., $O(c^k \cdot n^2)$ for some constant c.

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity, G.2.2 Graph Theory

Keywords and phrases greedy algorithm, analysis of algorithms, branching algorithm, parameterized computation, graph modification problem

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.1

1 Introduction

All graphs in this paper are undirected and simple. A graph G is given by its vertex set $V(G)$ and edge set $E(G)$, whose cardinalities will be denoted by n and m respectively. A set V_- of vertices is a *feedback vertex set* of graph G if $G - V_-$ is acyclic, i.e., being a forest. Given a graph G and an integer k, the feedback vertex set problem asks whether G has a feedback vertex set of at most k vertices.

The feedback vertex set problem was formulated from artificial intelligence, where a feedback vertex set is also called a *loop cutset*. For each instance of the constraint satisfaction problem one can define a constraint graph, and it is well known that the problem can be solved in polynomial time when the constraint graph is a forest [11]. Therefore, one way to solve the constraint satisfaction problem is to find first a minimum feedback vertex set of the constraint graph, enumerate all possible assignments on them, and then solve the remaining instance. Given an instance I of the constraint satisfaction problem on p variables, and a feedback vertex set V_- of the constraint graph, this approach can be implemented in $O(p^{|V_-|} \cdot |I|^{O(1)})$ time [7]. A similar application was found in Bayesian inference, also in artificial intelligence [18]; more updated material can be found in the thesis of Bidyuk [3].

The feedback vertex set problem is NP-hard [16]. The aforementioned approach for solving the constraint satisfaction problem only makes sense when $|V_-|$ is fairly small. This motivates the study of parameterized algorithms for the feedback vertex set problem, i.e., algorithms that find a feedback vertex set of size at most k in time $f(k) \cdot n^{O(1)}$. Since earlier 1990s, a chain of parameterized algorithms have been reported in literature; currently the best known $f(k)$ is 3.62^k [5, 15]. We refer to [5] for a complete list of published results. Instead of providing a new and improved algorithm, this paper considers a naive branching algorithm that *should have been discovered decades ago.*

* Supported in part by the National Natural Science Foundation of China (NSFC) under grant 61572414, the Hong Kong Research Grants Council (RGC) under grants 25202615 and 15201317, and the European Research Council (ERC) under grant 280152.

A trivial branching algorithm will work as follows. It picks a vertex and branches on either including it in the solution V_- (i.e., deleting it from G), or marking it "undeletable," until the remaining graph is already a forest. This algorithm however takes $O(2^n)$ time. A (rather informal) observation is that a vertex of a larger degree has a larger chance to be in a minimum feedback vertex set, thereby inspiring a two-phase greedy algorithm for the problem. If there are undecided vertices of degree larger than two after some preprocessing, then it always branches on an undecided vertex with the largest degree. Believe it or not, this greedy algorithm already beats most previous algorithms for this problem.

▶ **Theorem 1.1.** *The greedy algorithm can be implemented in $O(8^k \cdot n^2)$ time.*

The use of the observations on degrees in solving the feedback vertex set problem is quite natural. Indeed, the research on parameterized algorithms and that on approximation algorithms for the feedback vertex set problem have undergone a similar process. Early work used the cycle packing-covering duality, and hence ended with $O((\log k)^{O(k)} \cdot n^{O(1)})$-time parameterized algorithms [19] and $O(\log n)$-ratio approximation algorithms [8], respectively, while the first 2-approximation algorithm uses a similar greedy approach on high-degree vertices [1]. Indeed, all the four slightly different 2-approximation algorithms for this problem are based on similar degree observations [6, 12]. So is the quadratic kernel of Thomassé [20]. There is also an $O(4^k \cdot n)$-time randomized algorithm [2] based on this idea. Our greedy branching algorithm can be viewed as the de-randomization of this randomized algorithm.

For a reader familiar with parameterized algorithms of the feedback vertex set problem, Theorem 1.1 may sound somewhat surprising. Deterministic single-exponential algorithms for the feedback vertex set problem had been sedulously sought, before finally discovered in 2005. With so many different techniques, some very complicated, having been tried toward this end,[1] it is rather interesting that the goal can be achieved in such a naive way.

The significance of single-exponential algorithms for the feedback vertex set problem lies also in the theoretical interest, for which let us put it into context.

Together with the vertex cover problem (finding a set V_- of at most k vertices of a graph G such that $G - V_-$ is edgeless), the feedback vertex set problem is arguably the most studied parameterized problem. However, a simple $O(2^k \cdot (m + n))$-time algorithm for vertex cover was already known in 1980s [17]. For this difference there is a quick and easy explanation from the aspect of graph modification problems [16, 4]. Vertex deletion problems ask for the deletion of a minimum set of vertices from a graph to make it satisfy specific properties. The vertex cover problem and the feedback vertex set problem are precisely vertex deletion problems to, respectively, the edgeless graphs and acyclic graphs, i.e., forests. The obstruction (forbidden induced subgraph) for the edgeless graphs is an single edge, the simplest one that is nontrivial. On the other hand, the obstructions for forests are all cycles, which may be considered the simplest of all those infinite obstructions, for most of which single-exponential algorithms are quite nontrivial, if possible at all.

The problems vertex cover and feedback vertex set are also known as planar-$\mathcal{F}$-deletion problems, which, given a graph G, a set $\mathcal{F}$ of graphs of which at least one is planar, ask for a minimum set of vertices whose deletion make the graph H-minor-free for every $H \in \mathcal{F}$ [9]. They correspond to the cases with $\mathcal{F} = \{K_2\}$ and $\mathcal{F} = \{K_3\}$ respectively. Recently, Fomin et al. [10] and Kim et al. [14] showed that all planar-$\mathcal{F}$-deletion problems can be solved in single-exponential time. Yet another way to connect the vertex cover problem and the

[1] To date, the number of parameterized algorithms for feedback vertex set published in literature exceeds any other single problem, including the more famous vertex cover problem.

```
Algorithm naive-fvs($G, k, F$)
INPUT: a graph $G$, an integer $k$, and a set $F \subseteq V(G)$ inducing a forest.
OUTPUT: a feedback vertex set $V_- \subseteq V(G) \setminus F$ of size $\leq k$ or "NO."

0.    if $k < 0$ then return "NO"; if $V(G) = \emptyset$ then return $\emptyset$;
1.    if a vertex $v$ has degree less than two then
          return naive-fvs($G - \{v\}, k, F \setminus \{v\}$);
2.    if a vertex $v \in V(G) \setminus F$ has two neighbors in the same component of $G[F]$ then
          $X \leftarrow$ naive-fvs($G - \{v\}, k - 1, F$);
          return $X \cup \{v\}$;
3.    pick a vertex $v$ from $V(G) \setminus F$ with the maximum degree;
4.    if $d(v) = 2$ then
4.1.      $X \leftarrow \emptyset$;
4.2.      while there is a cycle $C$ in $G$ then
              take any vertex $x$ in $C \setminus F$;
              add $x$ to $X$ and delete it from $G$;
4.3.      if $|X| \leq k$ then return $X$; else return "NO";
5.    $X \leftarrow$ naive-fvs($G - \{v\}, k - 1, F$);          \\ case 1: $v \in V_-$.
      if $X$ is not "NO" then return $X \cup \{v\}$;
6.    return naive-fvs($G, k, F \cup \{v\}$).                \\ case 2: $v \notin V_-$.
```

Figure 1 A simple algorithm for feedback vertex set branching in a greedy manner. As we will see in the end of this section, we may terminate the recursive calls after certain number of steps. This leads to a search tree of bounded depth.

feedback vertex set problem is that a graph has treewidth zero if and only if it is edgeless, and treewidth at most one if and only if it is a forest.

2 The algorithm

There is no secret in our algorithm, which is presented in Figure 1, except the recursive form and an extra input F, the set of "undeletable" vertices. We say that (G, k, F), where $F \subseteq V(G)$ and the solution can only be picked from $V(G) \setminus F$, is an *extended instance*; note that to make such an extended instance nontrivial, F needs to induce a forest. (Indeed, the solution in the original loop cutset problem has to be selected from "allowed" vertices, which is exactly the case F comprising all vertices that are not allowed.) The algorithm can be viewed as two parts, the first (steps 1–4) applying some simple operations when the situation is simple and clear, while the second (steps 5 and 6) trying both possibilities on whether a vertex v is in a solution. The operations in the first part are called reductions in the parlance of parameterized algorithms. The three we use here are standard and well-known,[2] and their correctness is straightforward; see, e.g., [5].

[2] For the reader familiar with related algorithms, our reduction steps may seem slightly different from those in literature. First of all, unlike most algorithms for the problem, our algorithm does not involve multiple edges. We believe it is simpler to keep the graph simple. As a result, we are not able to eliminate all vertices of degree two: The common way to dispose of a vertex v of degree-2 is to delete v and add an edge between its two neighbors, so called *smoothening*. Smoothening a vertex whose two neighbors were already adjacent would introduce parallel edges. Noting that there always exists an optimal solution avoiding v, one may move v into F [5], but we prefer the current form because it is simpler and easier to analyze. It is also easier to be extended in Section 3.

▶ **Lemma 2.1.** *Calling algorithm* `naive-fvs` *with $(G, k, \emptyset)$ solves the instance (G, k) of the feedback vertex set problem.*

Proof. The two termination conditions in step 0 are clearly correct. For each recursive call in steps 1 and 2, we show that the original instance is a yes-instance if and only if the new instance is a yes-instance. Note that no vertex is moved to F in these two steps. In step 1, the vertex v is not in any cycle, and hence it can be avoided by any solution. In step 2, there is a cycle consisting of the vertex v and vertices in F (any path connecting these two vertices in $G[F]$), and hence any solution has to contain v.

To argue the correctness of step 4, we show that the solution found in step 4 is optimal. Let c be the number of components in G and ℓ the size of optimal solutions. Note that every vertex in a solution has degree two, and hence after deleting ℓ vertices the graph has $n - \ell$ vertices and at least $m - 2\ell$ edges. Moreover, deleting vertices from an optimal solution will not decrease the number of components of the graph, we have $(n - \ell) - (m - 2\ell) \geq c$. Hence, $\ell \geq m - n + c$, and showing $|X| = m - n + c$ would finish the task. Deleting a vertex of degree 2 from a cycle never increases the number of components. Also note that a graph of c components contains a cycle if and only if it has more than $n - c$ edges. Therefore, the while loop in step 4 would be run exactly $m - n + c$ iterations: After deleting $m - n + c$ vertices, each of degree two when deleted, the remaining graph has $2n - m - c$ vertices and $2n - m - 2c$ edges, which has to be a forest of c trees.

The last two steps are trivial: If there is a solution containing v, then it is found in step 5; otherwise, step 6 always gives the correct answer. ◀

We now analyze the running time of the algorithm, which is simple but nontrivial. The execution of the algorithm can be described as a search tree in which each node corresponds to two extended instances of the problem, the *entry instance* and the *exit instance*. The entry instance of the root node is $(G, k, \emptyset)$. The exit instance of a node is the one after steps 0–2 have been exhaustively applied on the entry instance. If step 5 is further called, then two children nodes are generated, with entry instances $(G - \{v\}, k - 1, F)$ and $(G, k, F \cup \{v\})$ respectively. (Note that the second child may not be explored by the algorithm, but this is not of our concern.) A leaf node of the search tree returns either a solution or "NO."

It is clear that each node can be processed in polynomial time, and thus the focus of our analysis is to bound the number of nodes in the search tree. Since the tree is binary, it suffices to bound its depth. We say that a path from the root of the search tree to a leaf node is an *execution path*. Let us fix an arbitrary execution path in the search tree of which the leaf node returns a solution V_-, and let F' denote all the vertices moved into F by step 6 in this execution path. The length of this execution path is at most $|V_-| + |F'|$: Each non-leaf node puts at least one vertex to V_- or F. We are allowed to put at most k vertices into V_-, i.e., $|V_-| \leq k$, and hence our task in the rest of this section is to bound $|F'|$.

Let us start from some elementary facts on trees. Any tree T satisfies

$$\sum_{v \in V(T)} d(v) = 2|E(T)| = 2|V(T)| - 2 \qquad \text{and} \qquad \sum_{v \in V(T)} (d(v) - 2) = -2.$$

Let L denote the set of leaves of T, and V_3 the set of vertices of degree at least three. If $V(T) \geq 2$, then $|L| \geq 2$ and

$$-2 = \sum_{v \in L} (d(v) - 2) + \sum_{v \in V(T) \setminus L} (d(v) - 2) = \sum_{v \in L} (-1) + \sum_{v \in V_3} (d(v) - 2) = \sum_{v \in V_3} (d(v) - 2) - |L|.$$

Hence

$$\sum_{v\in V_3}(d(v)-2) = |L|-2. \tag{1}$$

The implication of (1) for our problem is that the more large-degree vertices (V_3) in the final forest $G - V_-$, the more leaves (L) it has. Every vertex $u \in F'$ will be in the forest. Since its original degree is at least three, either its degree is decreased to two or less, or there must be some leaves produced to "balance the equation (1)." On the other hand, however, every vertex has degree at least two when u is moved to F. Therefore, if it is the second case, the leaves have to be "produced" in later steps. The requirement of degree decrements is decided by the degree of u, and can be satisfied by vertices deleted later, whose degrees cannot be larger than that of u. This informal observation would enable us to derive the desired lower bound on $|F'|$.

The following invariants will be used in our formal analysis.

Invariant 1 : During the algorithm, the degree of no vertex can increase.

Invariant 2 : When a recursive call is made in step 5 or 6, there is no vertex of degree 0 or 1 in the graph.

This algorithm never directly deletes any edge, and thus the degree of a vertex decreases only when some of its neighbors are deleted from the graph,—we are talking about the degree in the whole graph G, so moving a vertex to F does not change the degree of any vertex. In particular, only steps 1, 2, 4, and 5 can decrease the degree of vertices. By Invariant 2, after a vertex is moved to F, step 1 cannot be called before step 2, 4, or 5. In other words, the degree of a vertex in F decreases only after some vertex put into V_-. We can *attribute* them to vertices V_- as follows.

For a vertex $v \in V_- \cup F'$, we use $d^*(v)$ to denote the degree of v at the moment it is deleted from the graph and put into V_- (step 2, 4, or 5) or moved into F (step 6). Note that $d_G(v) \geq d^*(v)$ by Invariant 1, and $d^*(v) \geq 3$ when $v \in F'$. Let x_1, x_2, …, $x_{|V_-|}$ be the vertices in V_-, in the order of them being put into V_-, and let (G_i, k_i, F_i) be the exit instance in the node of the search tree corresponding to x_i.

▶ **Definition 2.2.** We say that the decrements of the degree of a vertex $u \in F'$ from $d^*(u)$ to 2 are *effective*, and an effective decrement is *incurred by* $x_i \in V_-$ if it happens between deleting x_i and x_{i+1}, or after deleting $x_{|V_-|}$ if $i = |V_-|$. Let $\delta(u, x_i)$ denote the number of effective decrements of u incurred by x_i.

Note that $\delta(u, x_i)$ may be larger than 1. It is worth stressing that we do not count the degree decrements of u before it is moved into F. Therefore, $\delta(u, x_i)$ can be positive only when u is in F when x_i is deleted, i.e., $u \in F_i$ and hence $d_{G_i}(u) \geq 2$:

$$\delta(u,x_i) = \begin{cases} d_{G_i}(u) - \max\{d_{G_{i+1}}(u), 2\} & \text{when } u \in F_i, \\ 0 & \text{otherwise.}\end{cases}$$

▶ **Proposition 2.3.** *For any $u \in F'$ and $x_i \in V_-$, if $\delta(u, x_i) > 0$ then $d^*(u) \geq d^*(x_i)$.*

First, we bound the total number of effective decrements incurred by x_i for each $x_i \in V_-$.

▶ **Lemma 2.4.** *For each $x_i \in V_-$, it holds $\sum_{u\in F'} \delta(u, x_i) \leq d^*(x_i)$.*

Proof. Recall that all effective decrements incurred by x_i happen after deleting x_i from G_i. If $d_{G_i}(v) > 2$ for every vertex $v \in N_{G_i}(x_i)$, then the deletion of x_i will not make the degree of any vertex smaller than two. Therefore, step 1 will not be called before putting the next vertex into V_-. The degree of each vertex in $N_{G_i}(x_i)$ decreases by one, and the total number of effective decrements incurred by x_i is thus at most $d^*(x_i)$.

In the rest $d_{G_i}(v) = 2$ for some $v \in N_{G_i}(x_i)$, and it becomes 1 with the deletion of x_i. This decrement is not effective, but it will trigger step 1, which may subsequently lead to effective decrements. Let d denote the number of degree-2 neighbors of x_i in G_i. After the deletion of x_i, all of them have degree one, and there is no other vertex having degree one in $G_i - \{x_i\}$ (Invariant 2). We consider the application of step 1, and let x be the vertex deleted. If the only neighbor of x has degree two when this step is executed, then its degree becomes 1 after the deletion of x, and hence the number of degree-1 vertices is not changed. Otherwise, there is one less vertex of degree 1 but there may be one effective decrement (only when the only neighbor of x is in F and has degree at least three). Therefore, when step 1 is no longer applicable, the total number of effective decrements is at most $d^*(x_i) - d + d = d^*(x_i)$. ◀

We are now ready to bound the number of calls of step 6 made in this execution path, i.e., the size of F', by the size of V_-. This is exactly the place the greedy order of branching plays the magic.

▶ **Lemma 2.5.** *In an execution path that leads to a solution, $|F'| \leq 3|V_-|$.*

Proof. Since this execution path leads to a solution, all vertices must be deleted from the graph at the end of the path. In the algorithm, a vertex in F can only be deleted from the graph in step 1, when the degree of the vertex has to be one or zero. On the other hand, $d^*(u) \geq 3$. Thus, all the $d^*(u) - 2$ effective decrements must have happened on this vertex, i.e., $\sum_{v \in V_-} \delta(u, v) = d^*(u) - 2$. Putting everything together, we have

$$\begin{aligned}
|V_-| = \sum_{v \in V_-} 1 &= \sum_{v \in V_-} \frac{d^*(v)}{d^*(v)} \\
&\geq \sum_{v \in V_-} \frac{1}{d^*(v)} \sum_{u \in F'} \delta(u, v) && \text{(Lemma 2.4)} \\
&= \sum_{v \in V_-} \sum_{u \in F'} \frac{\delta(u, v)}{d^*(v)} \\
&\geq \sum_{v \in V_-} \sum_{u \in F'} \frac{\delta(u, v)}{d^*(u)} && \text{(Proposition 2.3)} \\
&= \sum_{u \in F'} \frac{1}{d^*(u)} \sum_{v \in V_-} \delta(u, v) \\
&= \sum_{u \in F'} \frac{d^*(u) - 2}{d^*(u)} \\
&\geq \sum_{u \in F'} \frac{1}{3} && (d^*(u) \geq 3) \\
&= \frac{|F'|}{3},
\end{aligned}$$

and the proof is complete. ◀

▶ **Theorem 2.6.** *Algorithm* `naive-fvs` *can be implemented in $O(16^k \cdot n^2)$ time to decide whether a graph G has a feedback vertex set of size at most k.*

Proof. If the input graph G has a feedback vertex set of size at most k, then there must be an execution path that returns a solution, and by Lemma 2.5, the length of this path is at most $4k$. Otherwise, all execution paths return "NO," disregard of their lengths. Therefore, we can terminate every execution path after it has put $3k$ vertices into F by returning "NO" directly. The new search tree would then have depth at most $4k$. Clearly, the processing in each node can be done in $O(n^2)$ time. This gives the running time $O(2^{4k+1} \cdot n^2) = O(16^k \cdot n^2)$. ◀

3 An improved running time

It is long (but *not* well) known that if the maximum degree of a graph is at most three, then a minimum feedback vertex set can be found in polynomial time [13, 21]. This can be extended to the setting that the degree bound holds only for the undecided vertices i.e., vertices in $V(G) \setminus F$.

▶ **Lemma 3.1** ([5]). *Given a graph G and a set F of vertices such that every vertex in $V(G)\setminus F$ has degree at most three, there is a polynomial-time algorithm for finding a minimum set $V_- \subseteq V(G) \setminus F$ such that $G - V_-$ is a forest.*

Therefore, we can change step 4 of algorithm `naive-fvs` to the following:

```
4.    if $d(v) \leq 3$ then
4.1.      call Lemma 3.1 to find a minimum solution $X$;
4.2.      if $|X| \leq k$ then return $X$; else return "NO";
```

Now that $d^*(u) \geq 4$ for each vertex $u \in F'$, as a result, in the last inequality in the proof of Lemma 2.5, we can use $(d^*(u) - 2)/d^*(u) \geq 2/4 = 1/2$, which implies $|F'| \leq 2|V_-|$. We can hence terminate every execution path after it has put $2k$ vertices into F by returning "NO" directly, and the algorithm would then run in $O(8^k \cdot n^{O(1)})$ time, as announced in Theorem 1.1.

We conclude this paper by pointing out that the analysis is not tight. The inequalities in the proof of Lemma 2.5 could be tight only when $d^*(v) = 4$ for every vertex $v \in V_- \cup F$, and more importantly, all the degree decrements incurred by putting a vertex to V_- are effective. If such a graph exists,—we may assume without loss of generality that it does not contains any vertex of degree two or less,—then all its vertices have degree four, and all neighbors of a vertex $x \in V_-$ are in F. But in such a graph there should be a different solution, and note that our algorithm explores the subtree rooted at the child node made by step 6 *only if* all the leaves in the other subtree (rooted at the node made by step 5) have returned "NO."

Acknowledgment. The author would like to thank O-joung Kwon and Saket Saurabh for pointing out a mistake in the introduction of a previous version.

References

1 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

2 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234, 2000. doi:10.1613/jair.638.

3 Bozhena Petrovna Bidyuk. *Exploiting Graph Cutsets for Sampling-Based Approximations in Bayesian Networks*. PhD thesis, University of California, Irvine, 2006.

4 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

5 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015. A preliminary version appeared in SWAT 2010. doi:10.1007/s00453-014-9904-6.

6 Fabián A. Chudak, Michel X. Goemans, Dorit S. Hochbaum, and David P. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters*, 22(4-5):111–118, 1998. doi:10.1016/S0167-6377(98)00021-2.

7 Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987. doi:10.1016/0004-3702(87)90002-6.

8 Paul Erdős and Lajos Pósa. On the maximal number of disjoint circuits of a graph. *Publicationes Mathematicae Debrecen*, 9:3–12, 1962.

9 Michael R. Fellows and Michael A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35(3):727–739, 1988. doi:10.1145/44483.44491.

10 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation and optimal FPT algorithms. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

11 Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982. doi:10.1145/322290.322292.

12 Toshihiro Fujito. A note on approximation of the vertex cover and feedback vertex set problems - unified approach. *Information Processing Letters*, 59(2):59–63, 1996. doi:10.1016/0020-0190(96)00094-4.

13 Merrick L. Furst, Jonathan L. Gross, and Lyle A. McGeoch. Finding a maximum-genus graph imbedding. *Journal of the ACM*, 35(3):523–534, 1988. doi:10.1145/44483.44485.

14 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2015. doi:10.1145/2797140.

15 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

16 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. Preliminary versions independently presented in STOC 1978. doi:10.1016/0022-0000(80)90060-4.

17 Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1984.

18 Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, 1988.

19 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In Prosenjit Bose and Pat Morin, editors, *ISAAC*, volume 2518 of *LNCS*, pages 241–248. Springer, 2002. doi:10.1007/3-540-36136-7_22.

20 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2):32.1–32.8, 2010. A preliminary version appeared in SODA 2009. `doi:10.1145/1721837.1721848`.

21 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988. `doi:10.1016/0012-365X(88)90226-9`.

A Note on Iterated Rounding for the Survivable Network Design Problem[*]

Chandra Chekuri[1] and Thapanapong Rukkanchanunt[†2]

1 **Dept. of Computer Science, University of Illinois, Urbana, 61801, USA**
`chekuri@illinois.edu`

2 **Chiang Mai University, 239 Huay Kaew Road, Chiang Mai 50200, Thailand**
`thapanapong.r@cmu.ac.th`

Abstract

In this note we consider the survivable network design problem (SNDP) in undirected graphs. We make two contributions. The first is a new counting argument in the iterated rounding based 2-approximation for edge-connectivity SNDP (EC-SNDP) originally due to Jain [10]. The second contribution is to make some connections between hypergraphic version of SNDP (Hypergraph-SNDP) introduced in [17] and edge and node-weighted versions of EC-SNDP and element-connectivity SNDP (Elem-SNDP). One useful consequence is a 2-approximation for Elem-SNDP that avoids the use of set-pair based relaxation and analysis.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems: Computations on Discrete Structures

Keywords and phrases survivable network design, iterated rounding, approximation, element connectivity

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.2

1 Introduction

The *survivable network design problem* (SNDP) is a fundamental problem in network design and has been instrumental in the development of several algorithmic techniques. The input to SNDP is a graph $G = (V, E)$ and an integer requirement $r(uv)$ between each unordered pair of nodes uv. The goal is to find a minimum-cost subgraph H of G such that for each pair uv, the connectivity in H between u and v is at least $r(uv)$. We use $r_{\max}$ to denote $\max_{uv} r(uv)$, the maximum requirement. We restrict attention to undirected graphs in this paper. There are several variants depending on whether the costs are on edges or on nodes, and whether the connectivity requirement is edge, element or node connectivity. Unless otherwise specified we will assume that G has edge-weights $c : E \to \mathbb{R}_+$. We refer to the three variants of interest based on edge, element and vertex connectivity as EC-SNDP, Elem-SNDP and VC-SNDP. All of them are NP-Hard and APX-hard to approximate even in very special cases.

The seminal work of Jain [10] obtained a 2-approximation for EC-SNDP via the technique of iterated rounding that was introduced in the same paper. A 2-approximation for Elem-SNDP was obtained, also via iterated rounding, in [7, 5]. For VC-SNDP the current best approximation bound is $O(r_{\max}^3 \log |V|)$ [6]; it is also known from hardness results in [2] that the approximation bound for VC-SNDP must depend polynomially on $r_{\max}$ under standard hardness assumptions.

[*] Work on this paper supported in part by NSF grants CCF-1016684, CCF-1319376 and CCF-1526799.

[†] The second author was an undergraduate student in the Dept. of Computer Science at Univ. of Illinois when he worked on results of this paper.

Our contribution: In this note we revisit the iterated rounding framework that yields a 2-approximation for EC-SNDP and Elem-SNDP. The framework is based on arguing that for a class of covering problems, a basic feasible solution to an LP relaxation for the covering problem has a variable of value at least $\frac{1}{2}$. This variable is then rounded up to 1 and the residual problem is solved inductively. A key fact needed to make this iterative approach work is that the residual problem lies in the same class of covering problems. This is ensured by working with the class of skew-supermodular (also called weakly-supermodular) requirement functions which capture EC-SNDP as a special case. The proof of existence of an edge with large value in a basic feasible solution for this class of requirement functions has two components. The first is to establish that a basic feasible solution is characterized by a laminar family of sets in the case of EC-SNDP (and set pairs in the case of Elem-SNDP). The second is a counting argument that uses this characterization to obtain a contradiction if no variable is at least $\frac{1}{2}$. The counting argument of Jain [10] has been simplified and streamlined in subsequent work via fractional token arguments [1, 13]. These arguments have been applied for several related problems for which iterated rounding has been shown to be a powerful technique; see [12]. The fractional token argument leads to short and slick proofs. At the same time we feel that it is hard to see the intuition behind the argument. Partly motivated by pedagogical reasons, in this note, we provide a different counting argument along with a longer explanation. The goal is to give a more *combinatorial* flavor to the argument. We give this argument in Section 2.

The second part of the note is on Elem-SNDP. A 2-approximation for this problem has been derived by generalizing the iterated rounding framework to a set-pair based relaxation [7, 5]. The set-pair based relaxation and arguments add substantial notation to the proofs although one can see that there are strong similarities to the proofs used in EC-SNDP. The notational overhead limits the ability to teach and understand the proof for Elem-SNDP. Interestingly, in a little noticed paper, Zhao, Nagamochi and Ibaraki [17] defined a generalization of EC-SNDP to hypergraphs which we refer to as Hypergraph-SNDP. They observed that Elem-SNDP can be easily reduced to Hypergraph-SNDP in which the only non-zero weight hyperedges are of size 2 (regular edges in a graph). The advantage of this reduction is that one can derive a 2-approximation for Elem-SNDP by essentially appealing to the same argument as for EC-SNDP with a few minor details. We believe that this is a useful perspective. Second, there is a simple and well-known connection between node-weighted network design in graphs and network design problems on hypergraphs. We explicitly point these connections which allows us to derive some results for Hypergraph-SNDP. Section 3 describes these connections and results.

This note assumes that the reader has some basic familiarity with previous literature on SNDP and iterated rounding.

2 Iterated rounding for EC-SNDP

The 2-approximation for EC-SNDP is based on casting it as a special case of covering a skew-supermodular requirement function by a graph. We set up the background now. Given a finite ground set V an integer valued set function $f : 2^V \to \mathbb{Z}$ is skew-supermodular if for all $A, B \subseteq V$ one of the following holds:

$$\begin{aligned} f(A) + f(B) &\leq f(A \cap B) + f(A \cup B) \\ f(A) + f(B) &\leq f(A - B) + f(B - A) \end{aligned}$$

Given an edge-weighted graph $G = (V, E)$ and a skew-supermodular requirement function $f : 2^V \to \mathbb{Z}$, we can consider the problem of finding the minimum-cost subgraph $H = (V, F)$

of G such that H covers f; that is, for all $S \subseteq V$, $|\delta_F(S)| \geq f(S)$. Here $\delta_F(S)$ is the set of all edges in F with one endpoint in S and the other outside. Given an instance of EC-SNDP with input graph $G = (V, E)$ and edge-connectivity requirements $r(uv)$ for each pair uv, we can model it by setting $f(S) = \max_{u \in S, v \notin S} r(uv)$. It can be verified that f is skew-supermodular. The important aspect of skew-supermodular functions that make them well-suited for the iterated rounding approach is the following.

▶ **Lemma 1** ([10]). *Let $G = (V, E)$ be a graph and $f : 2^V \to \mathbb{Z}$ be a skew-supermodular requirement function, and $F \subseteq E$ be a subset of edges. The residual requirement function $g : 2^V \to \mathbb{Z}$ defined by $g(S) = f(S) - |\delta_F(S)|$ for each $S \subseteq V$ is also skew-supermodular.*

Although the proof is standard by now we will state it in a more general way.

▶ **Lemma 2.** *Let $f : 2^V \to \mathbb{Z}$ be a skew-supermodular requirement function and let $h : 2^V \to \mathbb{Z}_+$ be a symmetric submodular function. Then $g = f - h$ is a skew-supermodular function.*

Proof. Since h is submodular we have that for all $A, B \subseteq V$,

$$h(A) + h(B) \geq h(A \cup B) + h(A \cap B).$$

Since h is also symmetric it is posi-modular which means that for all $A, B \subseteq V$,

$$h(A) + h(B) \geq h(A - B) + h(B - A).$$

Note that h satisfies both properties for each A, B. It is now easy to check that $f - h$ is skew-supermodular. ◀

Lemma 1 follows from Lemma 2 by noting that the cut-capacity function $|\delta_F| : 2^V \to \mathbb{Z}_+$ is submodular and symmetric in undirected graphs. We also note that the same property holds for the more general setting when G is a hypergraph.

The standard LP relaxation for covering a function by a graph is described below where there is variable $x_e \in [0, 1]$ for each edge $e \in E$.

$$\begin{aligned} \min \sum_{e \in E} c_e x_e & & & \\ \sum_{e \in \delta(S)} x_e & \geq & f(S) & \qquad S \subset V \\ x_e & \in & [0,1] & \qquad e \in E \end{aligned}$$

The technical theorem that underlies the 2-approximation for EC-SNDP is the following.

▶ **Theorem 3** ([10]). *Let f be a non-trivial[1] skew-supermodular function. In any basic feasible solution $\overline{x}$ to the LP relaxation of covering f by a graph G there is an edge e such that $\overline{x}_e \geq \frac{1}{2}$.*

To prove the preceding theorem it suffices to focus on basic feasible solutions $\overline{x}$ that are fully fractional; that is, $\overline{x}_e \in (0, 1)$ for all e. For a set of edges $F \subseteq E$ let $\chi(F) \in \{0, 1\}^{|E|}$ denote the characteristic vector of F; that is, a $|E|$-dimensional vector that has a 1 in each position corresponding to an edge $e \in F$ and a 0 in all other positions. Theorem 3 is built upon the following characterization of basic feasible solutions and is shown via uncrossing arguments.

[1] We use the term non-trivial to indicate that there is at least one set $S \subset V$ such that $f(S) > 0$.

▶ **Lemma 4** ([10]). *Let $\overline{x}$ be a fully-fractional basic feasible solution to the the LP relaxation. Then there is a laminar family of vertex subsets $\mathcal{L}$ such that $\overline{x}$ is the unique solution to the system of equalities*

$$x(\delta(S)) = f(S) \qquad S \in \mathcal{L}.$$

In particular this also implies that $|\mathcal{L}| = |E|$ and that the vectors $\chi(\delta(S))$, $S \in \mathcal{L}$ are linearly independent.

The second part of the proof of Theorem 3 is a counting argument that relies on the characterization in Lemma 4. The rest of this section describes a counting argument which we believe is slightly different from the previous ones in terms of the main invariant. The goal is to derive it organically from simpler cases.

With every laminar family we can associate a rooted forest. We use terminology for rooted forests such as leaves and roots as well as set terminology. We refer to a set $C \in \mathcal{L}$ as a child of a set S if $C \subset S$ and there is no $S' \in \mathcal{L}$ such that $C \subset S' \subset S$; If C is the child of S then S is the parent of C. Maximal sets of $\mathcal{L}$ correspond to the roots of the forest associated with $\mathcal{L}$.

2.1 Counting Argument

The proof is via contradiction where we assume that $0 < \overline{x}_e < \frac{1}{2}$ for each $e \in E$. We call the two nodes incident to an edge as the endpoints of the edges. We say that an endpoint u *belongs* to a set $S \in \mathcal{L}$ if u is the minimal set from $\mathcal{L}$ that contains u.

We consider the simplest setting where $\mathcal{L}$ is a collection of disjoint sets, in other words, all sets are maximal. In this case the counting argument is easy. Let $m = |E| = |\mathcal{L}|$. For each $S \in \mathcal{L}$, $f(S) \geq 1$ and $\overline{x}(\delta(S)) = f(S)$. If we assume that $\overline{x}_e < \frac{1}{2}$ for each e, we have $|\delta(S)| \geq 3$ which implies that each S contains at least 3 distinct endpoints. Thus, the m disjoint sets require a total of $3m$ endpoints. However the total number of endpoints is at most $2m$ since there are m edges, leading to a contradiction.

Now we consider a second setting where the forest associated with $\mathcal{L}$ has k leaves and h internal nodes but each internal node has at least two children. In this case, following Jain, we can easily prove a weaker statement that $\overline{x}_e \geq 1/3$ for some edge e. If not, then each leaf set S must have four edges leaving it and hence the total number of endpoints must be at least $4k$. However, if each internal node has at least two children, we have $h < k$ and since $h + k = m$ we have $k > m/2$. This implies that there must be at least $4k > 2m$ endpoints since the leaf sets are disjoint. But m edges can have at most $2m$ endpoints. Our assumption on each internal node having at least two children is obviously a restriction. So far we have not used the fact that the vectors $\chi(\delta(S)), S \in \mathcal{L}$ are linearly independent. We can handle the general case to prove $\overline{x}_e \geq 1/3$ by using the following lemma.

▶ **Lemma 5** ([10]). *Suppose C is a unique child of S. Then there must be at least two endpoints in S that belong to S.*

Proof. If there is no endpoint that belongs to S then $\delta(S) = \delta(C)$ but then $\chi(\delta(S))$ and $\chi(\delta(C))$ are linearly dependent. Suppose there is exactly one endpoint that belongs to S and let it be the endpoint of edge e. But then $\overline{x}(\delta(S)) = \overline{x}(\delta(C)) + \overline{x}_e$ or $\overline{x}(\delta(S)) = \overline{x}(\delta(C)) - \overline{x}_e$. Both cases are not possible because $\overline{x}(\delta(S)) = f(S)$ and $\overline{x}(\delta(C)) = f(C)$ where $f(S)$ and $f(C)$ are positive integers while $\overline{x}_e \in (0, 1)$. Thus there are at least two end points that belong to S. ◀

Using the preceding lemma we prove that $\overline{x}_e \geq 1/3$ for some edge e. Let k be the number of leaves in $\mathcal{L}$ and h be the number of internal nodes with at least two children and let ℓ be the number of internal nodes with exactly one child. We again have $h < k$ and we also have $k + h + \ell = m$. Each leaf has at least four endpoints. Each internal node with exactly one child has at least two end points which means the total number of endpoints is at least $4k + 2\ell$. But $4k + 2\ell = 2k + 2k + 2\ell > 2k + 2h + 2\ell > 2m$ and there are only $2m$ endpoints for m edges. In other words, we can ignore the internal nodes with exactly one child since there are two endpoints in such a node/set and we can effectively charge one edge to such a node.

We now come to the more delicate argument to prove the tight bound that $\overline{x}_e \geq \frac{1}{2}$ for some edge e. Our main contribution is to show an invariant that effectively reduces the argument to the case where we can assume that $\mathcal{L}$ is a collection of leaves. This is encapsulated in the claim below which requires some notation. Let $\alpha(S)$ be the number of sets of $\mathcal{L}$ contained in S including S itself. Let $\beta(S)$ be the number of edges whose *both* endpoints lie inside S. Recall that $f(S)$ is the requirement of S.

▶ **Claim.** *For all $S \in \mathcal{L}$, $f(S) \geq \alpha(S) - \beta(S)$.*

Assuming that the claim is true we can do an easy counting argument. Let $R_1, R_2, \ldots, R_h$ be the maximal sets in $\mathcal{L}$ (the roots of the forest). Note that $\sum_{i=1}^{h} \alpha(R_i) = |\mathcal{L}| = m$. Applying the claim to each R_i and summing up,

$$\sum_{i=1}^{h} f(R_i) \geq \sum_{i=1}^{h} \alpha(R_i) - \sum_{i=1}^{h} \beta(R_i) \geq m - \sum_{i=1}^{h} \beta(R_i).$$

Note that $\sum_{i=1}^{h} f(R_i)$ is the total requirement of the maximal sets. And $m - \sum_{i=1}^{h} \beta(R_i)$ is the total number of edges that cross the sets $R_1, \ldots, R_h$. Let E' be the set of edges crossing these maximal sets. Now we are back to the setting with h disjoint sets and E' edges with $\sum_{i=1}^{h} f(R_i) \geq |E'|$. This easily leads to a contradiction as before if we assume that $\overline{x}_e < \frac{1}{2}$ for all $e \in E'$. Formally, each set R_i requires $> 2f(R_i)$ edges crossing it from E' and therefore R_i contains at least $2f(R_i) + 1$ endpoints of edges from E'. Since $R_1, \ldots, R_h$ are disjoint the total number of endpoints is at least $2\sum_i f(R_i) + h$ which is strictly more than $2|E'|$.

Thus, it remains to prove the claim which we do by inductively starting at the leaves of the forest for $\mathcal{L}$.

Case 1: S is a leaf node. We have $f(S) \geq 1$ while $\alpha(S) = 1$ and $\beta(S) = 0$ which verifies the claim.

Case 2: S is an internal nodes with k children $C_1, C_2, \ldots, C_k$. See Figure 1 for the different types of edges that are relevant. E_{cc} is the set of edges with end points in two different children of S. E_{cp} be the set of edges that cross exactly one child but do not cross S. E_{po} be the set of edges that cross S but do not cross any of the children. E_{co} is the set of edges that cross both a child and S. This notation is borrowed from [15].

Let $\gamma(S)$ be the number of edges whose both endpoints belong to S but not to any child of S. Note that $\gamma(S) = |E_{cc}| + |E_{cp}|$.

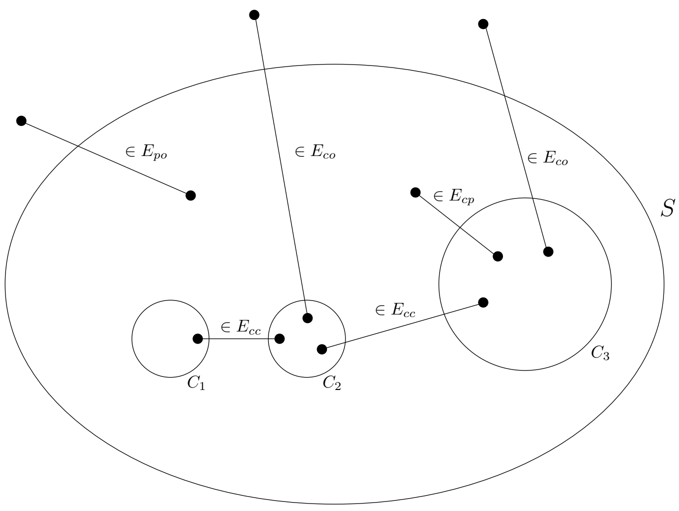

Figure 1 S is an internal node with several children. Different types of edges that play a role. p refers to parent set S, c refer to a child set and o refers to outside.

Then,

$$\begin{aligned}
\beta(S) &= \gamma(S) + \sum_{i=1}^{k} \beta(C_i) \\
&\geq \gamma(S) + \sum_{i=1}^{k} \alpha(C_i) - \sum_{i=1}^{k} f(C_i) \qquad (1) \\
&= \gamma(S) + \alpha(S) - 1 - \sum_{i=1}^{k} f(C_i)
\end{aligned}$$

(1) follows by applying the inductive hypothesis to each child. From the preceding inequality, to prove that $\beta(S) \geq \alpha(S) - f(S)$ (the claim for S), it suffices to show the following inequality.

$$\gamma(S) \geq \sum_{i=1}^{k} f(C_i) - f(S) + 1. \qquad (2)$$

The right hand side of the above inequality can be written as:

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 = \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e - \sum_{e \in E_{po}} x_e + 1. \qquad (3)$$

We consider two subcases.

Case 2.1: $\gamma(S) = 0$. This implies that E_{cc} and E_{cp} are empty. Since $\chi(\delta(S))$ is linearly independent from $\chi(\delta(C_1)), \ldots, \chi(\delta(C_k))$, we must have that E_{po} is not empty and hence $\sum_{e \in E_{po}} x_e > 0$. Therefore, in this case,

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 = \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e - \sum_{e \in E_{po}} x_e + 1 = - \sum_{e \in E_{po}} x_e + 1 < 1.$$

Since the left hand side is an integer, it follows that $\sum_{i=1}^{k} f(C_i) - f(S) + 1 \leq 0 = \gamma(S)$.

Case 2.2: $\gamma(S) \geq 1$. Recall that $\gamma(S) = |E_{cc}| + |E_{cp}|$.

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 = \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e - \sum_{e \in E_{po}} x_e + 1 \leq \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e + 1$$

By our assumption that $\overline{x}_e < \frac{1}{2}$ for each e, we have $\sum_{e \in E_{cc}} 2x_e < |E_{cc}|$ if $|E_{cc}| > 0$, and similarly $\sum_{e \in E_{cp}} x_e < |E_{cp}|/2$ if $|E_{cp}| > 0$. Since $\gamma(S) = |E_{cc}| + |E_{cp}| \geq 1$ we conclude that

$$\sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e < \gamma(S).$$

Putting together we have

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 \leq \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e + 1 < \gamma(S) + 1 \leq \gamma(S)$$

as desired.

This completes the proof of the claim.

3 Connections between Hypergraph-SNDP, EC-SNDP and Elem-SNDP

Zhao, Nagamochi and Ibaraki [17] considered the extension EC-SNDP to hypergraphs. In a hypergraph $G = (V, \mathcal{E})$ each edge $e \in \mathcal{E}$ is a subset of V. The degree d of a hypergraph is $\max_{e \in \mathcal{E}} |e|$. Graphs are hypergraphs of degree 2. Given a set of hyperedges $F \subseteq \mathcal{E}$ and a vertex subset $S \subset V$, we use $\delta_F(S)$ to denote the set all of all hyperedges in F that have at least one endpoint in S and at least one endpoint in $V \setminus S$. It is well-known that $|\delta_F| : 2^V \to \mathbb{Z}_+$ is a symmetric submodular function.

Hypergraph-SNDP is defined as follows. The input consists of an edge-weighted *hypergraph* $G = (V, \mathcal{E})$ and integer requirements $r(uv)$ for each vertex pair uv. The goal is to find a minimum-cost hypergraph $H = (V, \mathcal{E}')$ with $\mathcal{E}' \subseteq \mathcal{E}$ such that for all uv and all S that separate u, v (that is $|S \cap \{u, v\}| = 1$), we have $|\delta_{E'}(S)| \geq r(uv)$. Hypergraph-SNDP is a special case of covering a skew-supermodular requirement function by a hypergraph. It is clear that Hypergraph-SNDP generalizes EC-SNDP. Interestingly, [17] observed, via a simple reduction, that Hypergraph-SNDP generalizes Elem-SNDP as well. We now describe Elem-SNDP formally and briefly sketch the reduction from [17], and subsequently describe some implications of this connection.

In Elem-SNDP the input consists of an undirected edge-weighted graph $G = (V, E)$ with V partitioned into terminals T and non-terminals N. The “elements” are the edges and non-terimals, $N \cup E$. For each pair uv of terminals there is an integer requirement $r(uv)$, and the goal is to find a min-cost subgraph H of G such that for each pair uv of terminals there are $r(uv)$ element-disjoint paths from u to v in H. Note that element-disjoint paths can intersect in terminals. The notion of element-connectivity and Elem-SNDP have been useful in several settings in generalizing edge-connectivity problems while having some features of vertex connectivity. In particular, the current approximation for VC-SNDP relies on Elem-SNDP [6].

The reduction of [17] from Elem-SNDP to Hypergraph-SNDP is quite simple. It basically replaces each non-terminal $u \in N$ by a hyperedge. The reduction is depicted in Figure 2.

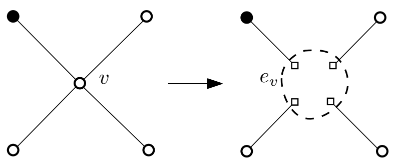

Figure 2 Reducing Elem-SNDP to Hypergraph-SNDP. Each non-terminal v is replaced by a hyperedge e_v by introducing dummy vertices on each edge incident to v. The original edges retain their cost while the new hyperedges are assigned a cost of zero.

The reduction shows that an instance of Elem-SNDP on G can be reduced to an instance of Hypergraph-SNDP on a hypergraph G' where the only hyperedges with non-zero weights in G' are the edges of the graph G. This motivates the definition of $d^+(G)$ which is the maximum degree of a hyperedge in G that has non-zero cost. Thus Elem-SNDP reduces to instances of Hypergraph-SNDP with $d^+ = 2$. In fact we can see that the same reduction proves the following.

▶ **Proposition 6.** *Node-weighted Elem-SNDP in which weights are only on non-terminals can be reduced in an approximation preserving fashion to Hypergraph-SNDP. In this reduction d^+ of the resulting instance of Hypergraph-SNDP is equal to Δ, the maximum degree of a non-terminal with non-zero weight in the instance of node-weighted Elem-SNDP.*

3.1 Reducing Elem-SNDP to problem of covering skew-supermodular functions by graphs

We saw that an instance of Elem-SNDP on a graph H can be reduced to an instance of Hypergraph-SNDP on a graph G where $d^+(G) = 2$. Hypergraph-SNDP on $G = (V, \mathcal{E})$ corresponds to covering a skew-supermodular function $f : 2^V \to \mathbb{Z}$ by G. Let $\mathcal{E} = F \uplus \mathcal{E}'$ where $\mathcal{E}'$ is the set of all hyperedges in G with degree more than 2; thus F is the set of all hyperedges of degree 2 and hence (V, F) is a graph. Since each edge in $\mathcal{E}'$ has zero cost we can include all of them in our solution, and work with the residual requirement function $g = f - |\delta_{\mathcal{E}'}|$. From Lemma 2 and the fact that the cut-capacity function of a hypergraph is also symmetric and submodular, g is a skew-supermodular function. Thus covering f by a min-cost sub-hypergraph of G can be reduced to covering g by a min-cost sub-graph of $G' = (V, F)$. We have already seen a 2-approximation for this in the context of EC-SNDP. The only issue is whether there is an efficient separation oracle for solving the LP for covering g by G'. This is a relatively easy exercise using flow arguments and we omit them. The main point we wish to make is that this reduction avoids working with set-pairs that are typically used for Elem-SNDP. It is quite conceivable that the authors of [17] were aware of this simple connection but it does not seem to have been made explicitly in their paper or in [16].

3.2 Approximating Hypergraph-SNDP

[17] derived a $d^+ H_{r_{\max}}$ approximation for Hypergraph-SNDP where $H_k = 1 + 1/2 + \ldots + 1/k$ is the k'th harmonic number. They obtain this bound via the augmentation framework for network design [9] and a primal-dual algorithm in each stage. In [16] they also observe that Hypergraph-SNDP can be reduced to Elem-SNDP via the following simple reduction. Given a hypergraph $G = (V, \mathcal{E})$ let $H = (V \cup N, E)$ be the standard bipartite graph representation of G where for each hyperedge $e \in \mathcal{E}$ there is a node $z_e \in N$; z_e is connected by edges in H

to each vertex $a \in e$. Let $r(uv)$ be the hyperedge connectivity requirement between a pair of vertices uv in the original instance of Hypergraph-SNDP. In H we label V as terminals and N as non-terminals. For any pair of vertices uv with $u, v \in V$, it is not hard to verify that the element-connectivity betwee u and v in H is the same as the hyperedge connectivity in G. See [16] for details. It remains to model the costs such that an approximation algorithm for element-connectivity in H can be translated into an approximation algorithm for hyperedge connectivity in G. This is straightforward. We simply assign cost to non-terminals in H; that is each node $z_e \in N$ corresponding to a hyperedge $e \in \mathcal{E}$ is assigned a cost equal to c_e. We obtain the following easy corollary.

▶ **Proposition 7.** *Hypergraph-SNDP can be reduced to node-weighted Elem-SNDP in an approximation preservation fashion.*

[16] do not explicitly mention the above but note that one can reduce Hypergraph-SNDP to (edge-weighted) Elem-SNDP as follows. Instead of placing a weight of c_e on the node z_e corresponding to the hyperedge $e \in \mathcal{E}$, they place a weight of $c_e/2$ on each edge incident to z_e. This transformation loses an approximation ratio of $d^+(G)/2$. From this they conclude that a β-approximation for Elem-SNDP implies a $d^+\beta/2$-approximation for Hypergraph-SNDP; via the 2-approximation for Elem-SNDP we obtain a d^+approximation for Hypergraph-SNDP. One can view this as reducing a node-weighted problem to an edge-weighted problem by transferring the cost on the nodes to all the edges incident to the node. Since a non-terminal can only be useful if it has at least two edges incident to it, in this particular case, we can put a weight of half the node on the edges incident to the node. A natural question here is whether one can directly get a d^+ approximation for Hypergraph-SNDP without the reduction to Elem-SNDP. We raise the following technical question.

▶ **Problem 8.** *Suppose f is a non-trivial skew-supermodular function on V and $G = (V, \mathcal{E})$ be a hypergraph. Let $\overline{x}$ be a basic feasible solution to the LP for covering f by G. Is there an hyperedge $e \in \mathcal{E}$ such that $\overline{x}_e \geq \frac{1}{d}$ where d is the degree of G?*

The preceding propositions show that Hypergraph-SNDP is essentially equivalent to node-weighted Elem-SNDP where the node-weights are only put on non-terminals. Node-weighted Steiner tree can be reduced to node-weighted Elem-SNDP and it is known that Set Cover reduces in an approximation preserving fashion to node-weighted Steiner tree [11]. Hence, unless $P = NP$, we do not expect a better than $O(\log n)$-approximation for Hypergraph-SNDP where $n = |V|$ is the number of nodes in the graph. Thus, the approximation ratio for Hypergraph-SNDP cannot be a constant independent of d^+. Node-weighted Elem-SNDP admits an $O(r_{\max} \log |V|)$ approximation; see [14, 3, 4, 8]. For planar graphs, and more generally graphs from a proper minor-closed family, an improved bound of $O(r_{\max})$ is claimed in [3]. The $O(r_{\max} \log |V|)$ bound can be better than the bound of d^+ in some instances. Here we raise a question based on the fact that planar graphs have constant average degree which is used in the analysis for node-weighted network design.

▶ **Problem 9.** *Is there an $O(1)$-approximation for node-weighted EC-SNDP and Elem-SNDP in planar graphs, in particular when $r_{\max}$ is a fixed constant?*

Finally, we hope the counting argument and the connections between Hypergraph-SNDP, EC-SNDP and Elem-SNDP will be useful for related problems including the problems involving degree constraints in network design.

References

1 Nikhil Bansal, Rohit Khandekar, and Viswanath Nagarajan. Additive guarantees for degree-bounded directed network design. *SIAM Journal on Computing*, 39(4):1413–1431, 2009.

2 Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 167–176. ACM, 2008.

3 Chandra Chekuri, Alina Ene, and Ali Vakilian. Node-weighted network design in planar and minor-closed families of graphs. In *Automata, Languages, and Programming*, pages 206–217. Springer, 2012.

4 Chandra Chekuri, Alina Ene, and Ali Vakilian. Prize-collecting survivable network design in node-weighted graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 98–109. Springer, 2012.

5 J. Cheriyan, S. Vempala, and A. Vetta. Network design via iterative rounding of setpair relaxations. *Combinatorica*, 26(3):255–275, 2006.

6 Julia Chuzhoy and Sanjeev Khanna. An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. *Theory of Computing*, 8:401–413, 2012. Preliminary version in Proc. of IEEE FOCS, 2009.

7 L. Fleischer, K. Jain, and D.P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006.

8 Takuro Fukunaga. Spider covers for prize-collecting network activation problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 9–24. SIAM, 2015. URL: `http://dl.acm.org/citation.cfm?id=2722129.2722131`.

9 M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, E. Tardos, and D.P. Williamson. Improved approximation algorithms for network design problems. In *Proc. of ACM-SIAM SODA*, pages 223–232, 1994.

10 K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. Preliminary version in FOCS 1998.

11 P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *J. Algorithms*, 19(1):104–115, 1995. Preliminary version in IPCO 1993.

12 Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.

13 Viswanath Nagarajan, R Ravi, and Mohit Singh. Simpler analysis of lp extreme points for traveling salesman and survivable network design problems. *Operations Research Letters*, 38(3):156–160, 2010.

14 Z. Nutov. Approximating minimum cost connectivity problems via uncrossable bifamilies and spider-cover decompositions. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 417–426. IEEE, 2009.

15 David P Williamson and David B Shmoys. *The design of approximation algorithms.* Cambridge university press, 2011.

16 Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. A note on approximating the survivable network design problem in hypergraphs. *IEICE TRANSACTIONS on Information and Systems*, 85(2):322–326, 2002.

17 Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. A primal–dual approximation algorithm for the survivable network design problem in hypergraphs. *Discrete applied mathematics*, 126(2):275–289, 2003. Preliminary version appeared in *Proc. of STACS*, 2001.

Congestion Minimization for Multipath Routing via Multiroute Flows*

Chandra Chekuri[1] and Mark Idleman[2]

1 **Dept. of Computer Science, University of Illinois, Urbana, 61801, USA**
chekuri@illinois.edu

2 **Dept. of Computer Science, University of Illinois, Urbana, 61801, USA**
midleman2@illinois.edu

Abstract

Congestion minimization is a well-known routing problem for which there is an $O(\log n/\log\log n)$-approximation via randomized rounding [17]. Srinivasan [18] formally introduced the low-congestion *multi-path* routing problem as a generalization of the (single-path) congestion minimization problem. The goal is to route multiple disjoint paths for each pair, for the sake of fault tolerance. Srinivasan developed a *dependent* randomized scheme for a special case of the multi-path problem when the input consists of a given set of disjoint paths for each pair and the goal is to select a given subset of them. Subsequently Doerr [7] gave a different dependent rounding scheme and derandomization. In [8] the authors considered the problem where the paths have to be chosen, and applied the dependent rounding technique and evaluated it experimentally. However, their algorithm does not maintain the required disjointness property without which the problem easily reduces to the standard congestion minimization problem.

In this note we show a simple algorithm that solves the problem correctly without the need for dependent rounding — standard independent rounding suffices. This is made possible via the notion of multiroute flows originally suggested by Kishimoto et al. [13]. One advantage of the simpler rounding is an improved bound on the congestion when the path lengths are short.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems: Computations on Discrete Structures

Keywords and phrases multipath routing, congestion minimization, multiroute flows

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.3

1 Introduction

Congestion minimization is a routing problem which originally arose in the context of wire routing problem in VLSI design. It is also a relaxation of the classical disjoint paths problem. Here we restrict attention to directed graphs. The input to these problem consists of a directed graph $G = (V, E)$ and a collection of source-sink pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_h, t_h)$. The edge-dijsoint paths problem (EDP for short) is the following: given the graph and the pairs, can the given pairs be connected via edge-disjoint paths? More formally, are there edge-disjoint paths $P_1 \ldots, P_h$ such that for $1 \le i \le k$, P_i is an s_i-t_i path? This is a classical NP-Complete problem. An optimization related to EDP is the *congestion minimization* problem. The goal is to find a collection of paths for the given pairs such that the *congestion* of the paths is minimized. The congestion of an edge e with respect to a collection of paths is

* Supported in part by NSF grants CCF-1319376 and CCF-1526799. Part of this work appeared in the author's recent MS thesis [11].

the number of paths in the collection that contain e, and the congestion of a given collection of paths is simply the maximum congestion over all the edges. Raghavan and Thompson, in their influential work [17], introduced the randomized rounding technique and obtained an $O(\log n/\log\log n)$ approximation via a natural multicommodity flow relaxation. Here n is the number of nodes in the graph. Surprisingly this approximation ratio was recently shown to be the right threshold of approximability [6] (modulo appropriate complexity theoretic assumption). There are generalizations of the congestion minimization problem where the pairs have demands and the edges have capacities. We restrict attention to the basic version with unit demands and unit capacities. The results for this basic version generalize easily.

Multipath routing for fault tolerance: The focus of this paper is *multipath* routing. This is motivated by fault tolerance considerations in high-capacity networks such as optical networks. In such networks each pair (s_i, t_i) needs to be connected via k_i disjoint paths; in typical applications $k_i = 2$. The idea is to protect the connection between s_i and t_i in case of edge or node failures. We will restrict our attention to edge failures since node failures can be addressed via appropriate reductions in directed graphs. In the network literature the case of $k_i = 2$ is typically referred to a $1 + 1$ or $1 : 1$ protection. See [9] for some relevant background and additional references.

Srinivasan [18] considered approximation algorithms for the multipath congestion minimization problem that he formalized as follows. The input is a directed graph $G = (V, E)$ and h source-sink pairs as before. In addition, for each (s_i, t_i), we have an integer requirement $k_i \geq 1$. The goal is to choose for each pair i a set of k_i edge-disjoint s_i-t_i paths $Q_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,k_i}\}$ so as to minimize the congestion induced by the collection of paths $\mathcal{Q} = \cup_i Q_i$. Srinivasan developed an $O(\log n/\log\log n)$-approximation for a variant of this problem. He assumes that the input includes for each i a collection of disjoint paths $\mathcal{P}_i$. The goal now is to choose for each i a sub-collection of exactly k_i paths from $\mathcal{P}_i$ so as to minimize the congestion of the chosen paths. For this purpose Srinivasan developed and used his influential *dependent rounding* technique for cardinality constraints [18]. We refer the reader to [9, 7, 5] for several subsequent developments on dependent rounding including derandomization, and the ability to handle more general constraints than cardinality constraints. Doerr et al. [8] consider the multipath congestion minimization problem where the path collection $\mathcal{P}_i$ is not explicitly given and the goal of the algorithm is to find k_i disjoint paths for each pair i so as to minimize the congestion of the chosen path collection. They write a natural multicommodity flow LP relaxation and use dependent rounding techniques to derive an $O(\log n/\log\log n)$ congestion bound. However, their algorithm does not maintain the crucial property that the chosen path collection for each pair i is edge disjoint! If there is no requirement of edge-disjointness, the problem is easily reduced to the standard congestion minimization problem: simply create k_i copies of each pair (s_i, t_i) with only one path for each copy required.

Our contribution: This note is motivated by two considerations. The first is to address the deficiency of the algorithm from [8] that we pointed out. We also note that the assumption that the input consists of a large number of disjoint paths for each pair, as assumed by Srinivasan, may not be the right model. Indeed, in practice a number of candidate paths are generated for each pair but they need not all be disjoint. Typically, the edge connectivity in high-speed networks is not very large. Can we solve the multipath congestion minimization problem when the paths are not explicitly given? Our second motivation is regarding the use of dependent rounding. Dependent rounding has been an elegant and powerful methodology with several new applications. Nevertheless, it is useful to examine whether specific applications really need it since the dependent rounding adds to the complexity of the algorithm from a conceptual and implementation point of view.

Here we show that there is a simple solution to the multipath congestion minimization problem via the notion of multiroute flows. We show that standard randomized rounding can be used to obtain an $O(\log n/\log\log n)$ approximation. In other words there is no need to use dependent rounding for the multipath congestion minimization problem. There is also a concrete advantage to the simpler rounding. It allows us to improve the approximation to $O(\log d/\log\log d)$ when the paths have only d edges; this improvement requires the use of the Lovasz local lemma (LLL) and its constructive version [15, 10]. As far as we are aware this improvement is not easy to obtain via the dependent rounding approach.

One of our goals is to highlight multiroute flows. These flows were originally introduced by Kishimoto et al. [13] and some of their properties have been clarified by Aggarwal and Orlin [1]. Multiroute flows are a useful concept when considering fault-tolerance in networks, and it appears that they are less well known in the theoretical computer science literature although there have been several previous applications [3, 14, 4].

Finally, the simpler algorithm presented here inspired the following algorithmic question that we briefly describe although it is not the main focus of this paper. Given an s-t flow in a directed graph $G = (V, E)$ it is well-known that it can be decomposed into paths and cycles in $O(nm)$ time where $n = |V|$ and $m = |E|$. The corresponding question for multiroute flows has not been explored systematically. Given an s-t k-route flow (see Section 2 for definitions and background on multiroute flows) how fast can we decompose it into a collection of elementary k-flows? The existence of a polynomial time algorithm follows from the properties of these flows [1]. The MS thesis of the second author [11] describes faster exact and approximation algorithms for the multiroute flow decomposition problem.

2 Background on multiroute flows

We will assume that graphs are directed for the rest of the paper. Unless otherwise noted, we will use n and m to represent the number of nodes and edges of a graph in question, respectively.

Network flow and flow decomposition: Given a directed graph $G = (V, E)$ with non-negative capacities c_e on each edge $e \in E$, a *network flow* is defined as a function $f : E \to \mathbb{R}^+$. We will often express flows as vectors to help differentiate them from scalar values; for a given flow $\overline{f}$ written in this way, f_e is the flow value on an edge e. A *feasible flow* of k units from a source node s to a target node t is a flow $\overline{f}$ such that

1. The flow value f_e on each edge $e \in E$ satisfies $0 \le f_e \le c_e$,
2. k units of flow leave vertex s and enter vertex t, i.e.,

$$\sum_{e=(s,u)} f_e - \sum_{e=(u,s)} f_e = \sum_{e=(u,t)} f_e - \sum_{e=(t,u)} f_e = k, \text{ and}$$

3. $\overline{f}$ satisfies the *flow conservation property*: for each vertex $v \in V \setminus \{s, t\}$,

$$\sum_{e=(u,v)} f_e = \sum_{e=(v,u)} f_e.$$

The above definition characterizes network flows based on the flow value assigned to each edge in the graph; we will refer to such formulations as *edge-based flows*. It is often beneficial to also consider an equivalent formulation of network flows, the *path-based flow*.

Let $\mathcal{P}_{st}$ be the set of all paths from s to t. In the path based formulation, an s-t flow in a graph G is defined as a function $f : \mathcal{P}_{st} \to \mathbb{R}_+$. Although this definition works with an implicit and exponential sized set $\mathcal{P}_{st}$, there are several advantages to this view. The path-based and edge-based formulations of s-t flows are "equivalent" in a certain sense. One can see that a path-based flow induces an edge-based flow of the same value, and conversely, *flow-decomposition* allows one convert an edge-based acyclic flow into a path-based flow of the same value (note that the decomposition is not unique). Such a decomposition can be computed in $O(nm)$ time.

Multiroute aka k-route flows: Let k be a non-negative integer. Given a directed graph $G = (V, E)$ and two distinct nodes $s, t \in V$, an *elementary k-flow* from s to t is defined as an s-t flow of k total units, consisting of 1 unit sent along each of k edge-disjoint s-t paths. Equivalently we can view an elementary k-flow as a tuple $(p_1, p_2, \ldots, p_k)$ where each $p_i \in \mathcal{P}_{st}$ and the paths $p_1, p_2, \ldots, p_k$ are mutually edge-disjoint. Let $\mathcal{P}_{st}^{(k)}$ be the set of all elementary k-flows from s to t in G. We will some times use the notation $\bar{p}$ to denote an elementary k-flow in $\mathcal{P}_{st}^{(k)}$. A k-route s-t flow is defined as $f : \mathcal{P}_{st}^{(k)} \to \mathbb{R}_+$, in other words, as a non-negative sum of elementary k-flows. The value of a k-route flow f is simply $\sum_{\bar{p} \in \mathcal{P}_{st}^{(k)}} f(\bar{p})$. A k-route flow f is feasible if the total flow on any edge is at most its capacity $c(e)$; that is, $\sum_{\bar{p} \ni e} f(\bar{p}) \leq c(e)$ for each e. Note that the case of $k = 1$ is simply the standard definition of flow via the path formulation.

Two natural questions arise. Can a maximum k-route flow be computed efficiently? Second, what is the relationship between standard flows and k-route flows. The first question is easy to answer. One can write a natural LP relaxation for the maximum k-route flow based on the definition, however, the number of variables is $|\mathcal{P}_{st}^{(k)}|$ and hence exponential in the graph size. However, the number of non-trivial constraints is only m and one see that the separation oracle for the dual LP is poly-time solvable via min-cost flow. However, there are much faster algorithms via a crucial property that connects k-route flows to standard flows (1-route flows). We have defined k-route flows via a path formulation. We say that an edge-based flow $f : E \to \mathbb{R}_+$ is a k-route flow if it can be decomposed into a k-route flow. More formally f is a k-route flow if there is a $g : \mathcal{P}_{st}^{(k)} \to \mathbb{R}_+$ such that $f(e) = \sum_{\bar{p} \ni e} g(\bar{p})$.

The following theorem, first proved by Kishimoto [12] and subsequently simplified by Aggarwal and Orlin [1], gives a simple necessary and sufficient condition for a flow to be a k-route flow. See Figure 1. This condition is related to the integer decomposition property of polytopes defined by totally unimodular matrices [2].

▶ **Theorem 1** ([12, 1]). *An acyclic edge-based s-t flow $f : E \to \mathbb{R}_+$ can be decomposed into an s-t k-route flow if and only if $f(e) \leq v/k$ for each $e \in E$, where v is the value of f.*

The proof of the preceding theorem gives a polynomial time algorithm for the decomposition. Recently we have improved the running time for the decomposition; details can be found in [11].

Aggarwal and Orlin describe an algorithm that finds a maximum s-t k-route flow via $\min\{k, \log(kU)\}$ standard maximum flow computations. Here U is the maximum capacity of any edge in the graph. Note that the algorithm returns an edge-based multiroute flow.

3 Multipath routing via multiroute flows

In this section we consider the multipath minimum congestion routing problem via multiroute flows. We are given a directed network $G = (V, E)$ along with h *commodities*, each of which consists of a pair of vertices (s_i, t_i) in G. For each commodity (s_i, t_i), the node s_i is referred

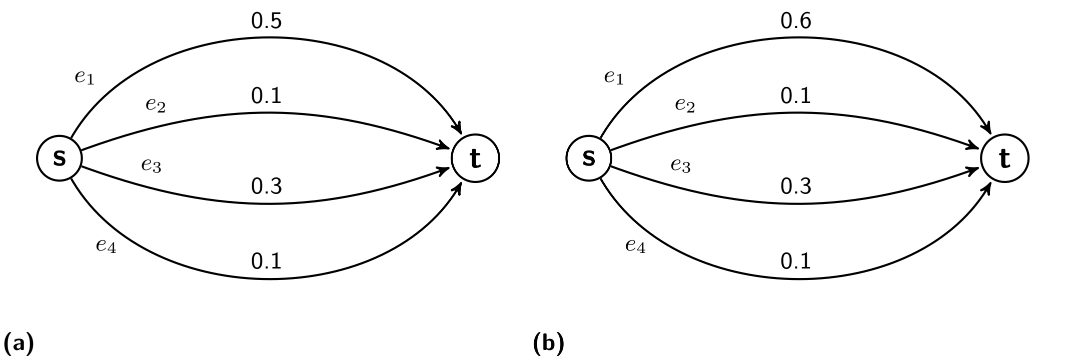

Figure 1 The flow in (a) is decomposable into a 2-route flow by sending 0.3 units of flow along e_1 and e_3, 0.1 units of flow along e_1 and e_2, and 0.1 units of flow along e_1 and e_4. The flow in (b) is not decomposable into a 2-route flow; the 0.6 units of flow on edge e_1 presents a bottleneck to obtaining a valid decomposition.

to as the *source* node, and the node t_i is referred to as the *sink* node. For each commodity we are also given an integer k_i which is the number of edge-disjoint paths needed for pair i. For notational simplicity we will assume that $k_i = k$ for all i. It is easy to generalize the entire approach when k_i are different. The goal is to find for each i an elementary k-flow $\bar{p}_i \in \mathcal{P}^{(k)}_{st}$ such that the congestion induced by the path collection in $\cup_i \bar{p}_i$ is minimized. The whole approach can also be generalized to the setting where pairs have demands and edges have capacities and the goal is to minimize the relative congestion. We avoid this general version for notational simplicity.

As we mentioned, Srinivasan [18, 9] considered the version of the problem where the elementary k-flow for each i has to be chosen from a given set of disjoint path collection $\mathcal{P}_i \subseteq \mathcal{P}_{st}$. Here we consider the version where the algorithm is not given such a path collection. (We later show that the given paths case can be treated as a special case.) We write two different relaxations, one based on the path based definition of multiroute flows and the other based on the edge based definition (via Theorem 1).

Figure 2 describes the path-based formulation. For ease of notation we will assume that the pairs are distinct (otherwise we can add dummy terminals to achieve this) and hence $\mathcal{P}^{(k)}_{s_i t_i}$ for the different i are distinct. For each $\bar{p} \in \cup_i \mathcal{P}^{(k)}_{s_i t_i}$ we have a flow variable $x(\bar{p})$ indicating the amount of flow that is sent on the elementary k-flow $\bar{p}$. The natural constraints are that the total k-route flow for (s_i, t_i) is 1 for each i. The goal is to minimize the maximum flow on any edge e which is the variable C. This is the same as minimizing the maximum congestion since we are working with the case when all capacities are 1.

Figure 3 describes the edge-based formulation. Here we have variables $x_{i,e}$ to indicate the flow for pair i on edge e. In addition to flow conservation constraints we seek a total flow of k units from s_i to t_i. The goal is again to minimize the maximum flow on any edge e which is the variable C. Note that we include the constraint that $x_{i,e} \leq 1$ for each e and each i. This is crucial. If we did not include this constraint we would not be able to guarantee that the flow for pair i defined by the variables $x_{i,e}$ is a k-route flow. We observe that [8] write this same relaxation.

The proof of the following lemma easily follows from Theorem 1. We note that the constraint $x_{i,e} \leq 1$ is necessary.

▶ **Lemma 2.** *The two LP relaxation are equivalent in that any feasible solution to one can be used to define a corresponding feasible solution of the same or better value to the other.*

$$\begin{aligned}
&\min C \\
&\sum_{\bar{p} \in \mathcal{P}^{(k)}_{s_i t_i}} x(\bar{p}) = 1 && i = 1, \dots, h \\
&\sum_{i=1}^{h} \sum_{\bar{p} \in \mathcal{P}^{(k)}_{s_i t_i}, e \in \bar{p}} x(\bar{p}) \le C && e \in E \\
&x(\bar{p}) \ge 0 && \bar{p} \in \cup_i \mathcal{P}^{(k)}_{s_i t_i}
\end{aligned}$$

Figure 2 LP relaxation for multipath congestion minimization via path-based flows.

$$\begin{aligned}
&\min C \\
&\sum_{e=(w,v)\in E} x_{i,e} - \sum_{e=(v,w)\in E} x_{i,e} = 0 && v \in V \setminus \{s_i, t_i\}, i = 1, \dots, h \\
&\sum_{e=(s_i,v)\in E} x_{i,e} - \sum_{e=(v,s_i)\in E} x_{i,e} = k && i = 1, \dots, h \\
&\sum_{i=1}^{h} x_{i,e} \le C && e \in E \\
&x_{i,e} \in [0,1] && e \in E, i = 1, \dots, h
\end{aligned}$$

Figure 3 LP relaxation for multipath congestion minimization via edge-based flows.

Proof. We sketch the more interesting direction. Consider a feasible solution the edge-based LP. Consider any commodity i and the s_i-t_i flow of k units induced by the variables $x_{i,e}$. Since $x_{i,e} \le 1$ for each e this flow satisfies the conditions of Theorem 1 and hence can be decomposed into a path-based k-route flow of value 1. We do this decomposition for each i to generate a path-based flow solution. It is easy to see that it is feasible for the path-based LP. ◀

We observe that the path-based LP can be solved in polynomial time. There are two ways to see this. One way is to note that the separation oracle for the dual of the path-based LP is min-cost flow. The other way is via the equivalence shown in Lemma 2 since the edge-based LP is a polynomial sized formulation; one can decompose the edge-based flow for each pair i into a path based flow via Theorem 1.

In [8] the following rounding strategy is used. They first solve the edge-based flow LP. Let C^* be the congestion in the fractional solution. We will assume without loss of generality that $C^* \ge 1$ since we know that 1 is a lower bound on the optimum integral solution. For each commodity i the $x_{i,e}$ variables define a flow of value k. They do a standard flow decomposition of this flow to obtain a collection of paths $\mathcal{Q}_i = \{p_{i,1}, \dots, p_{i,\ell_i}\} \subset \mathcal{P}_{s_i t_i}$ and associated fractions $\alpha_{i,1}, \dots, \alpha_{i,\ell_i}$ such that $\sum_j \alpha_{i,j} = k$ and $0 \le \alpha_{i,j} \le 1$. Then they apply the dependent rounding scheme of Srinivasan or the variant developed in [7] to select exactly k paths from $\mathcal{Q}_i$. They do this process independently for each commodity i. They exploit the negative correlation properties of the dependent rounding which implies that the process behaves as if the paths are chosen independently and hence one can apply Chernoff-bound style analysis and the union bound. This allows one to show that the congestion obtained by the rounding is, with high probability, $O(\log n/\log\log n \cdot C^*)$. One can, via well-known Chernoff-inequalities, also show related bounds that provide improved bounds when C^* is large.

The main issue with the above rounding is the fact that the path collection $\mathcal{Q}_i$ obtained via standard flow-decomposition is not guaranteed to give a collection of disjoint paths. Thus, the k paths chosen for (s_i, t_i) may not in fact be edge disjoint.

Randomized rounding via k-route flows: It is convenient to describe our rounding algorithm via the path-based LP formulation. We solve the LP to find a fractional solution with congestion C^*. Note that the LP solution gives us for each i a k-route flow of value 1. More formally for each i we have a collection $\mathcal{Q}_i = \{\bar{p}_{i,1}, \ldots, \bar{p}_{i,\ell_i}\}$ where $\bar{p}_{i,j} \in \mathcal{P}^{(k)}_{s_i t_i}$ for each j, and also associated non-negative numbers $\alpha_{i,1}, \ldots, \alpha_{i,\ell_i}$ such that $\sum_j \alpha_{i,j} = 1$. Note that we have a convex combination over elementary k-flows for each i. Now we can perform a simple randomized rounding similar to what Raghavan and Thompson did for the standard congestion minimization problem. For each i, we independently pick a single elementary k-flow $\bar{p}_{i,j}$ from $\mathcal{Q}_i$ where the probability of picking it is exactly $\alpha_{i,j}$. Since we are picking an elementary k-flow for each i, we are guaranteed that the k paths for each i are edge disjoint. We can use the same standard argument as in [17] to argue that the congestion induced by this randomized rounding is $O(\log n/\log n \cdot C^*)$. Note that we do not need to use any dependent rounding techniques since all the work has been done for us via the multiroute flow based LP relaxation.

For the sake of completeness we prove the desired bound on the congestion. We first state the standard Chernoff bound that we need (see [16]).

▶ **Theorem 3.** *Let $X_1, \ldots, X_n$ be n independent random variables (not necessarily distributed identically), with each variable X_i taking a value of 0 or v_i for some value $0 < v_i \leq 1$. Then for $X = \sum_{i=1}^n X_i$, $E[X] \leq \mu$, and $\delta > 0$,*

$$\Pr[X \geq (1+\delta)\mu] < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu.$$

▶ **Lemma 4.** *With probability $1 - 1/poly(n)$, the congestion resulting from the randomized rounding algorithm is $O(\log n/\log\log n \cdot C^*)$.*

Proof. For each edge $e \in E$, define $X_{i,e}$ to be a binary random variable where $X_{i,e} = 1$ if e lies on one of the s_i-t_i paths making up the elementary k-flow chosen by the above randomized rounding scheme, and 0 otherwise. Note that $E[X_{i,e}] = x_{i,e}$ is the total flow on e for commodity i. Let $Y_e = \sum_{i \in [h]} X_{i,e}$ be the random variable which is the total number of paths using edge e in the chosen solution. Note that

$$E[Y_e] = \sum_{i \in [h]} E[X_{i,e}] = \sum_{i \in [h]} \sum_{j \in [\ell_i] : e \in \bar{p}_{i,j}} x(\bar{p}_{i,j}) = \sum_{i \in [h]} x_{i,e} \leq C^*.$$

In the above we use the fact that any edge e belongs only to one of the paths of an elementary k-flow since the paths making up the elementary k-flow are by definition edge disjoint. For any edge $e \in E$, the variables $X_{i,e}$, $i \in [h]$ are independent via the rounding procedure; therefore, the bound in Theorem 3 applies. Choose δ such that $(1+\delta) = \frac{c \ln n}{\ln \ln n}$ for some constant c that will be determined later. Assume $n > e$ so that $\ln\ln n - \ln\ln\ln n > 0.5 \ln\ln n$.

By letting $\mu = C^* \geq 1$ in Theorem 3, we then have

$$\begin{aligned}
\Pr[Y_e \geq (1+\delta)C^*] &< \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{C^*} \\
&\leq \frac{e^\delta}{(1+\delta)^{(1+\delta)}} \qquad (*) \\
&\leq \frac{e^{(1+\delta)}}{(1+\delta)^{(1+\delta)}} \\
&= \left(\frac{c\ln n}{e\ln\ln n}\right)^{(-c\ln n/\ln\ln n)} \\
&= \exp((\ln c/e + \ln\ln n - \ln\ln\ln n)(-c\ln n/\ln\ln n)) \\
&\leq \exp(0.5\ln\ln n(-c\ln n/\ln\ln n)) \\
&\leq \frac{1}{n^{c/2}}
\end{aligned}$$

There are at most n^2 edges, so by the union bound, we have

$$\begin{aligned}
\Pr\left[\max_{e\in E} Y_e \geq (1+\delta)C^*\right] &\leq \sum_{e\in E} \Pr[Y_e \geq (1+\delta)C^*] \\
&\leq n^2 \cdot \frac{1}{n^{c/2}} = n^{2-c/2}.
\end{aligned}$$

Choosing $c = 8$ makes the claim fail to hold with probability at most $\frac{1}{n^2}$, and ensures that the inequality marked with (*) above is true (this choice of c ensures that the value of the expression within parenthesis is less than 1). This probability can be made arbitrarily small by increasing c. Because

$$(1+\delta)C^* = (c\ln n/\ln\ln n)C^* = O(\log n/\log\log n)\cdot C^*,$$

this completes the proof. ◀

Using variants of the Chernoff bounds we can prove the following two lemmas as well. These bounds show improved relative bounds on the congestion when C^* is large. One can also show similar analysis if the capacities are large compared to the demands. The analysis is standard and we omit details in this version.

▶ **Lemma 5.** *If $C^* \geq 1$, then for any δ with $0 \leq \delta \leq 1$, there exists some constant $c > 0$ such that the congestion resulting from the randomized rounding algorithm is no more than $(1+\delta)C^* + c\log n/\delta^2$ with probability $1 - 1/n^{\Omega(c)}$.*

▶ **Lemma 6.** *There is a constant $c > 1$ such that if $C^* \geq c\ln n$, then with high probability, the congestion of the rounding algorithm is at most $C^* + \sqrt{C^*(c\ln n)}$.*

3.1 Short paths and improved congestion via local lemma

In this section we point to another advantage of the simple rounding that we described in the preceding section. Consider the basic congestion minimization problem when $k = 1$. It is known that if all the flow paths in the fractional solution are "short", then the congestion bound improves. More formally, if all the paths in the decomposition have at most d edges then one can obtain an integral solution with congestion $O(\log d/\log\log d \cdot C^*)$ [19]. This can be substantially better than the bound of $O(\log n/\log\log n \cdot C^*)$ when $d \ll n$. One can

$$\begin{aligned}
&\min C \\
&\sum_{e=(w,v)\in E} x_{i,e} - \sum_{e=(v,w)\in E} x_{i,e} = 0 && v \in V \setminus \{s_i,t_i\}, i = 1,\dots,h \\
&\sum_{e=(s_i,v)\in E} x_{i,e} - \sum_{e=(v,s_i)\in E} x_{i,e} = k && i = 1,\dots,h \\
&\sum_{i=1}^{h} x_{i,e} \le C && e \in E \\
&\sum_{e\in E} x_{i,e} \le d && i = 1,\dots,h \\
&x_{i,e} \in [0,1] && e \in E, i = 1,\dots,h
\end{aligned}$$

Figure 4 LP relaxation for multipath congestion minimization with additional constraint to limit the number of edges used in the flow to at most d.

also ensure that the flow paths are short by solving a path-based LP relaxation with the restriction that the flow is only on paths with at most d edges. The rounding that achieves the improved bound relies on the Lovász local lemma. LLL based analysis does not yield a polynomial-time. Srinivasan [19] obtained a polynomial-time algorithm by derandomizing the LLL based algorithm. More recently, building on the constructive version of the LLL due to Moser and Tardos [15], Haupeler, Saha and Srinivasan [10] obtained a much simpler randomized polynomial time algorithm that achieves a congestion of $O(\log d/\log\log d \cdot C^*)$. In fact they consider a general class of min-max integer programs, and one can cast the single path routing problem as a special case after the flow-decomposition. In the general setting of min-max integer programs the structure of the routing problem is not relevant for the bound we seek. The only parameter that matters is the maximum number of constraints that any variable participates in, which corresponds to the length of the flow paths.

Now suppose we consider the multipath routing problem. The multiroute flow based path LP formulation can be easily seen to be a special case of min-max integer programs considered in [10]. We can thus obtain an improved congestion bound of $O(\log d/\log\log d \cdot C^*)$ where d is the maximum number of edges in any elementary k-flow. The following natural question then arises. Find a fractional multipath routing for the given instance with the additional constraint that for each pair (s_i, t_i) the elementary flow chosen has at most d edges. For single path setting this LP can be solved in polynomial time since the separation oracle for the dual LP can be seen to be the constrained shortest path problem which is polynomial-time solvable. Unfortunately the corresponding separation oracle for the multipath case is NP-Hard even when $k = 2$. Nevertheless, we can apply a simple trick to obtain a bi-criteria approximation. We can ensure that each elementary k-flow has at most $2d$ edges as follows.

We consider the following relaxation which adds additional constraints to the edge-based relaxation from Figure 3. The additional constraint says that for each commodity i, the total number of edges used is at most d in a fractional sense.

We now describe the modification to the rounding algorithm. As before we consider each commodity i and consider the k-route flow given by the variables $x_{i,e}$. We decompose this into a path based flow to obtain a collection $\mathcal{Q}_i = \{\bar{p}_{i,1}, \dots, \bar{p}_{i,\ell_i}\}$ of elementary k-flows between s_i and t_i. Abusing notation, we let amount of flow on $\bar{p}_{i,j}$ as $x(\bar{p}_{i,j})$. Note that $\sum_j x(\bar{p}_{i,j}) = 1$. Let d_j be the number of edges in $\bar{p}_{i,j}$. From the constraint in the LP on

$$\begin{aligned} &\min C \\ &\sum_{p \in \mathcal{P}_i} y(p) = k \qquad i = 1, \dots, h \\ \sum_{i=1}^{h} &\sum_{p \in \mathcal{P}_i, e \in p} y(p) \le C \qquad e \in E \\ &y(p) \in [0,1] \qquad p \in \cup_i \mathcal{P}_i \end{aligned}$$

Figure 5 LP relaxation for multipath congestion minimization when paths for each pair are specified.

$\sum_e x_{i,e}$ we have that $\sum_j d_j x(\bar{p}_{i,j}) \le d$. Let $\mathcal{Q}'_i \subseteq \mathcal{Q}_i$ be the subset of elementary k-flows such that each of them contains at most $2d$ edges. It is easy to see, via Markov's inequality, that $\sum_{j \in \mathcal{Q}'_i} x(\bar{p}_{i,j}) \ge 1/2$. By scaling up the fractional values of the elementary k-flows in $\mathcal{Q}'_i$ by 2 we obtain a feasible solution to the path-based LP using only elementary k-flows which have at most $2d$ edges. We have thus found a feasible fractional solution to the path LP formulation with the following guarantees: (i) the congestion of the solution is at most $2C^*$ where C^* is the congestion of the original relaxation (ii) the support of the solution consists of elementary k-flows for each commodity i which have at most $2d$ edges. We can now apply the algorithm from [10] to round this solution to obtain a randomized algorithm with congestion $O(\log d / \log\log \cdot C^*)$. Note that there is nothing special about the factor of 2. We can obtain a trade off. For any $\epsilon > 0$ we can ensure that the elementary k-flows have at most $(1+\epsilon)d$ edges while guaranteeing that the congestion is $O(\frac{1}{\epsilon} \cdot \log d / \log\log d \cdot C^*)$.

3.2 Choosing paths from a given collection

Now we consider the setting that Srinivasan considered in his original paper where he assumes that the input includes for each i, a collection of disjoint paths $\mathcal{P}_i$. The goal is to select exactly k paths from $\mathcal{P}_i$ for each i. We can handle this problem also via multiroute flow decomposition. First, we consider the natural LP relaxation for this problem from [18]. For simplicity we will again assume that the given h pairs are distinct and hence $\mathcal{P}_\rangle \cap \mathcal{P}_j = \emptyset$ for $i \ne j$. We have a variable $y(p)$ for each $p \in \cup_i \mathcal{P}_i$ to indicate whether p is chosen or not. We require k paths to be chosen from each $\mathcal{P}_i$ and also that $y(p) \in [0,1]$. Subject to these conditions we minimize the congestion. The relaxation is formally specified in Figure 5.

Suppose we are given a feasible solution y to the preceding LP with congestion value C. We claim that we can find a feasible solution x to the LP in Figure 2 with congestion value at most C with the following additional condition: for any i, if $x(\bar{p}) > 0$ for some $\bar{p} \in \mathcal{P}^{(k)}_{s_i t_i}$ then $\bar{p}$ is a tuple of k paths from $\mathcal{P}_i$. If this is true then we can do randomized rounding via x as before and obtain the desired congestion bound while picking for each i exactly k paths from $\mathcal{P}_i$. Moreover, if the paths in $\cup_i \mathcal{P}_i$ are short we can obtain an improved congestion bound via the algorithm described in the preceding subsection.

We now justify the claim. Suppose y is a feasible solution to the LP in Figure 5. Fix a particular i. Without loss of generality $\mathcal{P}_i = \{p_1, p_2, \dots, p_{\ell_i}\}$ for some $\ell_i \ge k$. Consider a graph H_i with two nodes s_i, t_i and ℓ_i parallel edges $e_1, e_2, \dots, e_{\ell_i}$ from s_i and t_i with unit capacities where e_j corresponds to the path p_j. We now create a flow of value k from s_i to t_i in H_i where the flow on edge e_j is equal to $y(p_j)$. Note that $y(p_j) \le 1$ for each p_j. Thus, via

Theorem 1, we can decompose this flow into a s_i-t_i k-route flow in H_i. Suppose this k-route flow is given by $x' : \mathcal{Q}^{(k)}_{s_it_i} \to [0,1]$. Here $\mathcal{Q}^{(k)}_{s_it_i}$ is the set of elementary k-flows in H_i; each such elementary k-flow is a set of k distinct edges from $\{e_1, e_2, \ldots, e_{\ell_i}\}$. For each $\bar{q} \in \mathcal{Q}^{(k)}_{s_it_i}$ there is a unique $\bar{p} \in \mathcal{P}^{(k)}_{s_it_i}$ where the edge e_j maps to the path p_j; we set $x(\bar{p}) = x(\bar{q})$. We set $x(\bar{p}) = 0$ for all $\bar{p} \in \mathcal{P}^{(k)}_{s_it_i}$ which don't correspond to an elementary k-flow in H_i. We do this for each i and the resulting x is the claimed feasible solution to the LP in Figure 2. By construction x satisfies (i) $\sum_{\bar{p} \in \mathcal{P}^{(k)}_{s_it_i}} x(\bar{p}) = 1$ for each i and (ii) if $\bar{p} \in \mathcal{P}^{(k)}_{s_it_i}$ and $x(\bar{p}) > 0$ then $\bar{p}$ is a set of k paths from $\mathcal{P}_i$. It is also easy to check that the congestion induced by x on any edge is the same as the congestion induced by y.

References

1 Charu C. Aggarwal and James B. Orlin. On multiroute maximum flows in networks. *Networks*, 39(1):43–52, 2002. `doi:10.1002/net.10008`.

2 Stephen Baum and Leslie E Trotter Jr. Integer rounding and polyhedral decomposition for totally unimodular systems. *Optimization and Operations Research*, 157:15–23, 1978.

3 Graham Brightwell, Gianpaolo Oriolo, and F Bruce Shepherd. Reserving resilient capacity in a network. *SIAM journal on discrete mathematics*, 14(4):524–539, 2001.

4 Chandra Chekuri, Alina Ene, and Ali Vakilian. Prize-collecting survivable network design in node-weighted graphs. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 98–109, 2012.

5 Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 575–584. IEEE, 2010.

6 Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, and Kunal Talwar. Hardness of routing with congestion in directed graphs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 165–178. ACM, 2007.

7 Benjamin Doerr. Randomly rounding rationals with cardinality constraints and derandomizations. *STACS 2007*, pages 441–452, 2007.

8 Benjamin Doerr, Marvin Künnemann, and Magnus Wahlström. Randomized rounding for routing and covering problems: Experiments and improvements. In *Proceedings of the 9th International Conference on Experimental Algorithms*, SEA'10, pages 190–201, Berlin, Heidelberg, 2010. Springer-Verlag. `doi:10.1007/978-3-642-13193-61_7`.

9 Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360, 2006.

10 Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovász local lemma. *Journal of the ACM (JACM)*, 58(6):28, 2011.

11 Mark Idleman. Approximation algorithms for the minimum congestion routing problem via k-route flows. Master's thesis, University of Illinois, July 2017.

12 Wataru Kishimoto. A method for obtaining the maximum multiroute flows in a network. *Networks*, 27(4):279–291, 1996. `doi:10.1002/(SICI)1097-0037(199607)27:4<279::AID-NET3>3.0.CO;2-D`.

13 Wataru Kishimoto and Masashi Takeuchi. m-route flows in a network. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 77(5):1–18, 1994. `doi:10.1002/ecjc.4430770501`.

14 Andrew McGregor and F. Bruce Shepherd. Island hopping and path colouring with applications to WDM network design. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein,

editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 864–873. SIAM, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283476`.

15 Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

16 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms.* Chapman & Hall/CRC, 2010.

17 Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987. `doi:10.1007/BF02579324`.

18 Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 588–597. IEEE Computer Society, 2001. `doi:10.1109/SFCS.2001.959935`.

19 Aravind Srinivasan. An extension of the lovász local lemma, and its applications to integer programming. *SIAM Journal on Computing*, 36(3):609–634, 2006.

Better and Simpler Error Analysis of the Sinkhorn-Knopp Algorithm for Matrix Scaling*

Deeparnab Chakrabarty[1] and Sanjeev Khanna[2]

1 **Department of Computer Science, Dartmouth College, Hanover NH, USA**
deeparnab@dartmouth.edu
2 **Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA, USA**
sanjeev@cis.upenn.edu

Abstract

Given a non-negative $n \times m$ real matrix A, the *matrix scaling* problem is to determine if it is possible to scale the rows and columns so that each row and each column sums to a specified target value for it. The matrix scaling problem arises in many algorithmic applications, perhaps most notably as a preconditioning step in solving linear system of equations. One of the most natural and by now classical approach to matrix scaling is the Sinkhorn-Knopp algorithm (also known as the RAS method) where one alternately scales either all rows or all columns to meet the target values. In addition to being extremely simple and natural, another appeal of this procedure is that it easily lends itself to parallelization. A central question is to understand the rate of convergence of the Sinkhorn-Knopp algorithm.

Specifically, given a suitable error metric to measure deviations from target values, and an error bound ε, how quickly does the Sinkhorn-Knopp algorithm converge to an error below ε? While there are several non-trivial convergence results known about the Sinkhorn-Knopp algorithm, perhaps somewhat surprisingly, even for natural error metrics such as ℓ_1-error or ℓ_2-error, this is not entirely understood. In this paper, we present an elementary convergence analysis for the Sinkhorn-Knopp algorithm that improves upon the previous best bound. In a nutshell, our approach is to show (i) a simple bound on the number of iterations needed so that the KL-divergence between the current row-sums and the target row-sums drops below a specified threshold δ, and (ii) then show that for a suitable choice of δ, whenever KL-divergence is below δ, then the ℓ_1-error or the ℓ_2-error is below ε. The well-known Pinsker's inequality immediately allows us to translate a bound on the KL divergence to a bound on ℓ_1-error. To bound the ℓ_2-error in terms of the KL-divergence, we establish a new inequality, referred to as (KL vs ℓ_1/ℓ_2). This new inequality is a strengthening of the Pinsker's inequality that we believe is of independent interest. Our analysis of ℓ_2-error significantly improves upon the best previous convergence bound for ℓ_2-error.

The idea of studying Sinkhorn-Knopp convergence via KL-divergence is not new and has indeed been previously explored. Our contribution is an elementary, self-contained presentation of this approach and an interesting new inequality that yields a significantly stronger convergence guarantee for the extensively studied ℓ_2-error.

1998 ACM Subject Classification F.2.1.3 Computations on matrices

Keywords and phrases Matrix Scaling, Entropy Minimization, KL Divergence Inequalities

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.4

* This work was supported in part by the National Science Foundation grants CCF-1552909 and CCF-1617851.

1 Introduction

In the matrix scaling problem one is given an $n \times m$ non-negative matrix A, and positive integer vectors $\mathbf{r} \in \mathbb{Z}^n_{>0}$ and $\mathbf{c} \in \mathbb{Z}^m_{>0}$ with the same ℓ_1 norm $\sum_{i=1}^n \mathbf{r}_i = \sum_{j=1}^m \mathbf{c}_j = h$. The objective is to determine if there exist diagonal matrices $R \in \mathbb{R}^{n\times n}$ and $S \in \mathbb{R}^{m\times m}$ such that the ith row of the matrix RAS sums to $\mathbf{r}_i$ for all $1 \le i \le n$ *and* the jth column of RAS sums to $\mathbf{c}_j$ for all $1 \le j \le m$. Of special importance is the case when $n = m$ and $\mathbf{r} \equiv \mathbf{c} \equiv \mathbf{1}_n$, the n-dimensional all-ones vector – the $(\mathbf{1}, \mathbf{1})$-matrix scaling problem wishes to scale the rows and columns of A to make it doubly stochastic. This problem arises in many different areas ranging from transportation planning [12, 26] to quantum mechanics [32, 1]; we refer the reader to a recent comprehensive survey by Idel [15] for more examples.

One of the most natural algorithms for the matrix scaling problem is the following Sinkhorn-Knopp algorithm [33, 34], which is known by many names including the RAS method [4] and the Iterative Proportional Fitting Procedure [30]. The algorithm starts off by multiplicatively scaling all the columns by the columns-sum times $\mathbf{c}_j$ to get a matrix $A^{(0)}$ with column-sums $\mathbf{c}$. Subsequently, for $t \ge 0$, it obtains the $B^{(t)}$ by scaling each row of $A^{(t)}$ by the respective row-sum times $\mathbf{r}_i$, and obtain $A^{(t+1)}$ by scaling each column of $B^{(t)}$ by the respective column sums time $\mathbf{c}_j$. More precisely,

$$A^{(0)}_{ij} := \frac{A_{ij}}{\sum_{i=1}^n A_{ij}} \cdot \mathbf{c}_j \qquad \forall t \ge 0, \quad B^{(t)}_{ij} := \frac{A^{(t)}_{ij}}{\sum_{j=1}^m A^{(t)}_{ij}} \cdot \mathbf{r}_i, \quad A^{(t+1)}_{ij} := \frac{B^{(t)}_{ij}}{\sum_{i=1}^n B^{(t)}_{ij}} \cdot \mathbf{c}_j$$

The above algorithm is simple and easy to implement and each iteration takes $O(\mathsf{nnz}(A))$, the number of non-zero entries of A. Furthermore, it has been known for almost five decades [33, 34, 13, 35] that if A is $(\mathbf{r}, \mathbf{c})$-scalable then the above algorithm asymptotically[1] converges to a right solution. More precisely, given $\varepsilon > 0$, there is some finite t by which one obtains a matrix which is "ε-close to having row- and column-sums $\mathbf{r}$ and $\mathbf{c}$".

However, the rate of convergence of this simple algorithm is still not fully understood. Since the rate depends on how we measure "ε-closeness", we look at two natural error definitions. For any t, let $\mathbf{r}^{(t)} := A^{(t)}\mathbf{1}_m$ denote the vector of row-sums of $A^{(t)}$. Similarly, we define $\mathbf{c}^{(t)} := {B^{(t)}}^\top \mathbf{1}_n$ to be the vector of the column-sums of $B^{(t)}$. Note that $\sum_{i=1}^n \mathbf{r}^{(t)}_i = \sum_{j=1}^m \mathbf{c}^{(t)}_j = h$ for all t. The error of the matrix A_t (the error of matrix B_t similarly defined) is

$$\ell_1\text{-error}: \quad \mathsf{error}_1(A_t) := ||\mathbf{r}^{(t)} - \mathbf{r}||_1 \qquad\qquad \ell_2\text{-error}: \quad \mathsf{error}_2(A_t) := ||\mathbf{r}^{(t)} - \mathbf{r}||_2$$

In this note, we give simple convergence analysis for both error norms. Our result is the following.

[1] Computationally, this asymptotic viewpoint is unavoidable in the sense that there are simple examples for which the unique matrix scaling matrices need to have irrational entries. For instance, consider the following example from Rothblum and Schneider [29]. The matrix is $\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ with $\mathbf{r} \equiv \mathbf{c} \equiv [1, 1]^\top$. The unique R and S matrices are $\begin{bmatrix} (\sqrt{2}+1)^{-1} & 0 \\ 0 & (\sqrt{2}+2)^{-1} \end{bmatrix}$ and $\begin{bmatrix} \sqrt{2} & 0 \\ 0 & 1 \end{bmatrix}$, respectively, giving $RAS = \begin{bmatrix} 2-\sqrt{2} & \sqrt{2}-1 \\ \sqrt{2}-1 & 2-\sqrt{2} \end{bmatrix}$.

▶ **Theorem 1.** *Given a matrix $A \in \mathbb{R}_{\geq 0}^{n \times m}$ which is $(\mathbf{r}, \mathbf{c})$-scalable, and any $\varepsilon > 0$, the Sinkhorn-Knopp algorithm*

1. *in time $t = O\left(\frac{h^2 \ln(n\rho/\nu)}{\varepsilon^2}\right)$ returns a matrix A_t or B_t with ℓ_1-error $\leq \varepsilon$.*
2. *in time $t = O\left(\rho h \ln(n\rho/\nu) \cdot \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon^2}\right)\right)$ returns a matrix A_t or B_t with ℓ_2-error $\leq \varepsilon$.*

Here $h = \sum_{i=1}^n \mathbf{r}_i = \sum_{j=1}^m \mathbf{c}_j$, $\rho = \max(\max_i \mathbf{r}_i, \max_j \mathbf{c}_j)$, and $\nu = \frac{\min_{i,j:A_{ij}>0} A_{ij}}{\max_{i,j} A_{ij}}$.

For the special case of $n = m$ and $\mathbf{r} \equiv \mathbf{c} \equiv \mathbf{1}_n$, we get the following as a corollary.

▶ **Corollary 2.** *Given a matrix $A \in \mathbb{Z}_{\geq 0}^{n \times n}$ which is $(\mathbf{1}, \mathbf{1})$-scalable, and any $\varepsilon > 0$, the Sinkhorn-Knopp algorithm*

1. *in time $t = O\left(\frac{n^2 \ln n}{\varepsilon^2}\right)$ returns a matrix A_t or B_t with ℓ_1-error $\leq \varepsilon$.*
2. *in time $t = O\left(n \ln n \cdot \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon^2}\right)\right)$ returns a matrix A_t or B_t with ℓ_2-error $\leq \varepsilon$.*

▶ Remark. To our knowledge, the ℓ_1-error hasn't been explicitly studied in the literature, although for small $\varepsilon \in (0, 1)$ the same can be deduced from previous papers on matrix scaling [20, 14, 19, 16]. One of our main motivations to look at ℓ_1-error arose from the connections to perfect matchings in bipartite graphs as observed by Linial, Samorodnitsky and Wigderson [20]. For the ℓ_2 error, which is the better studied notion in the matrix scaling literature, the best analysis is due to Kalantari et al [18, 19]. They give a $\tilde{O}(\rho h^2/\varepsilon^2)$ upper bound on the number of iterations for the general problem, and for the special case when $m = n$ and the square matrix has positive permanent (see [18]), they give a $\tilde{O}(\rho(h^2 - nh + n)/\varepsilon^2)$ upper bound. Thus, for $(\mathbf{1}, \mathbf{1})$-scaling, they get the same result as in Corollary 2. We get a quadratic improvement on h in the general case, and we think our proof is more explicit and simpler.

▶ Remark. Both parts of Theorem 1 and Corollary 2 are interesting in certain regimes of error. When the error ε is "small" (say, ≤ 1) so that $1/\varepsilon^2 \geq 1/\varepsilon$, then statement 2 of Corollary 2 implies statement 1 by Cauchy-Schwarz. However, this breaks down when ε is "large" (say $\varepsilon = \delta n$ for some constant $\delta > 0$). In that case, statement 1 implies that in $O(\ln n/\delta^2)$ iterations, the ℓ_1-error is $\leq \delta n$, but Statement 2 only implies that in $O(\ln n/\delta^2)$ iterations, the ℓ_2 norm is $\leq \delta n$. This "large ℓ_1-error regime" is of particular interest for an application to approximate matchings in bipartite graphs discussed below.

Applications to Parallel Algorithms for Bipartite Perfect Matching. As a corollary, we get the following application, first pointed by Linial et al [20], to the existence of perfect matchings in bipartite graphs. Let A be the adjacency matrix of a bipartite graph $G = (L \cup R, E)$ with $A_{ij} = 1$ iff $(i, j) \in E$. If G has a perfect matching, then clearly there is a doubly stochastic matrix X in the support of A. This suggests the algorithm of running the Sinkhorn-Knopp algorithm to A, and the following claim suggests when to stop. Note that each iteration can be run in $O(1)$ parallel time with m-processors.

▶ **Lemma 3.** *If we find a column (or row) stochastic matrix Y in the support of A such that* $\mathsf{error}_1(Y) \leq \varepsilon$*, then G has a matching of size $\geq n(1 - \varepsilon)$.*

Proof. Suppose Y is column stochastic. Given $S \subseteq L$, consider $\sum_{i \in S, j \in \Gamma S} Y_{ij} = |S| + \sum_{i \in S}\left(\sum_{j=1}^n Y_{ij} - 1\right) \geq |S| - \sum_{i=1}^n \left|\sum_{j=1}^n Y_{ij} - 1\right| \geq |S| - n \cdot \mathsf{error}(Y) \geq |S| - n\varepsilon$. On the other hand, $\sum_{i \in S, j \in \Gamma S} Y_{ij} \leq \sum_{j \in \Gamma S} \sum_{i=1}^n Y_{ij} = |\Gamma S|$. Therefore, for every $S \subseteq L$, $|\Gamma S| \geq |S| - n\varepsilon$. The claim follows by approximate Hall's theorem. ◀

▶ **Corollary 4** (Fast Parallel Approximate Matchings). *Given a bipartite graph G of max-degree Δ and an $\varepsilon \in (0, 1)$, $O(\ln \Delta/\varepsilon^2)$-iterations of Sinkhorn-Knopp algorithm suffice to distinguish* *between the case when G has a perfect matching and the case when the largest matching in G has size at most $n(1-\varepsilon)$.*

Thus the approximate perfect matching problem in bipartite graphs is in NC for ε as small as polylogarithmic in n. This is not a new result and can indeed be obtained from the works on parallel algorithms for packing-covering LPs [21, 36, 3, 23], but the Sinkhorn-Knopp algorithm is arguably simpler.

1.1 Perspective

As mentioned above, the matrix scaling problem and in particular the Sinkhorn-Knopp algorithm has been extensively studied over the past 50 years. We refer the reader to Idel's survey [15] and the references within for a broader perspective; in this subsection we mention the most relevant works.

We have already discussed the previously best known, in their dependence on h, analysis for the Sinkhorn-Knopp algorithm in Remark 1. For the special case of *strictly positive* matrices, better rates are known. Kalantari and Khachiyan [16] showed that for positive matrices and the $(\mathbf{1},\mathbf{1})$-scaling problem, the Sinkhorn-Knopp algorithm obtains ℓ_2 error $\leq \varepsilon$ in $O(\sqrt{n}\ln(1/\nu)/\varepsilon)$-iterations; this result was extended to the general matrix scaling problem by Kalantari et al [19]. In a different track, Franklin and Lorenz [13] show that in fact the dependence on ε can be made logarithmic, and thus the algorithm has "linear convergence", however their analysis[2] has a polynomial dependence of $(1/\nu)$. All these results use the positivity crucially and seem to break down even with one 0 entry.

The Sinkhorn-Knopp algorithm has polynomial dependence on the error parameter and therefore is a "pseudopolynomial" time approximation. We conclude by briefly describing bounds obtained by other algorithms for the matrix scaling problem whose dependence on ε is logarithmic rather than polynomial. Kalantari and Khachiyan [17] describe a method based on the ellipsoid algorithm which runs in time $O(n^4\ln(n/\varepsilon)\ln(1/\nu))$. Nemirovskii and Rothblum [25] describe a method with running time $O(n^4\ln(n/\varepsilon)\ln\ln(1/\nu))$. The first strongly polynomial time approximation scheme (with no dependence on ν) was due to Linial, Samoridnitsky, and Wigderson [20] who gave a $\tilde{O}(n^7\ln(h/\varepsilon))$ time algorithm. Rote and Zachariasen [28] reduced the matrix scaling problem to flow problems to give a $O(n^4\ln(h/\varepsilon))$ time algorithms for the matrix scaling problem. To compare, we should recall that Theorem 1 shows that our algorithm runs in time $O(\mathsf{nnz}(A)h^2/\varepsilon^2)$ time.

Very recently, two independent works obtain vastly improved running times for matrix scaling. Cohen et al [9] give $\tilde{O}(\mathsf{nnz}(A)^{3/2})$ time algorithm, while Allen-Zhu et al [2] give a $\tilde{O}(n^{7/3}+\mathsf{nnz}(A)\cdot(n+n^{1/3}h^{1/2}))$ time algorithm; the tildes in both the above running times hide the logarithmic dependence on ε and ν. Both these algorithms look at the matrix scaling problem as a convex optimization problem and perform second order methods.

2 Entropy Minimization Viewpoint of the Sinkhorn-Knopp Algorithm

There have been many approaches (see Idel [15], Section 3 for a discussion) towards analyzing the Sinkhorn-Knopp algorithm including convex optimization and log-barrier methods [16, 19, 22, 5], non-linear Perron-Frobenius theory [24, 35, 13, 8, 16], topological methods [27, 6], connections to the permanent [20, 18], and the entropy minimization method [7, 10, 11, 14] which is what we use for our analysis.

[2] [13] never make the base of the logarithm explicit, but their proof shows it can be as large as $1-1/\nu^2$.

We briefly describe the entropy minimization viewpoint. Given two non-negative matrices M and N let us define the *Kullback-Leibler* divergence[3] between M and N as follows

$$\mathbf{D}(M,N) := \frac{1}{h}\sum_{1\le i\le n}\sum_{1\le j\le m} M_{ij}\ln\left(\frac{M_{ij}}{N_{ij}}\right) \tag{1}$$

with the convention that the summand is zero if both M_{ij} and N_{ij} are 0, and is ∞ if $M_{ij} > 0$ and $N_{ij} = 0$. Let Φ_r be the set of $n \times m$ matrices whose row-sums are $\mathbf{r}$ and let Φ_c be the set of $n \times m$ matrices whose column sums are $\mathbf{c}$. Given matrix A suppose we wish to find the matrix $A^* = \arg\min_{B\in\Phi_r\cap\Phi_c}\mathbf{D}(B,A)$. One algorithm for this is to use the method of alternate projections with respect to the KL-divergence [7] (also known as I-projections [10]) which alternately finds the matrices in Φ_r and Φ_c closest in the KL-divergence sense to the current matrix at hand, and then sets the minimizer to be the current matrix. It is not too hard to see (see Idel [15], Observation 3.17 for a proof) that the above alternate projection algorithm is precisely the Sinkhorn-Knopp algorithm. Therefore, at least in this sense, the right metric to measure the distance to optimality is not the ℓ_1 or the ℓ_2 error as described in the previous section, but the rather the KL-divergence between the normalized vectors as described below.

Let $\pi^{(t)}_{\mathbf{r}} := \mathbf{r}^{(t)}/h$ be the n-dimensional probability vector whose ith entry is $\mathbf{r}^{(t)}_i/h$; similarly define the m-dimensional vector $\pi^{(t)}_{\mathbf{c}}$. Let $\pi_{\mathbf{r}}$ denote the n-dimensional probability vector with the ith entry being $\mathbf{r}_i/h$; similarly define $\pi_{\mathbf{c}}$. Recall that the KL-divergence between two probability distributions p, q is defined as $\mathbf{D}_{KL}(p||q) := \sum_{i=1}^n p_i \ln(q_i/p_i)$. The following theorem gives the convergence time for the KL-divergence.

▶ **Theorem 5.** *If the matrix $A \in \mathbb{R}^{n\times m}_{\ge 0}$ is $(\mathbf{r},\mathbf{c})$-scalable, then for any $\delta > 0$ there is a $t \le T = \lceil\left(\frac{\ln(1+2n\rho/\nu)}{\delta}\right)\rceil$ with either $\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi^{(t)}_{\mathbf{r}}) \le \delta$ or $\mathbf{D}_{KL}(\pi_{\mathbf{c}}||\pi^{(t)}_{\mathbf{c}}) \le \delta$..*

Proof. Let $Z := RAS$ be a matrix with row-sums $\mathbf{r}$ and column-sums $\mathbf{c}$ for diagonal matrices R, S. Recall A^0 is the matrix obtained by column-scaling A. Note that the minimum non-zero entry of A^0 is $\ge \nu/n$.

▶ **Lemma 6.** *$\mathbf{D}(Z, A^0) \le \ln(1 + 2n\rho/\nu)$ and $\mathbf{D}(Z, A^t) \ge 0$ for all t.*

Proof. By definition,

$$\mathbf{D}(Z, A^{(t)}) = \frac{1}{h}\sum_{j=1}^{m}\sum_{i=1}^{n} Z_{ij}\ln\left(\frac{Z_{ij}}{A^{(t)}_{ij}}\right) = \frac{1}{h}\sum_{j=1}^{m}\mathbf{c}_j\sum_{i=1}^{n}\frac{Z_{ij}}{\mathbf{c}_j}\ln\left(\frac{Z_{ij}}{A^{(t)}_{ij}}\right)$$

For a fixed j, the vectors $\left(\frac{Z_{1j}}{\mathbf{c}_j},\frac{Z_{2j}}{\mathbf{c}_j},\ldots,\frac{Z_{nj}}{\mathbf{c}_j}\right)$ and $\left(\frac{A^{(t)}_{1j}}{\mathbf{c}_j},\frac{A^{(t)}_{2j}}{\mathbf{c}_j},\ldots,\frac{A^{(t)}_{nj}}{\mathbf{c}_j}\right)$ are probability vectors, and therefore the above is a sum of $\mathbf{c}_j$-weighted KL-divergences which is always non-negative. For the upper bound, one can use the fact (Inequality 27, [31]) that for any two distributions p and q, $D(p||q) \le \ln(1 + \frac{||p-q||_2^2}{q_{\min}}) \le \ln(1 + \frac{2}{q_{\min}})$ where $q_{\min}$ is the smallest non-zero entry of q. For our purpose, we note that the minimum non-zero probability of the $A^{(0)}_j$ distribution being $\ge \nu/n\rho$. Therefore, the second summand is at most $\ln(1 + 2n\rho/\nu)$ giving us $D(Z, A^{(0)}) \le \frac{1}{h}\sum_{j=1}^m \mathbf{c}_j \cdot \ln(1 + 2n\rho/\nu) = \ln(1 + 2n\rho/\nu)$. ◀

[3] The KL-divergence is normally stated between two distributions and doesn't have the $1/h$ factor. Also the logarithms are usually base 2.

▶ **Lemma 7.**

$$\mathbf{D}(Z,A^{(t)})-\mathbf{D}(Z,B^{(t)})=\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)}) \quad \textit{and} \quad \mathbf{D}(Z,B^{(t)})-\mathbf{D}(Z,A^{(t+1)})=\mathbf{D}_{KL}(\pi_{\mathbf{c}}||\pi_{\mathbf{c}}^{(t)})$$

Proof. The LHS of the first equality is simply

$$\begin{aligned}\frac{1}{h}\sum_{j=1}^{m}\sum_{i=1}^{n} Z_{ij}\ln\left(\frac{B_{ij}^{(t)}}{A_{ij}^{(t)}}\right) &= \frac{1}{h}\sum_{j=1}^{m}\sum_{i=1}^{n} Z_{ij}\ln\left(\frac{\mathbf{r}_i}{\mathbf{r}_i^{(t)}}\right) \\ &= \frac{1}{h}\sum_{i=1}^{n}\ln\left(\frac{\mathbf{r}_i}{\mathbf{r}_i^{(t)}}\right)\sum_{j=1}^{m} Z_{ij} \\ &= \sum_{i=1}^{n}\left(\frac{\mathbf{r}_i}{h}\right)\cdot\ln\left(\frac{\mathbf{r}_i/h}{\mathbf{r}_i^{(t)}/h}\right)\end{aligned}$$

since $\sum_{j=1}^{m} Z_{ij}=\mathbf{r}_i$. The last summand is precisely $\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)})$. The other equation follows analogously. ◀

The above two lemmas easily imply the theorem. If for all $0\le t\le T$, both $\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)})>\delta$ and $\mathbf{D}_{KL}(\pi_{\mathbf{c}}||\pi_{\mathbf{c}}^{(t)})>\delta$, then substituting in Lemma 7 and summing we get $\mathbf{D}(Z,A^{(0)})-\mathbf{D}(Z,A^{(T+1)})>T\delta>\ln(1+2n\rho/\nu)$ contradicting Lemma 6. ◀

Theorem 1 follows from Theorem 5 using connections between the KL-divergence and the ℓ_1 and ℓ_2 norms. One is the following famous Pinsker's inequality which allows us to easily prove part 1 of Theorem 1. Given any two probability distributions p,q,

$$\mathbf{D}_{KL}(p||q)\ge\frac{1}{2}\cdot||p-q||_1^2 \tag{Pinsker}$$

Proof of Theorem 1, Part 1. Apply (Pinsker) on the vectors $\pi_{\mathbf{r}}$ and $\pi_{\mathbf{r}}^{(t)}$ to get

$$\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)})\ge\frac{1}{2h^2}||\mathbf{r}^{(t)}-\mathbf{r}||_1^2$$

Set $\delta:=\frac{\varepsilon^2}{2h^2}$ and apply Theorem 5. In $O\left(\frac{h^2\ln(n\rho/\nu)}{\varepsilon^2}\right)$ time we would get a matrix with $\delta>\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)})$ which from the above inequality would imply $||\mathbf{r}^{(t)}-\mathbf{r}||_1\le\varepsilon$. ◀

To prove Part 2, we need a way to relate the ℓ_2 norm and the KL-divergence. In order to do so, we prove a different lower bound which implies Pinsker's inequality (with a worse constant), but is significantly stronger in certain regimes. To the best of our knowledge this is a new bound which may be of independent interest in other domains. Below we state the version which we need for the proof of Theorem 1, part 2. This is an instantiation of the general inequality Lemma 9 whcih we prove in Section 3.

▶ **Lemma 8.** *Given any pair of probability distributions p,q over a finite domain, define $\mathcal{A}:=\{i:q_i>2p_i\}$ and $\mathcal{B}:=\{i:q_i\le 2p_i\}$. Then,*

$$\mathbf{D}_{KL}(p||q)\ge(1-\ln 2)\cdot\left(\sum_{i\in\mathcal{A}}|q_i-p_i|+\sum_{i\in\mathcal{B}}\frac{(q_i-p_i)^2}{p_i}\right) \tag{KL vs ℓ_1/ℓ_2}$$

Proof of Theorem 1, Part 2. We apply Lemma 8 on the vectors $\pi_{\mathbf{r}}$ and $\pi_{\mathbf{r}}^{(t)}$. Lemma 8 gives us

$$\begin{aligned}\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)}) &\geq C\cdot\left(\frac{1}{h}\sum_{i\in A}|\mathbf{r}_i^{(t)}-\mathbf{r}_i| \;+\; \frac{1}{h}\sum_{i\in B}\frac{(\mathbf{r}_i^{(t)}-\mathbf{r}_i)^2}{\mathbf{r}_i}\right)\\ &\geq \frac{C}{h}\left(\sum_{i\in A}|\mathbf{r}_i^{(t)}-\mathbf{r}_i| \;+\; \frac{1}{\rho}\sum_{i\in B}(\mathbf{r}_i^{(t)}-\mathbf{r}_i)^2\right)\end{aligned}$$

where $C = 1-\ln 2$. If the second summand in the parenthesis of the RHS is $\geq \frac{1}{2}||\mathbf{r}^{(t)}-\mathbf{r}||_2^2$, then we get $\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)}) \geq \frac{C}{2\rho h}||\mathbf{r}^{(t)}-\mathbf{r}||_2^2$. Otherwise, we have $\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)}) \geq \frac{C}{\sqrt{2}h}||\mathbf{r}^{(t)}-\mathbf{r}||_2$, where we used the weak fact that the sum of some positive numbers is at least the square-root of the sum of their squares. In any case, we get the following

$$\mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)}) \geq \min\left(\frac{C}{2\rho h}||\mathbf{r}^{(t)}-\mathbf{r}||_2^2, \frac{C}{\sqrt{2}h}||\mathbf{r}^{(t)}-\mathbf{r}||_2\right) \tag{2}$$

To complete the proof of part 2 of Theorem 1, set $\delta := \frac{C}{2\rho h\left(\frac{1}{\varepsilon}+\frac{1}{\varepsilon^2}\right)}$ and apply Theorem 5. In $O\left(\rho h \ln\left(n\rho/\nu\right)\cdot\left(\frac{1}{\varepsilon}+\frac{1}{\varepsilon^2}\right)\right)$ time we would get a matrix with $\delta \geq \mathbf{D}_{KL}(\pi_{\mathbf{r}}||\pi_{\mathbf{r}}^{(t)})$. If the minimum of the RHS of (2) is the first term, then we get $||\mathbf{r}^{(t)}-\mathbf{r}||_2^2 \leq \varepsilon^2$ implying the ℓ_2-error is $\leq \varepsilon$. If the minimum is the second term, then we get $||\mathbf{r}^{(t)}-\mathbf{r}||_2 \leq \frac{\varepsilon}{\sqrt{2}\rho} < \varepsilon$ since $\rho \geq 1$. ◀

3 New Lower Bound on the KL-Divergence

We now establish a new lower bound on KL-divergence which yields (KL vs ℓ_1/ℓ_2) as a corollary.

▶ **Lemma 9.** *Let p and q be two distributions over a finite n-element universe. For any fixed $\theta > 0$, define the sets $\mathcal{A}_\theta := \{i \in [n] : q_i \geq (1+\theta)p_i\}$ and $\mathcal{B}_\theta = [n] \setminus \mathcal{A}_\theta = \{i \in [n] : q_i \leq (1+\theta)p_i\}$. Then we have the following inequality*

$$\mathbf{D}_{KL}(p||q) \geq \left(1-\frac{\ln(1+\theta)}{\theta}\right)\cdot\left(\sum_{i\in\mathcal{A}_\theta}|q_i-p_i| + \frac{1}{\theta}\sum_{i\in\mathcal{B}_\theta}p_i\left(\frac{q_i-p_i}{p_i}\right)^2\right) \tag{3}$$

When $\theta = 1$, we get (KL vs ℓ_1/ℓ_2).

A Comparison of (Pinsker) and (KL vs ℓ_1/ℓ_2): To see why (KL vs ℓ_1/ℓ_2) generalizes (Pinsker) with a weaker constant, note that

$$||p-q||_1^2 = \left(\sum_{i\in\mathcal{A}}|q_i-p_i| + \sum_{i\in\mathcal{B}}|q_i-p_i|\right)^2 \leq 2\left(\sum_{i\in\mathcal{A}}|q_i-p_i|\right)^2 + 2\left(\sum_{i\in\mathcal{B}}p_i\frac{|q_i-p_i|}{p_i}\right)^2$$

The first parenthetical term above, since it is ≤ 1, is at most the first summation in the parenthesis of (KL vs ℓ_1/ℓ_2). The second parenthetical term above, by Cauchy-Schwarz, is at most the second summation in the parenthesis of (KL vs ℓ_1/ℓ_2). Thus (KL vs ℓ_1/ℓ_2) implies $\mathbf{D}_{KL}(p||q) \geq \frac{(1-\ln 2)}{2}||p-q||_1^2$. On the other hand, the RHS of (KL vs ℓ_1/ℓ_2) can be much larger than that of (Pinsker). For instance, suppose $p_i = 1/n$ for all i, $q_1 = 1/n + 1/\sqrt{n}$, and for $i \neq 1$, $q_i = 1/n - \frac{1}{(n-1)\sqrt{n}}$. The RHS of (Pinsker) is $\Theta(1/n)$ while that of (KL vs ℓ_1/ℓ_2) is $\Theta(1/\sqrt{n})$ which is the correct order of magnitude for $\mathbf{D}_{KL}(p||q)$.

Proof of Lemma 9: We need the following fact which follows from calculus; we provide a proof later for completeness.

▶ **Lemma 10.** *Given any $\theta > 0$, define $a_\theta := \frac{\ln(1+\theta)}{\theta}$ and $b_\theta := \frac{1}{\theta}\left(1 - \frac{\ln(1+\theta)}{\theta}\right)$. Then,*
- *For $t \geq \theta$, $(1+t) \leq e^{a_\theta t}$*
- *For $t \leq \theta$, $(1+t) \leq e^{t - b_\theta t^2}$*

Define $\eta_i := \frac{q_i - p_i}{p_i}$. Note that $\mathcal{A}_\theta = \{i : \eta_i > \theta\}$ and $\mathcal{B}_\theta$ is the rest. We can write the KL-divergence as follows

$$\mathbf{D}_{KL}(p||q) := \sum_{i=1}^{n} p_i \ln(p_i/q_i) = -\sum_{i=1}^{n} p_i \ln(1+\eta_i)$$

For $i \in \mathcal{A}_\theta$, since $\eta_i > \theta$, we upper bound $(1+\eta_i) \leq e^{a_\theta \eta_i}$ using Fact 10. For $i \in \mathcal{B}_\theta$, that is $\eta_i \leq \theta$, we upper bound $(1+\eta_i) \leq e^{\eta_i - b_\theta \eta_i^2}$ using Fact 10. Lastly, we note $\sum_i p_i\eta_i = 0$ since p, q both sum to 1, implying $\sum_{i\in\mathcal{B}_\theta} p_i\eta_i = -\sum_{i\in\mathcal{A}_\theta} p_i\eta_i$. Putting all this in the definition above we get

$$\mathbf{D}_{KL}(p||q) \geq -a_\theta \cdot \sum_{i\in\mathcal{A}_\theta} p_i\eta_i - \sum_{i\in\mathcal{B}_\theta} p_i\eta_i + b_\theta \sum_{i\in\mathcal{B}_\theta} p_i\eta_i^2 = (1-a_\theta)\sum_{i\in\mathcal{A}_\theta} p_i\eta_i + b_\theta \sum_{i\in\mathcal{B}_\theta} p_i\eta_i^2$$

The proof of inequality (3) follows by noting that $b_\theta = \frac{1-a_\theta}{\theta}$. ◀

Proof of Lemma 10. The proof of both facts follow by proving non-negativity of the relevant function in the relevant interval. Recall $a_\theta = \ln(1+\theta)/\theta$ and $b_\theta = \frac{1}{\theta}(1-a_\theta)$. We start with the following three inequalities about the log-function.

$$\text{For all } z > 0, \quad z + z^2/2 > (1+z)\ln(1+z) > z \quad \text{and} \quad \ln(1+z) > z - z^2/2 \tag{4}$$

The third inequality in (4) implies $a_\theta > 1 - \theta/2$ and thus, $b_\theta < 1/2$. The first inequality in (4) implies $a_\theta < \frac{1+\frac{\theta}{2}}{1+\theta}$ which in turn implies $b_\theta > 1/2(1+\theta)$. For brevity, henceforth let us lose the subscript on a_θ and b_θ.

Consider the function $f(t) = e^{at} - (1+t)$. Note that $f'(t) = ae^{at} - 1$ which is increasing in t since $a > 0$. So, for any $t \geq \theta$, we have $f'(t) \geq ae^{a\theta} - 1 = \frac{(1+\theta)\ln(1+\theta)}{\theta} - 1 \geq 0$, by the second inequality in (4). Therefore, f is increasing when $t \geq \theta$. The first part of Fact 10 follows since $f(\theta) = 0$ by definition of a.

Consider the function $g(t) = e^{t(1-bt)} - (1+t)$. Note that $g(0) = g(\theta) = 0$. We break the argument in two parts: we argue that $g(t)$ is strictly positive for all $t \leq 0$, and that $g(t)$ is strictly positive for $t \in (0, \theta)$. This will prove the second part of Fact 10.

The first derivative is $g'(t) = (1-2bt)e^{t(1-bt)} - 1$ and the second derivative is $g''(t) = e^{t(1-bt)}\left((1-2bt)^2 - 2b\right)$. Since $b < 1/2$, we have $2b < 1$, and thus for $t \leq 0$, $g''(t) > 0$. Therefore, g' is strictly increasing for $t \leq 0$. However, $g'(0) = 0$, and so $g'(t) < 0$ for all $t < 0$. This implies g is strictly decreasing in the interval $t < 0$. Noting $g(0) = 0$, we get $g(t) > 0$ for all $t < 0$. This completes the first part of the argument.

For the second part, we first note that $g'(\theta) < 0$ since $b > \frac{1}{2(1+\theta)}$. That is, g is strictly decreasing at θ. On the other hand g is increasing at θ. To see this, looking at g' is not enough since $g'(0) = 0$. However, $g''(0) > 0$ since $b < 1/2$. This means that 0 is a strict (local) minimum for g implying g is increasing at 0. In sum, g vanishes at 0 and θ, and is increasing at 0 and decreasing at θ. This means that if g does vanish at some $r \in (0, \theta)$, then it must vanish once again in $[r, \theta)$ for the it to be decreasing at θ. In particular, g' must vanish three times in $(0, \theta)$ and thus four times in $[0, \theta)$ since $g'(0) = 0$. This in turn implies g'' vanishes three times in $[0, \theta)$ which is a contradiction since g'' is a quadratic in t multiplied by a positive term.

We end by proving (4). This also follows the same general methodology. Define $p(z) := (1+z)\ln(1+z) - z$ and $q(z) := p(z) - z^2/2$. Differentiating, we get $p'(z) = \ln(1+z) > 0$ for all $z > 0$, and $q'(z) = \ln(1+z) - z < 0$ for all $z > 0$. Thus, p is increasing, and q is decreasing, in $(0, \infty)$. The first two inequalities of (4) follow since $p(0) = q(0) = 0$. To see the third inequality, define $r(z) = \ln(1+z) - z + z^2/2$ and observe $r'(z) = \frac{1}{1+z} - 1 + z = \frac{z^2}{1+z}$ which is > 0 if $z > 0$. Thus r is strictly increasing, and the third inequality of (4) follows since $r(0) = 0$. ◀

References

1 Scott Aaronson. Quantum computing and hidden variables. *Phys. Rev. A*, 71:032325, Mar 2005. doi:10.1103/PhysRevA.71.032325.

2 Zeyuan Allen Zhu, Yuanzhi Li, Rafael Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2017. URL: http://arxiv.org/abs/1704.02315.

3 Zeyuan Allen Zhu and Lorenzo Orecchia. Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, San Diego, CA, USA*, pages 1439–1456, 2015.

4 Michael Bacharach. Estimating nonnegative matrices from marginal data. *International Economic Review*, 6(3):294–310, 1965. URL: http://www.jstor.org/stable/2525582.

5 H. Balakrishnan, Inseok Hwang, and C. J. Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 5, pages 4874–4879 Vol.5, Dec 2004. doi:10.1109/CDC.2004.1429569.

6 R.B. Bapat and T.E.S. Raghavan. An extension of a theorem of Darroch and Ratcliff in loglinear models and its application to scaling multidimensional matrices. *Linear Algebra and its Applications*, 114:705–715, 1989. Special Issue Dedicated to Alan J. Hoffman. doi:10.1016/0024-3795(89)90489-8.

7 L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967. doi:10.1016/0041-5553(67)90040-7.

8 Richard A Brualdi, Seymour V Parter, and Hans Schneider. The diagonal equivalence of a nonnegative matrix to a stochastic matrix. *Journal of Mathematical Analysis and Applications*, 16(1):31–50, 1966. doi:10.1016/0022-247X(66)90184-3.

9 Michael B. Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2017. URL: http://arxiv.org/abs/1704.02310.

10 Imre Csiszar. I-divergence geometry of probability distributions and minimization problems. *Ann. Probab.*, 3(1):146–158, 02 1975. doi:10.1214/aop/1176996454.

11 Imre Csiszar. A geometric interpretation of darroch and ratcliff's generalized iterative scaling. *Ann. Statist.*, 17(3):1409–1413, 09 1989. doi:10.1214/aos/1176347279.

12 W. Edwards Deming and Frederick F. Stephan. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *Ann. Math. Statist.*, 11(4):427–444, 12 1940. doi:10.1214/aoms/1177731829.

13 Joel Franklin and Jens Lorenz. On the scaling of multidimensional matrices. *Linear Algebra and its Applications*, 114:717–735, 1989. Special Issue Dedicated to Alan J. Hoffman. doi:10.1016/0024-3795(89)90490-4.

14 Leonid Gurvits and Peter N. Yianilos. The deflation-inflation method for certain semidefinite programming and maximum determinant completion problems. Technical report, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, 1998.

15 Martin Idel. A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps. *ArXiv e-prints*, 2016. arXiv:1609.06349.

16 Bahman Kalantari and Leonid Khachiyan. On the rate of convergence of deterministic and randomized RAS matrix scaling algorithms. *Oper. Res. Lett.*, 14(5):237–244, 1993. doi:10.1016/0167-6377(93)90087-W.

17 Bahman Kalantari and Leonid Khachiyan. On the complexity of nonnegative-matrix scaling. *Linear Algebra and its Applications*, 240:87–103, 1996. doi:10.1016/0024-3795(94)00188-X.

18 Bahman Kalantari, Isabella Lari, Federica Ricca, and Bruno Simeone. On the complexity of general matrix scaling and entropy minimization via the RAS algorithm. *Technical Report, n.24, Department of Statistics and Applied probability, La Sapienza University, Rome*, 2002.

19 Bahman Kalantari, Isabella Lari, Federica Ricca, and Bruno Simeone. On the complexity of general matrix scaling and entropy minimization via the RAS algorithm. *Math. Program.*, 112(2):371–401, 2008. doi:10.1007/s10107-006-0021-4.

20 Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000. doi:10.1007/s004930070007.

21 Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 448–457. ACM, 1993. doi:10.1145/167088.167211.

22 Sally M Macgill. Theoretical properties of biproportional matrix adjustments. *Environment and Planning A*, 9(6):687–701, 1977. doi:10.1068/a090687.

23 Michael W. Mahoney, Satish Rao, Di Wang, and Peng Zhang. Approximating the Solution to Mixed Packing and Covering LPs in Parallel $O(\varepsilon^{-3})$ Time. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 52:1–52:14, 2016.

24 MV Menon. Reduction of a matrix with positive elements to a doubly stochastic matrix. *Proceedings of the American Mathematical Society*, 18(2):244–247, 1967.

25 Arkadi Nemirovskii and Uriel Rothblum. On complexity of matrix scaling. *Linear Algebra and its Applications.*, 302:435–460, 1999.

26 Juan de Dios Ortúzar and Luis G. Willumsen. *Modelling Transport.* John Wiley & Sons, Ltd, 2011. doi:10.1002/9781119993308.fmatter.

27 T.E.S. Raghavan. On pairs of multidimensional matrices. *Linear Algebra and its Applications*, 62:263–268, 1984. doi:10.1016/0024-3795(84)90101-0.

28 Günter Rote and Martin Zachariasen. Matrix scaling by network flow. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 848–854, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283474.

29 Uriel Rothblum and Hans Schneider. Scalings of matrices which have prespecified row sums and column sums via optimization. *Linear Algebra and its Applications*, 114:737–764, 1989. Special Issue Dedicated to Alan J. Hoffman.

30 Ludger Ruschendorf. Convergence of the iterative proportional fitting procedure. *Ann. Statist.*, 23(4):1160–1174, 1995. doi:10.1214/aos/1176324703.

31 Igal Sason and Sergio Verdú. Upper bounds on the relative entropy and rényi divergence as a function of total variation distance for finite alphabets. In *2015 IEEE Information Theory Workshop - Fall (ITW), Jeju Island, South Korea, October 11-15, 2015*, pages 214–218. IEEE, 2015. `doi:10.1109/ITWF.2015.7360766`.

32 Erwin Schrödinger. Über die umkehrung der naturgesetze. *Preuss. Akad. Wiss., Phys.-Math. Kl.*, 1931.

33 Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967. URL: `http://www.jstor.org/stable/2314570`.

34 Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific J. Math.*, 21(2):343–348, 1967. URL: `https://projecteuclid.org:443/euclid.pjm/1102992505`.

35 George W. Soules. The rate of convergence of sinkhorn balancing. *Linear Algebra and its Applications*, 150:3–40, 1991. `doi:10.1016/0024-3795(91)90157-R`.

36 Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *42nd Annual Symposium on Foundations of Computer Science, FOCS, Las Vegas, Nevada, USA*, pages 538–546, 2001.

Approximation Schemes for 0-1 Knapsack

Timothy M. Chan

Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
tmc@illinois.edu

Abstract

We revisit the standard 0-1 knapsack problem. The latest polynomial-time approximation scheme by Rhee (2015) with approximation factor $1+\varepsilon$ has running time near $\widetilde{O}(n+(1/\varepsilon)^{5/2})$ (ignoring polylogarithmic factors), and is randomized. We present a simpler algorithm which achieves the same result and is deterministic.

With more effort, our ideas can actually lead to an improved time bound near $\widetilde{O}(n+(1/\varepsilon)^{12/5})$, and still further improvements for small n.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases knapsack problem, approximation algorithms, optimization, (min,+)-convolution

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.5

1 Introduction

In the *0-1 knapsack* problem, we are given a set of n items where the i-th item has weight $w_i \le W$ and profit $p_i > 0$, and we want to select a subset of items with total weight bounded by W while maximizing the total profit. In other words, we want to maximize $\sum_{i=1}^n p_i\xi_i$ subject to the constraint that $\sum_{i=1}^n w_i\xi_i \le W$ over all $\xi_1,\ldots,\xi_n \in \{0,1\}$.

This classic textbook problem is among the most fundamental in combinatorial optimization and approximation algorithms, and is important for being one of the first NP-hard problems shown to possess *fully polynomial-time approximation schemes (FPTASs)*, i.e., algorithms with approximation factor $1+\varepsilon$ for any given parameter $\varepsilon \in (0,1)$, taking time polynomial in n and $\frac{1}{\varepsilon}$.

Despite all the attention the problem has received, the "fine-grained complexity" of FPTASs remains open: we still do not know the best running time as a function of n and $\frac{1}{\varepsilon}$. An $O(\frac{1}{\varepsilon}n^3)$-time algorithm via dynamic programming is perhaps the most often taught in undergraduate algorithm courses. The first published FPTAS by Ibarra and Kim [6] from 1975 required $\widetilde{O}(n+(\frac{1}{\varepsilon})^4)$ time, where the $\widetilde{O}$ notation hides polylogarithmic factors in n and $\frac{1}{\varepsilon}$. Lawler [12] subsequently obtained a small improvement, but only in the hidden polylogarithmic factors. For a long time, the record time bound was $\widetilde{O}(n+(\frac{1}{\varepsilon})^3)$ by Kellerer and Pferschy [10]. Recently, in a (not-too-well-known) Master's thesis, Rhee [14] described a new randomized algorithm running in $\widetilde{O}(n+(\frac{1}{\varepsilon})^{2.5})$ time. (Note that improved time bounds of this form tell us how much accuracy we can guarantee while keeping near-linear running time; for example, Rhee's result implies that a $(1+n^{-2/5})$-approximate solution can be found in $\widetilde{O}(n)$ time.)

In this paper, we give a new presentation of an algorithm that has the same running time as Rhee's, with the added advantages of being deterministic and simpler: One part of Rhee's algorithm relied on solving several linear programs with two constraints, using a Lagrangian relaxation and some sophisticated form of randomized binary search (although I suspect known low-dimensional linear programming techniques might help). In contrast,

Table 1 FPTASs for the 0-1 knapsack problem.

running time	reference	year
$n^{O(1/\varepsilon)}$	Sahni [15]	1975
$O(n\log n + (\frac{1}{\varepsilon})^4 \log\frac{1}{\varepsilon})$	Ibarra and Kim [6]	1975
$O(n\log\frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^4)$	Lawler [12]	1979
$O(n\log\frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^3 \log^2\frac{1}{\varepsilon})$	Kellerer and Pferschy [10]	2004
$O(n\log\frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^{5/2} \log^3\frac{1}{\varepsilon})$ (randomized)	Rhee [14]	2015
$O(n\log\frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^{5/2}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ (deterministic)	Section 4	
$O(n\log\frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^{12/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ (deterministic)	Appendix A	
$O(\frac{1}{\varepsilon}n^2)$	Lawler [12]	1979
$O((\frac{1}{\varepsilon})^2 n\log\frac{1}{\varepsilon})$	Kellerer and Pferschy [9]	1999
$\widetilde{O}(\frac{1}{\varepsilon}n^{3/2})$ (randomized)	Appendix B	
$O(((\frac{1}{\varepsilon})^{4/3}n + (\frac{1}{\varepsilon})^2)/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ (deterministic)	Appendix B	

our approach bypasses this part completely. Ironically, the "new" approach is just a simple combination of two previous approaches. Along the way, we also notice that the hidden polylogarithmic factors in the second term can be eliminated; in fact, we can get speedup of a *super*polylogarithmic factor ($2^{\Omega(\sqrt{\log(1/\varepsilon)})}$) by using the latest results on $(\min,+)$-*convolution* [2, 16], if we give up on simplicity.

In research, simplifying previous solutions can often serve as a starting point to obtaining new improved solutions. Indeed, by combining our approach with a few extra ideas, we can actually obtain a faster FPTAS for 0-1 knapsack running in $\widetilde{O}(n + (\frac{1}{\varepsilon})^{2.4})$ time. These extra ideas are interesting (relying on an elementary number-theoretic lemma), but since the incremental improvement is small and the algorithm is more complicated, we feel it is of secondary importance compared to the simpler $\widetilde{O}(n + (\frac{1}{\varepsilon})^{2.5})$ algorithm (in the true spirit of SOSA), and thus defer that result to the appendix. The appendix also describes some further improved bounds for small n (see the bottom half of Table 1).

In passing, we should mention two easier special cases. First, for the *subset sum* problem, corresponding to the case when $p_i = w_i$, Kellerer et al. [8] obtained algorithms with $\widetilde{O}(\frac{1}{\varepsilon}n)$ and $\widetilde{O}(n + (\frac{1}{\varepsilon})^2)$ running time. For the *unbounded* knapsack problem, where the variables ξ_i are unbounded nonnegative integers, Jansen and Kraft [7] obtained an $\widetilde{O}(n + (\frac{1}{\varepsilon})^2)$-time algorithm; the unbounded problem can be reduced to the 0-1 case, ignoring logarithmic factors [5]. These methods do not adapt to the general 0-1 knapsack problem.

2 Preliminaries

First we may discard all items with $p_i \le \frac{\varepsilon}{n}\max_j p_j$; this changes the optimal value by at most $\varepsilon\max_j p_j$, and thus at most a factor of $1+\varepsilon$. So we may assume that $\frac{\max_j p_j}{\min_j p_j} \le \frac{n}{\varepsilon}$. By rounding, we may assume that all p_i's are powers of $1+\varepsilon$. In particular, there are at most $m = O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$ distinct p_i values.

We adopt a "functional" approach in presenting our algorithms, which does not need explicit reference to dynamic programming, and makes analysis of approximation factors more elegant:

Given input $I = \{(w_1, p_1), \ldots, (w_n, p_n)\}$, we consider the more general problem of approximating the function

$$f_I(x) \;:=\; \max\left\{\sum_{i=1}^{n} p_i\xi_i \;:\; \sum_{i=1}^{n} w_i\xi_i \le x,\;\; \xi_1, \ldots, \xi_n \in \{0,1\}\right\}$$

for *all* $x \in \mathbb{R}$. Note that f_I is a monotone step function (in this paper, "monotone" always means "nondecreasing"). It is more convenient to define approximation from below: we say that a function $\widetilde{f}$ *approximates* a function f *with factor* $1+\varepsilon$ if $1 \le \frac{f(x)}{\widetilde{f}(x)} \le 1+\varepsilon$ for all $x \in \mathbb{R}$. We say that $\widetilde{f}$ *approximates* f *with additive error* δ if $0 \le f(x) - \widetilde{f}(x) \le \delta$.

We can merge f_I functions by the following easy observation: if I is the disjoint union of I_1 and I_2, then $f_I = f_{I_1} \oplus f_{I_2}$, where the operator $\oplus$ denotes the $(\max,+)$-*convolution*, defined by the formula

$$(f \oplus g)(x) \;=\; \max_{x' \in \mathbb{R}} \left(f(x') + g(x - x')\right).$$

In the "base case" when the p_i's are all equal to a common value p, the function f_I is easy to compute, by the obvious greedy algorithm: the function values are $-\infty, 0, p, 2p, \ldots, np$ and the x-breakpoints are $0, w_1, w_1 + w_2, \ldots, w_1 + \cdots + w_n$, after arranging the items in nondecreasing order of w_i. We say that a step function is *p-uniform* if the function values are of the form $-\infty, 0, p, 2p, \ldots, \ell p$ for some ℓ. Furthermore, we say that a p-uniform function is *pseudo-concave* if the sequence of differences of consecutive x-breakpoints is nondecreasing. When the p_i's are all equal, f_I is indeed uniform and pseudo-concave. Thus, the original problem reduces to computing a monotone step function that is a $(1+O(\varepsilon))$-factor approximation of the $\oplus$ of $m = O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$ uniform, pseudo-concave, monotone step functions.

The following facts provides the building blocks for all our algorithms.

▶ **Fact 1.** *Let f and g be monotone step functions with total complexity $O(\ell)$ (i.e., with $O(\ell)$ steps). We can compute $f \oplus g$ in*

(i) *$\ell^2/2^{\Omega(\sqrt{\log \ell})}$ time if f and g are p-uniform;*
(ii) *$O(\ell)$ time if f is p-uniform, and g is p-uniform and pseudo-concave;*
(iii) *$O((\ell + \ell' \cdot \frac{p'}{p})\log\frac{p'}{p})$ time if f is p-uniform, and g is p'-uniform and pseudo-concave with complexity ℓ', and p' is a multiple of p.*

Proof. Without loss of generality, assume that the ranges of f and g are $\{-\infty, 0, 1, 2, \ldots, \ell\}$.

(i) Define $f^{-1}(y)$ to be the smallest x with $f(x) = y$ (if no such x exists, define $f^{-1}(y)$ to be supremum of all x with $f(x) < y$). Define $g^{-1}(y)$ similarly. Both f^{-1} and g^{-1} can be generated in $O(\ell)$ time. We can compute the $(\min,+)$-convolution

$$(f \oplus g)^{-1}(y) \;=\; \min_{y' \in \{0,1,\ldots,\ell\}} \left(f^{-1}(y') + g^{-1}(y - y')\right)$$

for all $y \in \{0, 1, \ldots, 2\ell\}$ in $O(\ell^2)$ time naively. From $(f \oplus g)^{-1}$, we can obtain $f \oplus g$ in $O(\ell)$ time.

A slight speedup to $\ell^2/2^{\Omega(\sqrt{\log \ell})}$ time is known for the $(\min,+)$-convolution problem, by using Bremner et al.'s reduction to $(\min,+)$-matrix multiplication [2] and Williams' algorithm for the latter problem [16] (which was originally randomized but was derandomized later [4]). This improvement is not simple, however.

(ii) For this part, Kellerer and Pferschy [10] have already described an $O(\ell \log \ell)$-time algorithm (the extra logarithmic factor does not matter to us in the end), but actually we can directly reduce to a standard *matrix searching* problem [1]: computing the row minima in an $O(\ell) \times O(\ell)$ matrix A satisfying the *Monge property*. To compute the above $(\min,+)$-convolution, we can set $A[y,y'] = f^{-1}(y) + g^{-1}(y'-y)$, and observe that the Monge property $A[y,y'] + A[y+1,y'+1] \le A[y,y'+1] + A[y+1,y']$ is equivalent to $g^{-1}(y'-y) - g^{-1}(y'-y-1) \le g^{-1}(y'-y+1) - g^{-1}(y'-y)$, which corresponds precisely to the definition of pseudo-concavity of g. The well-known SMAWK algorithm [1] solves the matrix searching problem in $O(\ell)$ time.

(ii′) This part can be directly reduced to (ii) as follows. Say that a function h is *shifted-p-uniform* if $h + a$ is p-uniform for some value a. The *upper envelope* of $h_1, \ldots, h_m$ refers to the function $h(x) = \max\{h_1(x), \ldots, h_m(x)\}$.
We can express the given p-uniform function f as an upper envelope of $\frac{p'}{p}$ shifted-p'-uniform functions f_i, each with complexity $O(\ell \frac{p}{p'})$. For each i, we can compute $f_i \oplus g$ by (ii) (after shifting f_i) in $O(\ell \frac{p}{p'} + \ell')$ time. The total time is $O(\frac{p'}{p} \cdot (\ell \frac{p}{p'} + \ell'))$. We can then return the upper envelope of all these functions $f_i \oplus g$. Note that the upper envelope of $\frac{p'}{p}$ step functions can be constructed in time linear in their total complexity times $\log \frac{p}{p'}$, by sweeping the breakpoints from left to right, using a priority queue to keep track of the current maximum. ◀

3 Two Known Methods with Exponent 3

We begin with two simple approximation approaches, one of which uses Fact 1(i) and the other uses Fact 1(ii′).

▶ **Lemma 1.** *Let f and g be monotone step functions with total complexity ℓ and ranges contained in $\{-\infty, 0\} \cup [A, B]$. Then we can compute a monotone step function that approximates $f \oplus g$ with factor $1 + O(\varepsilon)$ and complexity $\widetilde{O}(\frac{1}{\varepsilon})$ in*

(i) $O(\ell) + \widetilde{O}((\frac{1}{\varepsilon})^2 / 2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ *time in general;*

(ii) $O(\ell) + \widetilde{O}(\frac{1}{\varepsilon})$ *time if g is p-uniform and pseudo-concave.*[1]

Proof. For a given $b \in [A, B]$, we first describe how to compute an approximation[2] of $\min\{f \oplus g, b\}$ with additive error $O(\varepsilon b)$ and complexity $O(\frac{1}{\varepsilon})$:

(i) In the general case, we just round the function values of $\min\{f, b\}$ and $\min\{g, b\}$ down to multiples of εb (in $O(\ell)$ time). The new functions $\min\{f, b\}$ and $\min\{g, b\}$ are (εb)-uniform with complexity $O(\frac{1}{\varepsilon})$. We can then compute $\min\{f \oplus g, b\}$ by Fact 1(i) in $O((\frac{1}{\varepsilon})^2 / 2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time.

(ii) In the case when g is p-uniform and pseudo-concave, we consider two subcases:
- CASE 1: $p \ge \varepsilon b$. We may assume that p is a multiple of εb, by adjusting ε by an $O(1)$ factor. We round the function values of $\min\{f, b\}$ down to multiples of εb. The new function f is (εb)-uniform. We can then compute $\min\{f \oplus g, b\}$ by Fact 1(ii′) in $\widetilde{O}(\frac{1}{\varepsilon} + \frac{b}{p} \cdot \frac{p}{\varepsilon b}) = \widetilde{O}(\frac{1}{\varepsilon})$ time.

1 Throughout, we use the $\widetilde{O}$ notation to hide polylogarithmic factors not just in n and $\frac{1}{\varepsilon}$, but also other parameters such as $\frac{B}{A}$ and $\frac{1}{\delta_0}$. Eventually, these parameters will be set to values which are polynomial in n and $\frac{1}{\varepsilon}$.

2 $\min\{f, b\}$ denotes the function F with $F(x) = \min\{f(x), b\}$.

- Case 2: $\varepsilon b > p$. We may assume that εb is a multiple of p, by adjusting ε by an $O(1)$ factor. We can round the function values of $\min\{g, b\}$ down to multiples of εb while preserving pseudo-concavity (since each difference of consecutive x-breakpoints in the new function is the sum of $\frac{\varepsilon b}{p}$ differences in the old function); the rounding causes additive error $O(\varepsilon b)$. We have now effectively made p equal to εb, and so Case 1 applies.

To finish the proof of (i) or (ii), we apply the above procedure to every $b \in [A, B]$ that is a power of 2, and return the upper envelope of the resulting $O(\log \frac{B}{A})$ functions. This gives a $(1 + O(\varepsilon))$-factor approximation of $f \oplus g$ (since in the case when the function value lies between $b/2$ and b, the $O(\varepsilon b)$ additive error for $\min\{f \oplus g, b\}$ implies approximation factor $1 + O(\varepsilon)$). The running time increases by a factor of $O(\log \frac{B}{A})$. ◀

▶ **Lemma 2.** *Let $f_1, \ldots, f_m$ be monotone step functions with total complexity $O(n)$ and ranges contained in $\{-\infty, 0\} \cup [A, B]$. Then we can compute a monotone step function that approximates $f_1 \oplus \cdots \oplus f_m$ with complexity $\widetilde{O}(\frac{1}{\varepsilon})$ in*

(i) *$O(n) + \widetilde{O}((\frac{1}{\varepsilon})^2 m / 2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time in general;*

(ii) *$O(n) + \widetilde{O}(\frac{1}{\varepsilon} m^2)$ time if every f_i is p_i-uniform and pseudo-concave for some p_i.*

Proof.

(i) We use a simple divide-and-conquer algorithm: recursively approximate $f_1 \oplus \cdots \oplus f_{m/2}$ and $f_{m/2+1} \oplus \cdots \oplus f_m$, and return a $(1 + O(\varepsilon))$-factor approximation of the $\oplus$ of the two resulting functions, by using Lemma 1(i). Since the recursion tree has $O(m)$ nodes each with cost $\widetilde{O}((\frac{1}{\varepsilon})^2 / 2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ (except for the leaf nodes, which have a total additional cost $O(n)$), the total time is $O(n) + \widetilde{O}(m(\frac{1}{\varepsilon})^2 / 2^{\Omega(\sqrt{\log(1/\varepsilon)})})$. However, since the depth of the recursion is $\log m$, the approximation factor increases to $(1+O(\varepsilon))^{\log m} = 1 + O(\varepsilon \log m)$. We can adjust ε by a factor of $\log m$, which increases the running time only by polylogarithmic factors.

(ii) We use a simple incremental algorithm: initialize $f = f_1$; for each $i = 2, \ldots, m$, compute a $(1 + O(\varepsilon))$-factor approximation of $f \oplus f_i$, by using Lemma 1(ii), and reset f to this new function. The total time is $O(n) + \widetilde{O}(m \cdot \frac{1}{\varepsilon})$. However, the approximation factor increases to $(1+O(\varepsilon))^m = 1+O(\varepsilon m)$. We can adjust ε by a factor of m, which increases the running time to $O(n) + \widetilde{O}(m \cdot \frac{1}{\varepsilon/m})$. ◀

Both the divide-and-conquer and incremental methods in Lemmas 2(i) and (ii) are known, or are reinterpretations of known methods [9, 10, 14]. The divide-and-conquer method is similar to the "merge-and-reduce" technique often used in streaming (and in fact immediately implies a space-efficient streaming algorithm for the 0-1 knapsack problem). As $m = \widetilde{O}(\frac{1}{\varepsilon})$, both method happen to yield an 0-1 knapsack algorithm with roughly the same time bound, near $\widetilde{O}(n + (\frac{1}{\varepsilon})^3)$.

4 A Simpler Algorithm with Exponent 5/2

To improve the running time, we use a very simple idea: just combine the two methods!

▶ **Theorem 3.** *Let $f_1, \ldots, f_m$ be monotone step functions with total complexity $O(n)$ and ranges contained in $\{-\infty, 0\} \cup [A, B]$. If every f_i is p_i-uniform and pseudo-concave for some p_i, then we can compute a monotone step function that approximates $f_1 \oplus \cdots \oplus f_m$ with factor $1 + O(\varepsilon)$ and complexity $\widetilde{O}(\frac{1}{\varepsilon})$ in $O(n) + \widetilde{O}((\frac{1}{\varepsilon})^{3/2} m / 2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time.*

Proof. Divide the set of given functions into r subsets of $\frac{m}{r}$ functions, for a parameter r to be specified later. For each subset, approximate the $\oplus$ of its $\frac{m}{r}$ pseudo-concave functions by Lemma 2(ii). Finally, return an approximation of the $\oplus$ of the r resulting functions, by using Lemma 2(i). The total time is

$$O(n) + \widetilde{O}\left(r\,\frac{1}{\varepsilon}\left(\frac{m}{r}\right)^2 + (r-1)\left(\frac{1}{\varepsilon}\right)^2 \Big/ 2^{\Omega(\sqrt{\log(1/\varepsilon)})} \right).$$

Setting $r = \left\lceil \sqrt{\varepsilon} m 2^{c\sqrt{\log(1/\varepsilon)}} \right\rceil$ for a sufficiently small constant c yields the theorem. ◀

▶ **Corollary 4.** *There is a $(1+\varepsilon)$-approximation algorithm for 0-1 knapsack with running time $O(n\log\frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^{5/2}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$.*

Proof. We apply the theorem with $m = \widetilde{O}(\frac{1}{\varepsilon})$ and $\frac{B}{A} = O(\frac{n^2}{\varepsilon})$. Initial sorting of the w_i's takes $O(n\log n)$ time. (Note that we may assume $n \le (\frac{1}{\varepsilon})^{O(1)}$, for otherwise we can switch to Lawler's algorithm [12]. In particular, $\log n = O(\log\frac{1}{\varepsilon})$.) ◀

This completes the description of our new simpler algorithm.

5 Closing Remarks

We have described how to compute approximations of the optimal value, but not a corresponding subset of items. To output the subset, we can modify the algorithms to record extra information whenever we apply Fact 1 to compute the $\oplus$ of two functions f and g. Namely, for each step in the step function $f \oplus g$, we store the corresponding steps from f and g that define its y-value. Then a solution achieving the returned profit value can be retrieved by proceeding backwards in a straightforward way (as in most dynamic programming algorithms). Since we have performed a total of $\widetilde{O}(m)$ $\oplus$ operations to functions with complexity $\widetilde{O}(\frac{1}{\varepsilon})$, the total space usage is $O(n) + \widetilde{O}(\frac{1}{\varepsilon}m) = O(n) + \widetilde{O}((\frac{1}{\varepsilon})^2)$. (The space bound can probably be reduced by known space-reduction techniques [13, 9] on dynamic programming.)

The main open question is whether the running time can be improved to near $O(n+(\frac{1}{\varepsilon})^2)$. Our improvements in the appendix will hopefully inspire future work. Note that any improved subquadratic algorithm for $(\min,+)$-convolution would automatically lead to further improvements on the time bounds of our algorithms. The truly subquadratic algorithm by Chan and Lewenstein [3] for bounded monotone integer sequences does not seem applicable here for arbitrary weights, unfortunately. In the opposite direction, a variant of a recent reduction of Cygan et al. [5] or Künnemann et al. [11] shows that there is no algorithm for 0-1 (or unbounded) knapsack with $O((n+\frac{1}{\varepsilon})^{2-\delta})$ running time, assuming the conjecture that there is no truly subquadratic algorithm for $(\min,+)$-convolution.

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/s00453-012-9734-3.

3 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

4 Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255. SIAM, 2016. doi:10.1137/1.9781611974331.ch87.

5 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 22:1–22:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.22.

6 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.

7 Klaus Jansen and Stefan Erich Julius Kraft. A faster FPTAS for the unbounded knapsack problem. In Zsuzsanna Lipták and William F. Smyth, editors, *Combinatorial Algorithms - 26th International Workshop, IWOCA 2015, Verona, Italy, October 5-7, 2015, Revised Selected Papers*, volume 9538 of *Lecture Notes in Computer Science*, pages 274–286. Springer, 2015. doi:10.1007/978-3-319-29516-9_23.

8 Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003. doi:10.1016/S0022-0000(03)00006-0.

9 Hans Kellerer and Ulrich Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *J. Comb. Optim.*, 3(1):59–71, 1999. doi:10.1023/A:1009813105532.

10 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004. doi:10.1023/B:JOCO.0000021934.29833.6b.

11 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

12 Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979. doi:10.1287/moor.4.4.339.

13 M. J. Magazine and O. Oguz. A fully polynomial approximation algorithm for the 0-1 knapsack problem. *Europ. J. Oper. Res.*, 123:325–332, 2000. doi:10.1016/0377-2217(84)90286-8.

14 Donguk Rhee. Faster fully polynomial approximation schemes for knapsack problems. Master's thesis, MIT, 2015. URL: https://dspace.mit.edu/bitstream/handle/1721.1/98564/920857251-MIT.pdf.

15 Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *J. ACM*, 22(1):115–124, 1975. doi:10.1145/321864.321873.

16 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673. ACM, 2014. doi:10.1145/2591796.2591811.

A An Improved Algorithm with Exponent 12/5

In the appendix, we show how the ideas in our $\widetilde{O}(n + (\frac{1}{\varepsilon})^{5/2})$ algorithm can lead to further improvements.

In what follows, we make an extra input assumption that all the p_i's are within a constant factor of each other. This case is sufficient to solve the general problem, because we can divide the input items into $O(\log \frac{n}{\varepsilon})$ classes with the stated property, and then merge via Lemma 2(i). By rescaling, we assume that all p_i's are in $[1, 2]$. In this case, the optimal fractional solution approximates the optimal integral solution with $O(1)$ additive error (since rounding the fractional solution causes the loss of at most one item), and the optimal fractional solution can be found by the standard greedy algorithm. In other words, with $O(1)$ additive error, we can approximate f_I by the step function with function values $-\infty, 0, p_1, p_1 + p_2, \ldots, p_1 + \cdots + p_n$ and the x-breakpoints $0, w_1, w_1 + w_2, \ldots, w_1 + \cdots + w_n$, after arranging the items in nondecreasing order of w_i/p_i. A solution with $O(1)$ additive error has approximation factor $1 + O(\varepsilon)$ if the optimal value is $\Omega(\frac{1}{\varepsilon})$. Thus, we may assume that the optimal value is upper-bounded by $B = O(\frac{1}{\varepsilon})$.

A.1 Refining the Second Method

To obtain further improvement, we will refine the second incremental method in Lemma 2(ii). Recall that the inefficency of that method is due to the need to round in every iteration. We observe that if all the p_i's are integer multiples of a small set of values, we do not need to round as often, as explained in the following lemma.

For a set Δ, we say that p is a Δ-*multiple* if it is a multiple of δ for some $\delta \in \Delta$.

▶ **Lemma 5.** *Let $f_1, \ldots, f_m$ be monotone step functions and ranges contained in $\{-\infty, 0\} \cup [1, B]$. Let $\Delta \subset [\delta_0, 2\delta_0]$ and let $b \in [1, B]$. If every f_i is p_i-uniform and pseudo-concave for some $p_i \in [1, 2]$ which is a Δ-multiple, then we can compute a monotone step function that approximates $\min\{f_1 \oplus \cdots \oplus f_m, b\}$ with additive error $O(|\Delta|\delta_0)$ in $\widetilde{O}(\frac{1}{\delta_0} bm)$ time.*

Proof. We use a simple incremental algorithm: Initialize $f = -\infty$. In each iteration, take one $\delta \in \Delta$. Round the function values of $\min\{f, b\}$ down to multiples of δ, which incurs an additive error of $O(\delta) = O(\delta_0)$. The new function $\min\{f, b\}$ is now δ-uniform, with complexity $O(\frac{b}{\delta})$. For each not-yet-considered function f_i with p_i being a multiple of δ, reset f to $\min\{f \oplus f_i, b\}$, which can be computed exactly by Lemma 1(ii′) in $\widetilde{O}(\frac{b}{\delta} + \frac{b}{p_i} \cdot \frac{p_i}{\delta}) = \widetilde{O}(\frac{b}{\delta_0})$ time. Repeat for the next $\delta \in \Delta$. The total time is $\widetilde{O}(\frac{b}{\delta_0} m)$. The total additive error is $O(|\Delta|\delta_0)$. ◀

A.2 A Number-Theoretic Lemma

To use the above lemma efficiently, we need the following combinatorial/number-theoretic lemma, stating that all numbers can be approximated well by integer multiples of a small set of values.

▶ **Lemma 6.** *Given $\varepsilon < \delta_0 < 1$, there exists a set $\Delta \subset [\delta_0, 2\delta_0]$ of size $\widetilde{O}(\frac{\delta_0}{\varepsilon})$, such that every value $p \in [1, 2]$ can be approximated by a Δ-multiple with additive error $O(\varepsilon)$.*

The set can be constructed in $\widetilde{O}(\frac{1}{\varepsilon})$ time by a randomized algorithm.

Proof. Let $P = \{1, 1 + \varepsilon, 1 + 2\varepsilon, 1 + 3\varepsilon, \ldots, 1 + \lfloor \frac{1}{\varepsilon} \rfloor \varepsilon\}$. Then $|P| = O(\frac{1}{\varepsilon})$. In the stated property, it suffices to consider only values p in P.

Given $p \in P$ and $\delta \in [\delta_0, 2\delta_0]$, p is approximated by a multiple of δ with additive error ε iff $0 \le p - i\delta \le \varepsilon$ for some integer i, i.e., iff δ lies in the set

$$I_p \;:=\; [\delta_0, 2\delta_0] \;\cap\; \bigcup_i \left[\frac{p-\varepsilon}{i}, \frac{p}{i}\right]$$

where the union is over all integers i between $\frac{1-\varepsilon}{2\delta_0}$ and $\frac{2}{\delta_0}$. Our goal is to find a small set Δ of size $\widetilde{O}(\frac{\delta_0}{\varepsilon})$ that hits I_p for all $p \in P$.

Now, each set I_p is a union of $\Theta(\frac{1}{\delta_0})$ disjoint intervals of length $\Theta(\frac{\varepsilon}{1/\delta_0}) = \Theta(\varepsilon\delta_0)$ and thus has measure $\Theta(\varepsilon)$. Thus, a uniformly distributed $\delta \in [\delta_0, 2\delta_0]$ lies in I_p with probability $\Theta(\frac{\varepsilon}{\delta_0})$. For a simple randomized construction, we can just choose $O(\frac{\delta_0}{\varepsilon}\log|P|)$ values uniformly from $[\delta_0, 2\delta_0]$ and obtain a set Δ that hits every I_p with high probability $1 - O(\frac{1}{|P|^c})$ for any constant $c > 1$. This yields a Monte Carlo algorithm, but it can be converted to Las Vegas, since we can verify correctness of Δ by generating and sorting all Δ-multiples in $[1,2]$ in $\widetilde{O}(|\Delta|\frac{1}{\delta_0}) = \widetilde{O}(\frac{1}{\varepsilon})$ time. ◀

A.3 Putting the Refined Second Method Together

Applying Lemma 5 together with Lemma 6 now gives the following new result:

▶ **Lemma 7.** *Let $f_1, \ldots, f_m$ be monotone step functions with ranges contained in $\{-\infty, 0\} \cup [1, B]$. If every f_i is p_i-uniform and pseudo-concave for some $p_i \in [1,2]$, then we can compute a monotone step function that approximates $\min\{f_1 \oplus \cdots \oplus f_m, B\}$ with factor $1 + O(\varepsilon)$ and complexity $\widetilde{O}(\frac{1}{\varepsilon})$ in $\widetilde{O}(\frac{1}{\varepsilon}\sqrt{B}m)$ expected time, assuming $B = \widetilde{O}(\frac{1}{\varepsilon})$.*

Proof. For a given $b \in [1, B]$, we first describe how to compute an approximation of $\min\{f_1 \oplus \cdots \oplus f_m, b\}$ with additive error $O(\varepsilon b)$ and complexity $O(\frac{1}{\varepsilon})$:

- Construct the set Δ of size $\widetilde{O}(\frac{\delta_0}{\varepsilon})$ from Lemma 6 in $\widetilde{O}(\frac{1}{\varepsilon})$ expected time for some parameter $\delta_0 > \varepsilon$ to be specified later. Generate and sort all Δ-multiples in $[1,2]$ in $\widetilde{O}(|\Delta|\frac{1}{\delta_0}) = \widetilde{O}(\frac{1}{\varepsilon})$ time. For each p_i, round it down to a Δ-multiple with additive error $O(\varepsilon)$ (e.g., by binary search) and change f_i accordingly. This multiplies the approximation factor by $1 + O(\varepsilon)$, and thus increases the additive error by at most $O(\varepsilon b)$. Now apply Lemma 5. The additive error is $O(|\Delta|\delta_0) = O(\frac{\delta_0^2}{\varepsilon}) = O(\varepsilon b)$ by choosing $\delta_0 := \varepsilon\sqrt{b}$. The running time is $\widetilde{O}(\frac{1}{\delta_0}bm) = \widetilde{O}(\frac{1}{\varepsilon}\sqrt{b}m)$. Note that the complexity of the resulting function can be reduced to $O(\frac{1}{\varepsilon})$ by rounding down to multiples of εb.

To finish, we apply the above procedure to every $b \in [1, B]$ that is a power of 2, and then return the upper envelope of the resulting $O(\log B)$ functions. This gives a $(1 + O(\varepsilon))$-factor approximation of $\min\{f_1 \oplus \cdots \oplus f_m, B\}$. The running time increases by a factor of $O(\log B)$. ◀

As $m = \widetilde{O}(\frac{1}{\varepsilon})$ and $B = O(\frac{1}{\varepsilon})$ in our application, the above lemma immediately gives an alternative algorithm with near $\widetilde{O}(n + (\frac{1}{\varepsilon})^{5/2})$ running time. Notice that this alternative is based solely on the second incremental method, without combining with the first divide-and-conquer method. Naturally, it suggests the possibility that a combination might lead to a further improvement...

A.4 Putting Everything Together

To this end, we first show that if the size of Δ could be reduced (from $O(\frac{\delta_0}{\varepsilon})$ to, say, $O(\frac{\delta_0}{r\varepsilon})$) for some particular choice of δ_0, then Lemma 7 could be improved (from $\widetilde{O}(\frac{1}{\varepsilon}\sqrt{B}m)$ time to $\widetilde{O}(\frac{1}{r^{1/4}\varepsilon}\sqrt{B}m)$), by bootstrapping:

▶ **Lemma 8.** *Let $f_1, \dots, f_m$ be monotone step functions with ranges contained in $\{-\infty, 0\} \cup [1, B]$. Let $\Delta \subset [\delta_0, 2\delta_0]$ be a set of size $O(\frac{\delta_0}{r\varepsilon})$ for some $r \in [1, B^2]$ where $\delta_0 := r^{1/4}\varepsilon\sqrt{B}$. If every f_i is p_i-uniform and pseudo-concave for some $p_i \in [1,2]$ which is a Δ-multiple, then we can compute a monotone step function that approximates $\min\{f_1 \oplus \cdots \oplus f_m, B\}$ with factor $1 + O(\varepsilon)$ and complexity $\widetilde{O}(\frac{1}{\varepsilon})$ in $\widetilde{O}(\frac{1}{r^{1/4}\varepsilon}\sqrt{B}m)$ expected time, assuming $B = \widetilde{O}(\frac{1}{\varepsilon})$.*

Proof.

1. First compute an approximation of $\min\{f_1 \oplus \cdots \oplus f_m, B/s\}$ with factor $1 + O(\varepsilon)$ and complexity $\widetilde{O}(\frac{1}{\varepsilon})$ by Lemma 7 in $\widetilde{O}(\frac{1}{\varepsilon}\sqrt{B/s}\, m)$ time, for some parameter $s \geq 1$ to be specified later.
2. Next compute an approximation of $\min\{f_1 \oplus \cdots \oplus f_m, B\}$ with additive error $O(\varepsilon B/s)$. This can be done by applying Lemma 5. The additive error is $O(|\Delta|\delta_0) = O(\frac{\delta_0^2}{r\varepsilon}) = O(\varepsilon B/s)$ by choosing $\delta_0 := \varepsilon\sqrt{(r/s)B}$. The running time is $\widetilde{O}(\frac{1}{\delta_0}Bm) = \widetilde{O}(\frac{1}{\varepsilon}\sqrt{(s/r)B}m)$.

To finish, we return the upper envelope of the two resulting functions. This gives a $(1+O(\varepsilon))$-factor approximation of $\min\{f_1 \oplus \cdots \oplus f_m, B\}$ (since in the case when the function value exceeds B/s, the $O(\varepsilon B/s)$ additive error in the second function implies $1+O(\varepsilon)$ approximation factor). Note that the complexity of the resulting function can be reduced to $\widetilde{O}(\frac{1}{\varepsilon})$ by rounding down to powers of $1 + \varepsilon$, which multiplies the approximation factor by $1 + O(\varepsilon)$.

The total running time

$$\widetilde{O}\left(\frac{1}{\varepsilon}\sqrt{B/s}\, m + \frac{1}{\varepsilon}\sqrt{(s/r)B}m\right)$$

is $\widetilde{O}(\frac{1}{r^{1/4}\varepsilon}\sqrt{B}m)$ by setting $s := \sqrt{r}$. ◀

To reduce the size of Δ, we combine the above with the first divide-and-conquer method from Lemma 2(ii), which finally leads to our new improved result after fine-tuning the choice of parameters:

▶ **Theorem 9.** *Let $f_1, \dots, f_m$ be monotone step functions with ranges contained in $\{-\infty, 0\} \cup [A, B]$. If every f_i is p_i-uniform and pseudo-concave with $p_i \in [1, 2]$, then we can compute a monotone step function that approximates $\min\{f_1 \oplus \cdots \oplus f_m, B\}$ with factor $1 + O(\varepsilon)$ and complexity $\widetilde{O}(\frac{1}{\varepsilon})$ in $\widetilde{O}((\frac{1}{\varepsilon})^{12/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ expected time if $m, B = \widetilde{O}(\frac{1}{\varepsilon})$.*

Proof. Construct the set Δ of size $\widetilde{O}(\frac{\delta_0}{\varepsilon})$ from Lemma 6 with $\delta_0 := r^{1/4}\varepsilon\sqrt{B}$ for some parameter r to be specified later. Generate and sort all Δ-multiples in $[1, 2]$ in $\widetilde{O}(|\Delta|\frac{1}{\delta_0}) = \widetilde{O}(\frac{1}{\varepsilon})$ time. For each p_i, round it down to a Δ-multiple with additive error $O(\varepsilon)$ and change f_i accordingly. This multiplies the approximation factor by $1 + O(\varepsilon)$.

Divide Δ into r subsets $\Delta_1, \dots, \Delta_r$ each of size $\widetilde{O}(\frac{\delta_0}{r\varepsilon})$. For each subset Δ_j, approximate the $\oplus$ of all not-yet-considered functions f_i with p_i being a Δ_j-multiple, by Lemma 8. Finally, return an approximation of the $\oplus$ of the resulting r functions by using Lemma 2(i). The total time is

$$\widetilde{O}\left(\frac{1}{r^{1/4}\varepsilon}\sqrt{B}m + r\left(\frac{1}{\varepsilon}\right)^2/2^{\Omega(\sqrt{\log(1/\varepsilon)})}\right) = \widetilde{O}\left(\frac{1}{r^{1/4}}\left(\frac{1}{\varepsilon}\right)^{5/2} + r\left(\frac{1}{\varepsilon}\right)^2/2^{\Omega(\sqrt{\log(1/\varepsilon)})}\right). \tag{1}$$

Setting $r = \left\lceil (\frac{1}{\varepsilon})^{2/5}2^{c\sqrt{\log(1/\varepsilon)}} \right\rceil$ and $\delta_0 = \varepsilon^{2/5}2^{(c/3)\sqrt{\log(1/\varepsilon)}}$ for a sufficiently small constant c yields the theorem. ◀

▶ **Corollary 10.** *There is a $(1+\varepsilon)$-approximation algorithm for 0-1 knapsack with expected running time $O(n\log\frac{1}{\varepsilon}+(\frac{1}{\varepsilon})^{12/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$.*

Proof. In the case when all $p_i\in[1,2]$, we apply the theorem with $m=\widetilde{O}(\frac{1}{\varepsilon})$ and $B=O(\frac{1}{\varepsilon})$. In addition, we take the greedy approximation and return the upper envelope of the two resulting functions. As noted earlier, the general case reduces to the $p_i\in[1,2]$ case, by merging $O(\log\frac{n}{\varepsilon})$ functions via Lemma 2(i), taking additional $(\frac{1}{\varepsilon})^2/2^{\Omega(\sqrt{\log(1/\varepsilon)})}$ time. Initial sorting takes $O(n\log n)$ time. (As before, we may assume $n\le(\frac{1}{\varepsilon})^{O(1)}$, for otherwise we can switch to Lawler's algorithm.) ◀

A.5 Derandomization

Our algorithm is randomized because of Lemma 6. In the proof of Lemma 6, instead of random sampling, we can run the standard greedy algorithm for hitting set, with $O(\frac{\delta_0}{\varepsilon}\log|P|)$ iterations. We gather all the intervals of I_p over all $p\in P$. In each iteration, we find a *deepest* point λ, i.e., a point that hits the most intervals, and delete the intervals in all the sets I_p that are hit by λ. Initially, the total number of intervals is $O(\frac{1}{\delta_0}|P|)=O(\frac{1}{\delta_0\varepsilon})$, and this bounds the total number of deletions as well. It is not difficult to design a data structure that supports deletions, searching for the deepest point, and searching for the intervals hit by a given point, all in logarithmic time per operation. Thus, the total time is $\widetilde{O}(\frac{1}{\delta_0\varepsilon})$, which is at most $\widetilde{O}((\frac{1}{\varepsilon})^2)$.

This adds an $\widetilde{O}((\frac{1}{\varepsilon})^2)$ term to the time bounds of Lemmas 7 and 8, and an $\widetilde{O}(r(\frac{1}{\varepsilon})^2)$ to (1), which slightly affects the final bound in the extra superpolylogarithmic factors. We can fix this by modifying the proof of Lemma 7: if $b\ge(\frac{1}{\varepsilon})^{0.1}$, we proceed as before and notice that the construction time for Δ is $\widetilde{O}(\frac{1}{\delta_0\varepsilon})\le O(\frac{1}{\varepsilon^{2-\Omega(1)}})$; but if $b<(\frac{1}{\varepsilon})^{0.1}$, we can switch to using a singleton set $\Delta=\{\varepsilon\}$ with $\delta_0=\varepsilon$, which leads to running time $\widetilde{O}(\frac{1}{\varepsilon}bm)\le\widetilde{O}((\frac{1}{\varepsilon})^{1.1}m)$. All this adds an extra $\widetilde{O}((\frac{1}{\varepsilon})^{1.1}m+r\cdot(\frac{1}{\varepsilon})^{2-\Omega(1)})$ term to (1), which does not affect the final bound.

▶ **Corollary 11.** *The algorithm in Corollary 10 can be made deterministic.*

As before, the algorithm can be modified to compute not just an approximation of the optimal value but also a corresponding subset of items.

B Variants for Small n

We note two further results which are better when n is small relative to $\frac{1}{\varepsilon}$.

▶ **Corollary 12.** *There is a $(1+\varepsilon)$-approximation algorithm for 0-1 knapsack with expected running time $\widetilde{O}(\frac{1}{\varepsilon}n^{3/2})$.*

Proof. In the case when all $p_i\in[1,2]$, an $\widetilde{O}(\frac{1}{\varepsilon}n^{3/2})$ time bound follows directly from Lemma 7, since the number of distinct p_i values is $m\le n$, and a trivial upper bound for the maximum optimal value is $B\le 2n$.

As noted earlier, the general case reduces to the $p_i\in[1,2]$ case, by merging $O(\log\frac{n}{\varepsilon})$ functions via Lemma 2(i), taking additional $(\frac{1}{\varepsilon})^2/2^{\Omega(\sqrt{\log(1/\varepsilon)})}$ time. To avoid the merging cost, we need to bypass Lemma 2(i). First, we can easily generalize Lemmas 5 and 7 to compute $f\oplus f_1\oplus\cdots\oplus f_m$ for an arbitrary monotone step function f with complexity $\widetilde{O}(\frac{1}{\varepsilon})$. We can then apply Lemma 7 iteratively, with each iteration handling all $p_i\in[2^j,2^{j+1}]$ (which can be rescaled to $[1,2]$), in increasing order of j. The approximation factor increases to $(1+\varepsilon)^{O(\log\frac{B}{A})}=1+O(\varepsilon\log\frac{B}{A})$. We can adjust ε by a logarithmic factor. ◀

▶ **Corollary 13.** *There is a $(1+\varepsilon)$-approximation algorithm for 0-1 knapsack with running time $O(((\frac{1}{\varepsilon})^{4/3}n + (\frac{1}{\varepsilon})^2)/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$.*

Proof. Divide the input items into r subsets of $\frac{n}{r}$ items each. For each subset, apply the method from Corollary 12. Finally, return an approximation of the $\oplus$ of the resulting r functions by using Lemma 2(i). The total time is

$$\widetilde{O}\left(r\,\frac{1}{\varepsilon}\left(\frac{n}{r}\right)^{3/2} + r\left(\frac{1}{\varepsilon}\right)^2 / 2^{\Omega(\sqrt{\log(1/\varepsilon)})} \right).$$

Setting $r = \left\lceil \varepsilon^{2/3} n 2^{c\sqrt{\log(1/\varepsilon)}} \right\rceil$ for a sufficiently small constant c yields the corollary. This algorithm can be made deterministic as in Section A.5. The derandomization adds an extra $\widetilde{O}((\frac{1}{\varepsilon})^{1.1}m + r\cdot(\frac{1}{\varepsilon})^{2-\Omega(1)})$ term, which does not affect the final bound. ◀

Corollary 12 gives the current best bound for $n \ll (\frac{1}{\varepsilon})^{2/3}$, and Corollary 13 gives the current best bound for $(\frac{1}{\varepsilon})^{2/3} \ll n \ll (\frac{1}{\varepsilon})^{16/15}$. For example, when $n = \frac{1}{\varepsilon}$, Corollary 13 gives $\widetilde{O}((\frac{1}{\varepsilon})^{7/3})$, which is a little better than $\widetilde{O}((\frac{1}{\varepsilon})^{12/5})$. This case is of interest, since for certain related problems such as subset-sum or unbounded knapsack (but unfortunately not for the general 0-1 knapsack problem), there are efficient preprocessing algorithms that can reduce the input size n to $\widetilde{O}(\frac{1}{\varepsilon})$.

Counting Solutions to Polynomial Systems via Reductions[*]

Richard Ryan Williams

MIT CSAIL & EECS, Cambridge, MA, USA
rrw@mit.edu

Abstract

This paper provides both positive and negative results for counting solutions to systems of polynomial equations over a finite field. The general idea is to try to reduce the problem to counting solutions to a *single* polynomial, where the task is easier. In both cases, simple methods are utilized that we expect will have wider applicability (far beyond algebra).

First, we give an efficient deterministic reduction from approximate counting for a system of (arbitrary) polynomial equations to approximate counting for *one* equation, over any finite field. We apply this reduction to give a deterministic $\text{poly}(n, s, \log p)/\varepsilon^2$-time algorithm for approximately counting the fraction of solutions to a system of s quadratic n-variate polynomials over $\mathbb{F}_p$ (the finite field of prime order p) to within an additive ε factor, for any prime p. Note that uniform random sampling would already require $\Omega(s/\varepsilon^2)$ time, so our algorithm behaves as a full derandomization of uniform sampling. The approximate-counting algorithm yields efficient approximate counting for other well-known problems, such as 2-SAT, NAE-3SAT, and 3-Coloring. As a corollary, there is a deterministic algorithm (with analogous running time) for producing solutions to such systems which have at least εp^n solutions.

Second, we consider the difficulty of exactly counting solutions to a single polynomial of constant degree, over a finite field. (Note that *finding* a solution in this case is easy.) It has been known for over 20 years that this counting problem is already NP-hard for degree-three polynomials over $\mathbb{F}_2$; however, all known reductions increased the number of variables by a considerable amount. We give a subexponential-time reduction from counting solutions to k-CNF formulas to counting solutions to a degree-$k^{O(k)}$ polynomial (over any finite field of $O(1)$ order) which *exactly preserves* the number of variables. As a corollary, the Strong Exponential Time Hypothesis (even its weak counting variant #SETH) implies that counting solutions to constant-degree polynomials (even over $\mathbb{F}_2$) requires essentially 2^n time. Similar results hold for counting orthogonal pairs of vectors over $\mathbb{F}_p$.

1998 ACM Subject Classification G.1.5 Numerical Analysis – Roots of Nonlinear Equations, I.1.2 Symbolic and Algebraic Manipulation – Algorithms

Keywords and phrases counting complexity, polynomial equations, finite field, derandomization, strong exponential time hypothesis

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.6

1 Introduction

A canonical problem in pseudorandomness is:

Given a class $\mathcal{C}$ of Boolean circuits, is there a deterministic and efficient method for approximately counting the fraction of satisfying assignments to any circuit from $\mathcal{C}$?

[*] Supported by NSF CAREER Grant CCF-1741615. A talk on an early version of this work can be found at https://youtu.be/gJxpUhc1Gfc

1st Symposium on Simplicity in Algorithms (SOSA 2018).
Editor: Raimund Seidel; Article No. 6; pp. 6:1–6:15
OASICS Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

By uniformly random sampling $\Theta(1/\varepsilon^2)$ inputs to a given circuit, we can easily obtain an additive ε-approximation to the fraction of satisfying assignments, with high probability. Thus the above problem amounts to deterministically achieving what trivial random sampling can provide in a natural setting, with the target of reaching poly$(n)/\varepsilon^2$ time (or perhaps even better in some cases where randomness can also achieve more, such as approximately counting knapsack solutions [20]).

The general problem of finding such algorithms has been studied for decades. Of most relevance to this paper are the prior results for AC^0 circuits [2, 30, 8], for CNF/DNF of bounded or unbounded width [28, 23, 32, 21], and for $\mathbb{F}_q$ polynomials of constant degree [29, 6, 7, 33]. Approximate counting algorithms with only a $1/\varepsilon^2$ dependence in the running time are not known for any of these problems except in the case of degree-two polynomials, where one can *exactly* count solutions over a finite field in polynomial time (see the Preliminaries). We would like to "lift" this nice algorithm to more expressive representations.

In the first part of this paper, we study approximate counting for an NP-hard problem in algebra:

Counting Solutions to Multivariate Quadratic Systems (#MQS)
Given: A system of degree-two polynomials over $\mathbb{F}_p[x_1, \ldots, x_n]$, for some prime p
Output: The number of solutions to the system.

The decision version of #MQS ("is there a solution?") is well-known to be NP-hard, and is of interest in several theoretical and practical areas (see [27] for references). With regards to approximating #MQS, the best deterministic algorithm in the literature is due to Viola [33], who gave a pseudorandom generator for fooling a degree-d polynomial with seed length at least $d \log n + d2^d \log(1/\varepsilon)$. Since the Fourier spectrum of the AND function has low ℓ_1-norm (see for example [9]), a pseudorandom generator for a single polynomial extends to a system of polynomials, yielding a deterministic approximation algorithm for the fraction of #MQS solutions with running time poly$(n) \cdot \varepsilon^{-8}$.

Approximately Solving #MQS

The first main result of this paper is that the $1/\varepsilon^2$ dependence of the random sampling algorithm for #MQS can be matched in a deterministic way.

▶ **Theorem 1.** *For every prime p, there is a deterministic algorithm running in poly$(n, s, \log p)/\varepsilon^2$ time for approximately counting the fraction of solutions to a system of s quadratic equations in n variables over $\mathbb{F}_p$.*

As a corollary, one can also efficiently and deterministically *find* solutions to any system of quadratic equations, provided there are many solutions:

▶ **Corollary 2.** *For every prime p, there is a deterministic algorithm running in poly$(n, s, \log p)/\varepsilon^2$ time which, given any system of s quadratic equations in n variables over $\mathbb{F}_p$ with at least εp^n solutions, outputs a solution.*

Recalling that more common counting problems such as #2-SAT and #NAE-3-SAT can easily be expressed as an instance of #MQS with no increase in the number of variables, Theorem 1 implies improved approximation algorithms (in terms of ε) for those problems as well: the best known approximate counting algorithm for general k-SAT is that of Trevisan [32] which has an $1/\varepsilon^{k2^k \ln(4)}$ dependence in the running time. (Note that Viola's

PRG is slightly faster in terms of ε.) For a non-binary example, the 3-coloring problem can be represented as a system of quadratic polynomials over $\mathbb{F}_3$ (for each edge (i, j) in your graph, include a polynomial $P(x_i, x_j)$ which is 0 if and only if $x_i \neq x_j$). In general, constraint satisfaction problems over a prime-order domain and two variables per constraint are handled by Theorem 1.

The key to Theorem 1 is a reduction from approximate counting for a system of degree-d polynomials to approximate counting for a *single* polynomial:

▶ **Theorem 3.** *For all primes p, integers $d > 0$, and $\varepsilon \in (0, 1)$, there exists a deterministic poly$(n, s, \log p)$-time reduction from approximately counting solutions to systems of degree-d equations over $\mathbb{F}_p$ to within an ε factor, to approximately counting solutions to* one *degree-d equation over $\mathbb{F}_p$ to within a $\Theta(\varepsilon^3/s^2)$ factor.*

Theorem 3 is appealing for several reasons.[1]

- First, it is somewhat surprising at first glance. For example, *detecting* feasibility of a system of quadratic polynomials (over any field) is NP-hard, but detecting the feasibility of only one such polynomial is trivial. Thus in some cases, our reduction efficiently reduces an NP-hard problem to an easy one — but only for the task of approximate counting.
- Second, the proof is extremely simple in hindsight. The central idea consists of applying the known constructions of *small-biased sets* in just the right way to solve the problem. We also show how such sets yield new schemes for approximating the intersection of a set family.
- Third, Theorem 3 is extremely generic, in comparison with its application (Theorem 1): it works for any polynomial system of *any* degree.

Theorem 1 follows from applying the reduction of Theorem 3 and an exact counting algorithm for #MQS instances with only one equation. We certainly do not obtain a pseudorandom generator in this way, but we do get a considerably different perspective on approximate counting in this domain. (We also do not use any algebraic geometry tools, which often appear in the literature on counting solutions to polynomial systems.)

SETH-Hardness of Exactly Counting Roots of One O(1)-Degree Polynomial

Complementing the above approximation algorithm, we use similar ideas to give a strong hardness reduction for exactly counting solutions (zeroes) of degree-d polynomials over a finite field of constant order, for constant $d > 2$. Finding a solution to such a polynomial is not hard: by a variant of the Schwartz-Zippel-DeMillo-Lipton lemma (see for example [36]), a not-identically-zero polynomial p of degree-d is nonzero on at least a $1/2^d$-fraction of points in $\{0, 1\}^n$, so it is easy to find a nonzero of p with randomness over any small field. (It is also easy to find a zero of p in $\{0, 1\}^n$ as well, by considering the polynomial $1 - p^{q-1}$ where q is the order of the field.)

Ehrenfeucht and Karpinski [17] showed that the solution-counting problem is already NP-hard for a single degree-3 polynomial over $\mathbb{F}_2$. However, all reductions from k-SAT to the counting problem (to our knowledge) increased the number of variables by a polynomial

[1] Note that over the real numbers, it is easy to reduce a system of polynomial equations $\{p_i(x_1, \ldots, x_n) = 0\}$ of degree d to a single equation of degree $2d$, by simply taking the sum of squares of the p_i. This is not useful for us, since (1) it does not work over a finite field and (2) in order for the approximation algorithm to work, we need a reduction that does not increase the degree.

factor; in the worst case, $\Omega(n)$ new variables are introduced. Here we give a much improved reduction in terms of the number of variables, sacrificing a bit in the degree:

▶ **Theorem 4.** *Let q be a prime power and $\varepsilon > 0$ be arbitrarily small. There is an $O(q^{\varepsilon n})$-time deterministic reduction from #k-SAT instances with n variables to the problem of counting roots to a $\mathbb{F}_q$-polynomial of degree $q(k/\varepsilon)^{O(k)}$ with n variables.*

In fact, the proof of Theorem 4 provides a subexponential-time reduction from the problem of exactly counting solutions to systems of $O(n)$ degree-k equations to that of exactly counting the zeros of one polynomial of degree $q(k/\varepsilon)^{O(k)}$, similarly to how Theorem 3 gives a polynomial-time reduction from approximate counting a degree-k system to approximate counting for a single degree-k polynomial. The high-level structure of the two reductions are similar as well: both reductions output a linear combination of the outputs of their oracle calls. However, the actual oracle calls themselves are quite different. Theorem 3 uses small-biased sets to construct a polynomial number of oracle calls, and obtain an approximate solution count; Theorem 4 uses a multiplication trick so that the number of oracle calls is only subexponential, while preserving the exact count. (The multiplication trick is the reason why the degree of the underlying polynomials increases to $q(k/\varepsilon)^{O(k)}$.)

From Theorem 4, it follows that the Strong Exponential Time Hypothesis (even its counting variant #SETH) predicts that for all $\varepsilon > 0$, there is a constant $d_\varepsilon > 2$ such that counting Boolean solutions to degree-d_ε polynomials (even over $\mathbb{F}_2$) cannot be done in $(2-\varepsilon)^n$ time.[2]

Under #ETH (the hypothesis that counting 3-SAT solutions requires $2^{\Omega(n)}$ time), it is known (for example) that counting the number of independent sets of size $n/3$ in an n-node graph requires $2^{\Omega(n)}$ time [24], #2-SAT requires $2^{\Omega(n)}$ time [15], the permanent of $n \times n$ Boolean matrices requires $2^{\Omega(n)}$ [15], and counting the number of perfect matchings in graphs with m edges requires $2^{\Omega(m)}$ time [14]. The reductions proving these conditional lower bounds generally introduce a minor (linear) increase in the relevant parameter n. Theorem 4 gives a tight lower bound for counting solutions to polynomials, under SETH.

Hardness for Counting Orthogonal Vectors over $\mathbb{F}_p$

Let p be any (small) prime. To demonstrate the range of the simple ideas in our reductions, we also use them to show that *exactly counting* pairs of $O(\log n)$-dimensional vectors with inner product zero *modulo p* (among a given set of n vectors) requires $n^{2-o(1)}$ time under SETH. This follows from the theorem:

▶ **Theorem 5.** *Let p be prime, and let $\ell \in [1, d]$ be an integer dividing d. There is an $\tilde{O}(n \cdot \ell 2^{d/\ell} \cdot p^\ell)$-time reduction from #OV with n vectors in d dimensions to p^ℓ instances of #OVp, each with n vectors in $\ell 2^{d/\ell} + 1$ dimensions.*

▶ **Corollary 6.** *Let $\varepsilon > 0$ be sufficiently small. There is an $\tilde{O}(n^{1+\varepsilon} \cdot p^{O(c/\varepsilon)})$-time reduction from #OV with n vectors in $c \log n$ dimensions to n^ε instances of #OVp, each with n vectors in $O(p^{c/\varepsilon} \log(n))$ dimensions.*

[2] The Strong Exponential Time Hypothesis [12] (SETH) states that for every $\varepsilon > 0$, there is a $k > 2$ such that k-SAT cannot be solved in $(2-\varepsilon)^n$ time; it is a vast strengthening of $\mathsf{P} \neq \mathsf{NP}$ and a mild strengthening of ETH [25] which states that there is an $\varepsilon > 0$ such that 3-SAT cannot be solved in $(1+\varepsilon)^n$ time. The "counting variant" #SETH states the same strong lower bound for the problem of *counting* the number of solutions to a k-SAT instance.

This reduction is in stark contrast with the fact that *detecting* a pair of vectors with inner product zero modulo p can be accomplished in nearly-linear time when the vectors have $n^{o(1)}$ dimension [35]. One can either view this result as evidence that the counting problem is hard, or as (yet) another angle towards refuting SETH.

2 Preliminaries

Exact counting for degree-two equations

Our approximate counting algorithm will use the fact that the exact counting problem for a single degree-two equation over a finite field $\mathbb{F}_q$ is polynomial-time solvable:

▶ **Theorem 7** (Woods [37], p.6). *For every prime power q, there is a deterministic $n^3 \cdot$ poly$(\log q)$-time algorithm for counting the number of solutions to a given degree-two polynomial over $\mathbb{F}_q$.*

Ehrenfeucht and Karpinski [17] covered the case of $\mathbb{F}_2$, and showed that exact counting for a degree-three polynomial is already NP-hard. Cai, Chen, Lipton, and Lu [10] gave an algorithm for the counting problem that works over $\mathbb{Z}_m$, for any fixed integer $m > 1$. Woods remarks that his algorithm essentially follows from placing the matrices defining the degree-two polynomial in Jordan canonical form, in which case the number of solutions can be more-or-less read off from the form obtained.

Small-bias sets

For our approximation algorithm, we need explicit constructions of small sets of vectors which closely approximate the uniform distribution of vectors, with respect to inner products over a finite field.

▶ **Definition 8.** A set $S \subseteq \mathbb{F}_q^n$ is *ε-biased* if for all $u \in \mathbb{F}_q^n$ and $r \in \mathbb{F}_q^n$,

$$\left| \Pr_{v \in S}[\langle u, v\rangle = r] - \Pr_{v \in \mathbb{F}_q^n}[\langle u, v\rangle = r] \right| \leq \varepsilon.$$

(Note that sometimes a more general definition is given, involving the characters of $\mathbb{F}_p^n$, but the above simple condition is implied by it. See [18].) The following simple consequences of being ε-biased will be important:

- $\Pr_{v \in S}[\langle \vec{0}, v\rangle = 0] = 1$, where $\vec{0}$ is the all-zeroes vector.
- For all $u \neq \vec{0}$ and $r \in \mathbb{F}_q$, $\Pr_{v \in S}[\langle u, v\rangle = r] \in (1/q - \varepsilon, 1/q + \varepsilon)$.

We also use deterministic explicit constructions of ε-biased sets over $\mathbb{F}_p$, for every prime p.

▶ **Theorem 9** ([4, 18, 5]). *For every prime p, and every $\varepsilon \in (0, 1/p^n)$, there is a (poly$(n, \log p)/\varepsilon^2$)-time constructible set of vectors $S \subseteq \mathbb{F}_p^n$ of cardinality $O(n^2/\varepsilon^2)$ that is ε-biased.*

It will be important that the ε-dependencies in the above theorem are only $1/\varepsilon^2$, but this can be achieved. For example, the constructions based on linear feedback shift registers of [4] (which are easily generalized to $\mathbb{F}_p$, see [18]) take all vectors $x, y \in \mathbb{F}_p^d$, where $d = \log_p(n/\varepsilon)$ and y corresponds to the coefficient vector of a monic irreducible $\mathbb{F}_p$-polynomial of degree d. The n-length vector $v_{x,y}$ of the ε-biased set S is generated by repeatedly computing inner products of y with vectors made up of previously computed inner products, up to n times.

To construct this set S, the main difficulty is constructing the y's, which we can do by enumerating all monic $\mathbb{F}_p$-polynomials of degree d, and throwing out those with non-trivial divisors. Rabin's test for irreducibility ([31]) would take $O(d^2 \cdot \mathrm{poly}(\log n, \log p))$ time. There are $O(n/\varepsilon)$ such polynomials to enumerate, and since $\varepsilon \geq 1/p^n$ we have $d \leq O(n)$, so this step takes $O(n^3/\varepsilon) \cdot \mathrm{poly}(\log n, \log p)$ time. The remaining list of monic irreducible polynomials (paired up with all possible vectors $x \in \mathbb{F}_p^d$) forms our ε-biased S, and each component of each n-length vector $v_{x,y}$ in S is just an inner product of two known d-length vectors. The running time of this construction is therefore $O(n^4/\varepsilon^2)$ (omitting $\mathrm{poly}(\log p)$ factors).

3 Approximating #MQS: Reduction and Algorithm

We begin with the proof of Theorem 3, which reduces the counting problem for a system of equations to the counting problem for a single equation.

Let $S = \{v_1, \ldots, v_m\} \subseteq \mathbb{F}_p^s$ be an ε-biased set of vectors. Let $\{p_1(y) = 0, \ldots, p_s(y) = 0\}$ be a system of degree-d equations over $\mathbb{F}_p$ in n variables, and let $A \subseteq \mathbb{F}_p^n$ be the set of solutions to the system. For each $i = 1, \ldots, m$, define the polynomial

$$P_i(y) = \sum_j v_i[j] \cdot p_j(y).$$

We observe two distinct properties of solutions and non-solutions to the original system:

(a) Every $y \in A$ is a solution of the equation $P_i(y) = 0$ and not of the equation $P_i(y) = 1$, for all $i = 1, \ldots, m$. This follows because for all $y \in A$, $p_1(y) = \cdots = p_s(y) = 0$.

(b) For every $y \notin A$, there are integers $N_0, N_1 \in [m(1/p - \varepsilon), m(1/p + \varepsilon)]$ such that y is a solution to N_0 of the m equations $P_i(y) = 0$ and is a solution to N_1 of the m equations $P_i(y) = 1$. To see this, note that for $y \notin A$, the vector $u = [p_1(y), \ldots, p_s(y)]$ is not all-zeroes. Thus for any $r \in \mathbb{F}_p$ we have $\Pr_{i \in [m]}[\langle u, v_i \rangle = r] \in (1/p + \varepsilon, 1/p - \varepsilon)$, because S is ε-biased.

Given the ability to count solutions to one degree-d equation, here is an algorithm for approximately counting solutions to a system of equations:

```
1. Construct the $\varepsilon$-biased set $S = \{v_1, \ldots, v_m\} \subset \mathbb{F}_p^s$.
2. Count the number of solutions to the equation $P_i(y) = 0$, for all $i = 1, \ldots, m$.
   Let $Z$ be the sum of all these numbers.
3. Count the number of solutions to the equation $P_i(y) = 1$, for all $i = 1, \ldots, m$.
   Let $O$ be the sum of all these numbers.
4. Output $(Z - O)/m$.
```

Now we analyze the algorithm. Let Z_i (respectively, O_i) be the number of solutions to the equation $P_i(y) = 0$ (respectively, $P_i(y) = 1$), for all $i, \ldots, m$. Our algorithm outputs the quantity:

$$\frac{1}{m}\left(\sum_i Z_i - \sum_i O_i\right). \tag{1}$$

By property (a), every $y \in A$ contributes 1 to the sum $\frac{1}{m} \cdot \sum_i Z_i$, and contributes 0 to the sum $\sum_i O_i$. By property (b), every $y \notin A$ contributes a value $z_y \in [1/p - \varepsilon, 1/p + \varepsilon]$ to the sum $\frac{1}{m} \cdot \sum_i Z_i$, and contributes a value $o_y \in [1/p - \varepsilon, 1/p + \varepsilon]$ to the sum $\frac{1}{m} \cdot \sum_i O_i$. We can therefore re-express (1) as:

$$\frac{1}{m}\left(\sum_i Z_i - \sum_i O_i\right) = |A| + \sum_{y \notin A}(z_y - o_y).$$

Given the bounds on z_y's and o_y's, we can easily upper-bound and lower-bound (1):

$$|A| + \sum_{y \notin A}(z_y - o_y) \leq |A| + \sum_{y \notin A}((1/p + \varepsilon) - (1/p - \varepsilon)) = |A| + |\overline{A}| \cdot 2\varepsilon$$

and

$$|A| + \sum_{y \notin A}(z_y - o_y) \geq |A| + \sum_{y \notin A}((1/p - \varepsilon) - (1/p + \varepsilon)) = |A| - |\overline{A}| \cdot 2\varepsilon.$$

It follows that the algorithm outputs a number that approximates the fraction of solutions to within $\pm 2\varepsilon$.

Moreover, observe that to obtain an approximate answer, we do not need an *exact* algorithm for counting solutions to one equation: if our algorithm for one equation always outputs approximations that are within $\varepsilon/(2m)$ of the exact count, then each of the $2m$ Z_i and O_i terms will be computed to within an $\varepsilon/(2m)$ factor, and the output will still be within $\pm 3\varepsilon$ of the exact fraction. This completes the proof of Theorem 3.

To obtain the final algorithm (Theorem 1) for approximately computing #MQS, we simply apply Theorem 7 to count the number of satisfying assignments to a single quadratic equation over $\mathbb{F}_p$ in $n^3 \cdot \text{poly}(\log p)$ time. Using this algorithm in the above reduction, we get an approximate counting algorithm running in time $\tilde{O}(s^2/\varepsilon^2 \cdot (n^3 + s) + t(s, 1/\varepsilon, p))$, where $t(s, 1/\varepsilon, p)$ is the time needed to construct an ε-biased set over $\mathbb{F}_p^s$. This completes the proof of Theorem 1.

3.1 A succinct approximate inclusion-exclusion

The reduction of Theorem 3 works by approximately representing the cardinality of the *intersection* of s equations by a linear combination of cardinalities on single $\mathbb{F}_p$-equations. Along the lines of the work of Linial and Nisan [26] on approximate inclusion-exclusion via low-degree polynomials over the reals, the ideas of Theorem 3 imply a variant of the inclusion-exclusion principle. However, unlike Linial and Nisan, our approximation of the cardinality of the intersection has only polynomially many terms.

To simplify the discussion, here we consider just the case of $\mathbb{F}_2$, and consider unions instead of intersections. Over $\mathbb{F}_2$, we will demonstrate how a small-bias set lets us "approximately" express the cardinality of a union of a set collection as a short linear combination of cardinalities of what one might call "oddtersections" of sub-collections of sets.

Let $(x \bmod 2) : \mathbb{Z} \to \{0, 1\}$ map integers to bits in the natural way. In Theorem 3, we are effectively using a representation of the AND function as a short linear combination of PARITY functions (see, for instance, Alon and Bruck [3]). Below is a representation of the OR function (which is analogous):

▶ **Lemma 10.** *For all $n \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, there is a $(poly(n)/\varepsilon^2)$-time constructible collection of subsets $S_1, \ldots, S_m \subseteq [n]$, with $m \leq O(n^2/\varepsilon^2)$, such that for every $x \in \{0, 1\}^n$,*

$$\left|\left(\bigvee_{i=1}^{n} x_i\right) - \sum_{i=1}^{m} \frac{2}{m} \cdot \left(\sum_{j \in S_i} x_j \bmod 2\right)\right| \leq \varepsilon. \qquad (2)$$

Proof. Let $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq [n]$ be a set family whose corresponding indicator vectors in $\{0,1\}^n$ form an $(\varepsilon/2)$-biased set. By Theorem 9, we can take $m \leq O(n^2/\varepsilon^2)$. Observe:

- If $(x_1, \ldots, x_n) = \vec{0}$ then for all $i = 1, \ldots, m$, $(\sum_{j \in S_i} x_j \bmod 2) = 0$, so $\sum_{i=1}^{m} \frac{2}{m} \cdot \left(\sum_{j \in S_i} x_j \bmod 2\right) = 0$.
- If $(x_1, \ldots, x_n) \neq \vec{0}$ then by properties of $(\varepsilon/2)$-biased sets, the number of $i \in [m]$ such that $\sum_{j \in S_i} x_j \neq |S_i| \pmod 2$ is in the interval $[m/2 - \varepsilon m/2, m/2 + \varepsilon m/2]$. So in this case,

$$\sum_{i=1}^{m} \frac{2}{m} \cdot \left(\sum_{j \in S_i} x_j \bmod 2\right) \in \left[\frac{2}{m} \cdot \left(\frac{m - \varepsilon m}{2}\right), \frac{2}{m} \cdot \left(\frac{m + \varepsilon m}{2}\right)\right]$$
$$= [1 - \varepsilon, 1 + \varepsilon].$$

This completes the proof. ◀

Let $A_1, \ldots, A_k$ be any sets over a finite universe U, and define their *oddtersection* to be

$$\bigoplus_i A_i = \{x \in U \mid x \text{ appears in an odd number of the } A_i\text{'s}\}.$$

The upshot of Lemma 10 is that we can write:

$$\left|\bigcup_i A_i\right| \approx_\varepsilon \sum_{i=1}^{m} \frac{1}{m} \cdot \left|\bigoplus_{j \in S_i} A_j\right|, \tag{3}$$

where the $\approx_\varepsilon$ means that the two quantities are within $\varepsilon|U|$ of each other. (Note that $|\cup_i A_i|$ is the sum over all $y \in U$ of $\bigvee_{i=1}[y \in A_i]$, where $[y \in A_i]$ is 1 if $y \in A$, and 0 otherwise. Invoking (2) on each term in this sum, we obtain the right-hand side of (3) to within an additive $\pm\varepsilon|U|$ factor.) Thus we can approximately represent the cardinality of a union of sets in a sparse way, as "oddities" of various sub-collections. It seems likely that this observation has more applications. For example, equation (3) immediately implies that we can reduce approximate counting for k-DNF formulas (with additive error) to approximate counting for degree-k polynomials over $\mathbb{F}_2$ (with additive error), by letting A_i be the set of satisfying assignments to the ith clause of a DNF.

3.2 Producing a solution when there are many

Given Theorem 1, one can obtain a deterministic algorithm for producing a solution to a quadratic system given that it has many solutions, using a self-reducibility argument.[3]

Reminder of Corollary 2. *For every prime p, constant k, and fraction $\varepsilon \in [1/p^n, 1]$, there is a deterministic algorithm running in poly$(n, s, \log p)/\varepsilon^2$ time which, given any system of s quadratic equations in n variables over $\mathbb{F}_p$ with at least εp^n solutions, outputs a solution.*

Proof. Suppose we are given a system over the variables $x_1, \ldots, x_n$ with $S \geq \varepsilon \cdot p^n$ solutions, where $\varepsilon \geq 1/p^n$.

[3] This reduction is apparently folklore. See also Goldreich [19, Theorem 3.5] for a generic reduction from "search-to-decision" in this setting.

For each $a \in \mathbb{F}_p$, assign $x_1 := a$ in all equations of the system, and run the polynomial-time approximate counting algorithm of Theorem 1 with error parameter $\alpha := \varepsilon/(2n)$. Let $x_1 := a_1$ be the assignment that yields the largest count from the algorithm. (If the count returned is zero for all $a \in \mathbb{F}_p$, return *fail*.) Analogously, set the variables $x_2 := a_2, \ldots, x_{n-k} := a_{n-k}$ one at a time, for $k = 2\log_p(1/\varepsilon)$, always taking the assignment that yields the largest count. Finally, try all $p^k = p^{2\log_p(1/\varepsilon)} \leq 1/\varepsilon^2$ assignments on the remaining k variables, and return any solution found.

Given Theorem 1, it is clear that the algorithm runs in the desired time. Now we turn to correctness. The algorithm began with a guarantee of S solutions. At least one setting of the variable x_1 yields a system on $n-1$ variables with at least S/p solutions. So after setting x_1 to maximize the number of solutions returned by the algorithm, the number of solutions in the remaining $(n-1)$-variable system is at least $S_1 = S/p - \alpha \cdot p^{n-1}$. Similarly, after setting x_1 and x_2 appropriately, the number of solutions in the remaining system on $n-2$ variables is at least

$$S_2 = S_1/p - \alpha \cdot p^{n-2} = S/p^2 - \alpha \cdot p^{n-2} - \alpha \cdot p^{n-2} = S/p^2 - 2\alpha \cdot p^{n-2}.$$

After setting $x_1, \ldots, x_i$ for $i = 1, \ldots, n$, we are inductively guaranteed that the number of remaining solutions in the system is at least

$$S_i = S_{i-1}/p - \alpha \cdot p^{n-i} = S/p^i - \alpha \cdot i \cdot p^{n-i}.$$

For $i = n - 2\log_p(1/\varepsilon)$, the number of solutions remaining is at least

$$\varepsilon p^n/p^i - \alpha \cdot i \cdot p^{n-i} \geq \varepsilon \cdot p^{2\log_p(1/\varepsilon)} - \alpha \cdot np^{2\log_p(1/\varepsilon)} \geq (\varepsilon - \alpha n) \cdot 1/\varepsilon^2.$$

For $\alpha = \varepsilon/(2n)$, the number of solutions remaining after setting $x_1, \ldots, x_i$ is at least $1/\varepsilon^2 \cdot (\varepsilon/2) \geq 1/(2\varepsilon)$, i.e., the number is non-zero. Therefore the algorithm returns a solution, if there are at least εp^n solutions. ◀

4 From Counting k-SAT to Counting Roots to Polynomials of O(1)-Degree

Reminder of Theorem 4. *Let q be a prime power and $\varepsilon > 0$ be arbitrarily small. There is an $O(q^{\varepsilon n})$-time deterministic reduction from #k-SAT instances with n variables to the problem of counting roots to a $\mathbb{F}_q$-polynomial of degree $q(k/\varepsilon)^{O(k)}$ with n variables.*

Imagining q and k as fixed constants, and ε as a tiny parameter, we obtain a $2^{O(\varepsilon n)}$ time reduction from #k-SAT on n variables to counting roots of an $\mathbb{F}_q$-polynomial on n variables of degree poly$(1/\varepsilon)$.

Proof. Let $\varepsilon > 0$ be arbitrarily small. We are given a k-CNF formula F in n variables $x_1, \ldots, x_n$, and we want to reduce it to a single low-degree polynomial. We will in fact reduce the counting problem for F to a (sub-exponential) number of calls to counting roots of a single low-degree polynomial.

First, by the Sparsification Lemma [25, 11] (the counting version of which appears in [15]), we may assume without loss of generality that the k-CNF formula F has at most $m \leq (k/\varepsilon)^{O(k)}n$ clauses, with $2^{\varepsilon n}$-time overhead.

Second, we can express F as a system of m polynomial equations in the obvious way, where each equation contains at most k variables (and therefore each equation has degree at most k). For all $i = 1, \dots, n$, add the degree-two equations $x_i \cdot (1 - x_i) = 0$ to the system (these equations simply force all solutions to be Boolean). Call the overall system of $m + n$ equations G, and note the number of solutions to G equals the number of SAT assignments to F.

Arbitrarily partition G into εn subsystems of equations $G_1, \dots, G_{\varepsilon n}$, where each subsystem has at most $(k/\varepsilon)^{O(k)}$ equations. Our next move is to write each G_j as a *single* polynomial over the finite field $\mathbb{F}_q$. More precisely, given that G_j contains the $t = (k/\varepsilon)^{O(k)}$ equations

$$p_1(x_1, \dots, x_n) = 0, \dots, p_t(x_1, \dots, x_n) = 0,$$

define the $(q-1)(k/\varepsilon)^{O(k)}$-degree polynomial

$$P_j(x_1, \dots, x_n) := 1 - \prod_{i=1}^{t} (1 - p_i(x_1, \dots, x_n)^{q-1}).$$

Note that for all $(a_1, \dots, a_n) \in \mathbb{F}_q^n$, $P_j(a_1, \dots, a_n) = 0$ if and only if $p_i(a_1, \dots, a_n) = 0$ for each i with $1 \le i \le t$. Furthermore, since each p_i has at most k variables, there are at most kt variables in P_j. So by repeatedly applying the identity $x_i^q = x_i$ over $\mathbb{F}_q$, it takes no more than $q^{O(kt)} \le q^{(k/\varepsilon)^{O(k)}}$ time to express the polynomial P_j as a sum of monomials, for all $j = 1, \dots, \varepsilon n$.

Finally, we wish to exactly count the number of solutions to the system

$$P_1(x_1, \dots, x_n) = 0, \dots, P_{\varepsilon n}(x_1, \dots, x_n) = 0 \tag{4}$$

where each P_j has $(k/\varepsilon)^{O(k)}$ variables and degree at most $q(k/\varepsilon)^{O(k)}$. Here, we reason similarly to the earlier approximate counting algorithm (namely, the reduction of Theorem 3), except instead of using small-biased sets of size polynomial in n, we simply use all $q^{\varepsilon n}$ possible linear combinations of the P_j's to exactly count.

For every $\beta \in \mathbb{F}_q^{\varepsilon n}$, suppose we count the number of zeroes to the polynomial

$$Q_\beta(x_1, \dots, x_n) := \sum_{j=1}^{\varepsilon n} \beta_j \cdot P_j(x_1, \dots, x_n)$$

and suppose we count the number of solutions to the polynomial $R_\beta := 1 - Q_\beta$. Note for all β, the degree of Q_β is at most $q(k/\varepsilon)^{O(k)}$. We want to show that a linear combination of these $O(q^{\varepsilon n})$ zero-counts will tell us the number of solutions to the original k-CNF F.

Suppose $(a_1, \dots, a_n) \in \mathbb{F}_q^n$ is a solution to the system G. Then it is also a solution to the system (4). Hence $P_j(a_1, \dots, a_n) = 0$ for all j, and therefore $Q_\beta(a_1, \dots, a_n) = 0$ for all $\beta \in \mathbb{F}_q^{\varepsilon n}$. That is, $(a_1, \dots, a_n)$ is a zero for all $q^{\varepsilon n}$ polynomials Q_β, and is never a zero for any R_β.

On the other hand, if $(a_1, \dots, a_n) \in \mathbb{F}_q^n$ is not a solution to G, then $P_j(a_1, \dots, a_n) \neq 0$ for some j. So we can think of each $Q_\beta(a_1, \dots, a_n)$ as the inner product of the vector β with a fixed non-zero vector. Therefore in this case, $Q_\beta(a_1, \dots, a_n) = 0$ for exactly $q^{\varepsilon n}/q$ polynomials Q_β, and $R_\beta(a_1, \dots, a_n) = 0$ for exactly $q^{\varepsilon n}/q$ polynomials R_β.

Combining these observations, we conclude that

$$\frac{(\text{total number of zeros to all } Q_\beta) - (\text{total number of zeros to all } R_\beta)}{q^{\varepsilon n}}$$

equals the number of solutions to G. So we can solve #k-SAT by making $O(q^{\varepsilon n})$ calls to counting solutions to a single degree-$q(k/\varepsilon)^{O(k)}$ polynomial over $\mathbb{F}_q$. (Note that by tweaking ε slightly, we can write the number of calls as $O(2^{\varepsilon n})$.) ◀

We observe that the proof of Theorem 4 also provides a subexponential-time reduction from

exact counting for a system of $O(n)$ degree-$O(1)$ equations

to

exact counting for one degree-$O(1)$ equation.

(Referring back to the proof, even if each p_i depended on all n variables but had degree only k, each polynomial P_i would have n variables and degree at most $q(k/\varepsilon)^{O(k)}$, so it would take at most $n^{q(k/\varepsilon)^{O(k)}}$ time to expand each P_i into a sum of monomials.) To compare, Theorem 3 gave a polynomial-time reduction for the respective approximation versions (but from a degree-k system to a single degree-k polynomial).

4.1 A consequence for fine-grained counting complexity

The reduction method in the proof of Theorem 4 extends nicely to results on the fine-grained counting complexity of simple polynomial-time problems. Here we demonstrate this claim on the problem of counting the number of orthogonal pairs among a set of Boolean vectors:

#Orthogonal Vectors (#OV)
Given: vectors $v_1, \ldots, v_n, w_1, \ldots, w_n \in \{0,1\}^d$
Output: The number of pairs (i, j) such that $\langle v_i, w_j \rangle = 0$.

Note that #OV is trivially solvable in $O(n^2 d)$ time, although faster algorithms are known for certain ranges of d [22, 13]. The detection problem OV (determining if there is at least one orthogonal pair) is widely studied. Finding a significantly faster algorithm for OV will already be challenging, as it is known that (for example) a $n^{1.9} \cdot 2^{o(d)}$ time algorithm for OV would contradict SETH [34]. A minor variant of OV studies the problem modulo a fixed prime p:

Orthogonal Vectors Mod p (OVp)
Given: vectors $v_1, \ldots, v_n, w_1, \ldots, w_n \in \mathbb{F}_p^d$
Decide: Are there i, j such that $\langle v_i, w_j \rangle = 0 \bmod p$?

Williams and Yu [35] showed that OVp is apparently much easier than OV for constant p: it is solvable in $O(n \cdot d^{p-1})$ time.

One can similarly define #OVp, in which the task is to count the number of i, j such that $\langle v_i, w_j \rangle = 0 \bmod p$. Recently, Dell and Lapinskas [16] show how to use the algorithm for OVp to approximately compute #OVp efficiently. In particular, they show that for any $\varepsilon > 0$, given an #OVp instance with number of solutions N, one can output a value v such that $|v - N| \leq \varepsilon N$ in $\tilde{O}(\varepsilon^{-4} n \cdot d^{p-1})$ time.

Interestingly, a minor modification of Theorem 4 shows that *exactly* computing #OVp is as hard as #OV itself:

Reminder of Theorem 5. *Let p be prime, and let $\ell \in [1, d]$ be an integer that divides d. There is an $\tilde{O}(n \cdot \ell 2^{d/\ell} \cdot p^{\ell})$-time reduction from #OV with n vectors in d dimensions to p^{ℓ} instances of #OVp, each with n vectors in $\ell 2^{d/\ell} + 1$ dimensions.*

Proof. The idea of the reduction is analogous to Theorem 4, except we need to be slightly more abstract in our construction. We start with vectors $v_1, \ldots, v_n, w_1, \ldots, w_n \in \{0,1\}^d$, and we want to compute #OV on them. Partition the *components* of all vectors into ℓ parts, where each part has d/ℓ components. For each vector v_i, let $v_{i,1}, \ldots, v_{i,\ell} \in \{0,1\}^{d/\ell}$ be its decomposition into parts; define vectors $w_{i,j}$ similarly.

For each $j = 1, \ldots, \ell$, make a $2^{d/\ell}$-bit vector $a_{i,j}$ which has a 1 in the component corresponding to the d/ℓ-bit vector $v_{i,j}$, and 0s in all other components. Also for each $j = 1, \ldots, \ell$, make a $2^{d/\ell}$-bit vector $b_{i,j}$ which has a 1 for each d/ℓ-bit vector x such that $\langle x, w_{i,j} \rangle \neq 0$, and 0s everywhere else. (This is similar to "embedding 3" of Ahle, Pagh, Razenshteyn and Silvestri [1, Lemma 3].) Taking the vectors

$$a_i := (a_{i,1}, \ldots, a_{i,\ell}) \in \{0,1\}^{\ell 2^{d/\ell}}, b_i := (b_{i,1}, \ldots, b_{i,\ell}) \in \{0,1\}^{\ell 2^{d/\ell}},$$

over all $i = 1, \ldots, n$, we have $\langle v_i, w_j \rangle = 0$ if and only if $\langle a_i, b_i \rangle = 0$. Furthermore, notice that for all $j = 1, \ldots, \ell$, $\langle a_{i,j}, b_{i,j} \rangle \in \{0,1\}$, so for all primes p and for all j, we have $\langle a_{i,j}, b_{i,j} \rangle = 0$ if and only if $\langle a_{i,j}, b_{i,j} \rangle = 0 \bmod p$.[4]

We now build $2p^\ell$ instances of #OVp as follows. For every $\beta \in \mathbb{F}_p^\ell$, construct the $2n$ vectors

$$a_i^\beta = (\beta_1 a_{i,1}, \ldots, \beta_\ell a_{i,\ell}), b_i := (b_{i,1}, \ldots, b_{i,\ell}),$$

and let N_β be the number of pairs $(a_i^\beta, b_{i'})$ which are orthogonal modulo p, for all $i, i' = 1, \ldots, n$. Also construct

$$c_i^\beta = (1, \beta_1 a_{i,1}, \ldots, \beta_\ell a_{i,\ell}), d_i := (1, b_{i,1}, \ldots, b_{i,\ell}),$$

and let M_β be the number of pairs $(c_i^\beta, d_{i'})$ which are orthogonal modulo p. Our algorithm for #OV outputs the quantity

$$\sum_{\beta \in p^\ell} (N_\beta - M_\beta)/p^\ell.$$

It is easy to see that this reduction has the desired running time and number of oracle calls. We need to show that the reduction outputs the correct number of orthogonal pairs. For an orthogonal pair v_i, w_j in the original instance, we know that $\langle a_i, b_j \rangle = 0$, and therefore $\langle a_{i,j}, b_{i,j} \rangle = 0$ for all j. So for every vector β, $\langle a_i^\beta, b_j \rangle = 0 \bmod p$ as well. That is, every orthogonal pair v_i, w_j is counted p^ℓ times in the sum $\sum_\beta (N_\beta - M_\beta)$.

For an non-orthogonal pair v_i, w_j, we know that $\langle a_{i,j}, b_{i,j} \rangle \neq 0$ for some j. In particular, recalling that all $\langle a_{i,j}, b_{i,j} \rangle$ are either 0 or 1, we have that the ℓ-dimensional vector

$$ab_{i,j} = (\langle a_{i,1}, b_{i,1} \rangle, \ldots, \langle a_{i,\ell}, b_{i,\ell} \rangle)$$

is not the all-zero vector over $\mathbb{F}_p$. Observing that

$$\langle a_i^\beta, b_j \rangle = \langle \beta, ab_{i,j} \rangle \bmod p$$

and

$$\langle c_i^\beta, d_j \rangle = 1 + \langle \beta, ab_{i,j} \rangle \bmod p,$$

it follows that there are exactly $p^{\ell-1}$ choices of β for which $\langle a_i^\beta, b_j \rangle = 0 \bmod p$, and $p^{\ell-1}$ choices of β for which $\langle c_i^\beta, d_j \rangle = 0 \bmod p$. Therefore every non-orthogonal pair has a net contribution of zero to the sum $\sum_\beta (N_\beta - M_\beta)/p^\ell$. ◀

[4] Note [1] use the fact that $\langle a_i, b_i \rangle \in \{0, 1, \ldots, \ell\}$ to give a non-trivial inapproximability result for computing the maximum inner product between two vector sets.

Setting $\ell := \lceil \varepsilon \log_p(n) \rceil$ for tiny $\varepsilon > 0$, we obtain:

Reminder of Corollary 6. *Let $\varepsilon > 0$ be sufficiently small. There is an $\tilde{O}(n^{1+\varepsilon} \cdot p^{O(c/\varepsilon)})$-time reduction from $\#OV$ with n vectors in $c \log n$ dimensions to n^{ε} instances of $\#OVp$, each with n vectors in $O(p^{c/\varepsilon} \log(n))$ dimensions.*

Therefore an algorithm for counting orthogonal-mod-2 pairs in $n^{1.9} \cdot 2^{o(d)}$ time would yield a similar algorithm for counting orthogonal pairs, refuting SETH.

Acknowledgements. I am grateful to the Simons Institute at UC Berkeley for inviting me to the workshop on "Proving and Using Pseudorandomness" in Spring 2017. There, I got a chance to think more carefully about the ideas in the approximate counting algorithm. I also thank the reviewers of SOSA for some helpful comments, and Holger Dell for telling me about his work on approximate counting; the conversation prompted me to work out the details for Theorem 5.

References

1 Thomas Dybdahl Ahle, Rasmus Pagh, Ilya P. Razenshteyn, and Francesco Silvestri. On the complexity of inner product similarity join. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 151–164. ACM, 2016. `doi:10.1145/2902251.2902285`.

2 Miklós Ajtai and Avi Wigderson. Deterministic simulation of probabilistic constant depth circuits. *Advances in Computing Research*, 5:199–222, 1989.

3 Noga Alon and Jehoshua Bruck. Explicit constructions of depth-2 majority circuits for comparison and addition. *SIAM J. Discrete Math.*, 7(1):1–8, 1994. `doi:10.1137/S0895480191218496`.

4 Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992. `doi:10.1002/rsa.3240030308`.

5 Yossi Azar, Rajeev Motwani, and Joseph Naor. Approximating probability distributions using small sample spaces. *Combinatorica*, 18(2):151–171, 1998. `doi:10.1007/PL00009813`.

6 Andrej Bogdanov and Emanuele Viola. Pseudorandom bits for polynomials. *SIAM J. Comput.*, 39(6):2464–2486, 2010. `doi:10.1137/070712109`.

7 Andrej Bogdanov and Emanuele Viola. Pseudorandom bits for polynomials. *SIAM J. Comput.*, 39(6):2464–2486, 2010. `doi:10.1137/070712109`.

8 Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *J. ACM*, 57(5):28:1–28:10, 2010. `doi:10.1145/1754399.1754401`.

9 Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, acˆ0 functions, and spectral norms. *SIAM J. Comput.*, 21(1):33–42, 1992. `doi:10.1137/0221003`.

10 Jin-yi Cai, Xi Chen, Richard J. Lipton, and Pinyan Lu. On tractable exponential sums. In Der-Tsai Lee, Danny Z. Chen, and Shi Ying, editors, *Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings*, volume 6213 of *Lecture Notes in Computer Science*, pages 148–159. Springer, 2010. `doi:10.1007/978-3-642-14553-7_16`.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006. `doi:10.1109/CCC.2006.6`.

12 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0_6.

13 Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255. SIAM, 2016. doi:10.1137/1.9781611974331.ch87.

14 Radu Curticapean. Parity separation: A scientifically proven method for permanent weight loss. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.47.

15 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.

16 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. *CoRR*, abs/1707.04609, 2017. arXiv:1707.04609.

17 Andrej Ehrenfeucht and Marek Karpinski. The computational complexity of (xor, and)-counting problems. *Technical Report TR-90-031, International Computer Science Institute, Berkeley*, 1990. URL: http://www.icsi.berkeley.edu/pubs/techreports/tr-90-033.pdf.

18 Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Velickovic. Approximations of general independent distributions. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 10–16. ACM, 1992. doi:10.1145/129712.129714.

19 Oded Goldreich. In a world of p=bpp. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 191–232. Springer, 2011. doi:10.1007/978-3-642-22670-0_20.

20 Parikshit Gopalan, Adam R. Klivans, Raghu Meka, Daniel Stefankovic, Santosh Vempala, and Eric Vigoda. An FPTAS for #knapsack and related counting problems. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 817–826. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.32.

21 Parikshit Gopalan, Raghu Meka, and Omer Reingold. DNF sparsification and a faster deterministic counting algorithm. *Computational Complexity*, 22(2):275–310, 2013. doi:10.1007/s00037-013-0068-6.

22 Ben Gum and Richard J. Lipton. Cheaper by the dozen: Batched algorithms. In Vipin Kumar and Robert L. Grossman, editors, *Proceedings of the First SIAM International Conference on Data Mining, SDM 2001, Chicago, IL, USA, April 5-7, 2001*, pages 1–11. SIAM, 2001. doi:10.1137/1.9781611972719.23.

23 Edward A. Hirsch. A fast deterministic algorithm for formulas that have many satisfying assignments. *Logic Journal of the IGPL*, 6(1):59–71, 1998. doi:10.1093/jigpal/6.1.59.

24 Christian Hoffmann. Exponential time complexity of weighted counting of independent sets. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 2010. doi:10.1007/978-3-642-17493-3_18.

25 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

26 Nathan Linial and Noam Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990. doi:10.1007/BF02128670.

27 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017. doi:10.1137/1.9781611974782.143.

28 Michael Luby and Boban Velickovic. On deterministic approximation of DNF. *Algorithmica*, 16(4/5):415–433, 1996. doi:10.1007/BF01940873.

29 Michael Luby, Boban Velickovic, and Avi Wigderson. Deterministic approximate counting of depth-2 circuits. In *Second Israel Symposium on Theory of Computing Systems, ISTCS 1993, Natanya, Israel, June 7-9, 1993, Proceedings*, pages 18–24. IEEE Computer Society, 1993. doi:10.1109/ISTCS.1993.253488.

30 Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991. doi:10.1007/BF01375474.

31 Michael O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput.*, 9(2):273–280, 1980. doi:10.1137/0209024.

32 Luca Trevisan. A note on approximate counting for k-dnf. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, volume 3122 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 2004. doi:10.1007/978-3-540-27821-4_37.

33 Emanuele Viola. The sum of D small-bias generators fools polynomials of degree D. *Computational Complexity*, 18(2):209–217, 2009. doi:10.1007/s00037-009-0273-5.

34 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

35 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1867–1877. SIAM, 2014. doi:10.1137/1.9781611973402.135.

36 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.

37 Alan R. Woods. Unsatisfiable systems of equations, over a finite field. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 202–211. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743444.

On Sampling Edges Almost Uniformly

Talya Eden[1] and Will Rosenbaum[2]

1 **Tel Aviv University, Tel Aviv, Israel**
`talyaa01@gmail.com`
2 **Tel Aviv University, Tel Aviv, Israel**
`will.rosenbaum@gmail.com`

Abstract

We consider the problem of sampling an edge almost uniformly from an unknown graph, $G = (V, E)$. Access to the graph is provided via queries of the following types: (1) uniform vertex queries, (2) degree queries, and (3) neighbor queries. We describe a new simple algorithm that returns a random edge $e \in E$ using $\tilde{O}(n/\sqrt{\varepsilon m})$ queries in expectation, such that each edge e is sampled with probability $(1 \pm \varepsilon)/m$. Here, $n = |V|$ is the number of vertices, and $m = |E|$ is the number of edges. Our algorithm is optimal in the sense that any algorithm that samples an edge from an almost-uniform distribution must perform $\Omega(n/\sqrt{m})$ queries.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Sublinear Algorithms, Graph Algorithms, Sampling Edges, Query Complexity

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.7

1 Introduction

Suppose $G = (V, E)$ is a very large graph—too large to store in our local memory. Access to G is granted in accordance with the standard bounded degree graph query model of Goldreich and Ron [5] via queries of the following types: (1) sample a uniformly random vertex (*vertex queries*), (2) query for the i^{th} neighbor of a vertex v (*neighbor queries*), and (3) query the degree of a given vertex[1] (*degree queries*).[2] The query access model readily gives access to uniformly random vertices, but what if we are interested in sampling a uniformly random *edge* from E? How many queries are necessary and sufficient?

We describe an $\tilde{O}(n/\sqrt{\varepsilon m})$ time algorithm for sampling an edge in a graph such that each edge is sampled with almost equal probability. Our main result shows that it is possible to sample edges from a distribution which has *bias at most* ε, a notion formalized in the following definition.

▶ **Definition 1.** Let Q be a fixed probability distribution on a finite set Ω. We say that a probability distribution P is *pointwise ε-close to Q* if for all $x \in \Omega$,

$$|P(x) - Q(x)| \leq \varepsilon Q(x)\,, \quad \text{or equivalently} \quad 1 - \varepsilon \leq \frac{P(x)}{Q(x)} \leq 1 + \varepsilon\,.$$

If $Q = U$, the uniform distribution on Ω, then we say that P has *bias at most* ε.

[1] We note that degree queries can be implemented by performing $O(\log n)$ neighbor queries per degree query.
[2] One may also consider the more powerful "general graph model" of Parnas and Ron [8] that additionally allows *pair queries*. Indeed, our lower bound holds in the general graph model. Interestingly, our tight upper bound does not require the additional computational power afforded by pair queries.

▶ **Theorem 2.** *Let $G = (V, E)$ be an arbitrary graph with n vertices and m edges. There exists an algorithm that given n, $0 < \varepsilon < \frac{1}{2}$ and access to vertex, degree, and neighbor queries returns an edge with probability at least $2/3$, such that each returned edge is sampled according to a distribution P that has bias at most ε. The expected query complexity and running time of the algorithm are $\tilde{O}(n/\sqrt{\varepsilon m})$.*

The strength of the approximation guarantee of the algorithm in Theorem 2 allows us to obtain the following corollary for sampling weighted edges in graphs.

▶ **Corollary 3.** *Let $G = (V, E)$ and ε be as in Theorem 2, and let $w : E \to \mathbf{R}$ be an arbitrary function. Let P be the distributions on edges induced by the algorithm of Theorem 2. Then*

$$\left| \mathop{\mathbf{E}}_{e \sim P}(w(e)) - \mathop{\mathbf{E}}_{e \sim U}(w(e)) \right| \leq \varepsilon \left| \mathop{\mathbf{E}}_{e \sim U}(w(e)) \right| .$$

In Section 5, we show that the algorithm of Theorem 2 is essentially optimal, in the sense that any algorithm which returns an edge from E almost uniformly must use $\Omega(n/\sqrt{m})$ queries. This lower bound applies in the strictly more powerful general graph query model of Parnas and Ron [8], and even if the algorithm is only required to sample edges from a distribution that is close to uniform in total variational distance.

1.1 Related work

An algorithm for sampling an edge in a graph almost uniformly was first suggested by [7]. In [7], Kaufman et al. use random edge samples in order to test if a graph is bipartite. In particular, they devise a subroutine that guarantees that all but a small fraction of the edges are each sampled with probability $\Omega(1/m)$. More recently, in [1], Eden et al. use random edge sampling as a subroutine in their algorithm for approximating the number of triangles in a graph. A similar subroutine for random edge sampling is employed in [2], where the authors use edge sampling to approximate moments of the degree distribution of a graph. In all of the works [7, 1, 2], the authors avoid the "difficult task of selecting random edges from the entire graph," [1] by instead sampling edges from a smaller subgraph.

Our algorithm improves upon and simplifies the edge sampling procedures in the works cited above. Our approximation guarantee in terms of *pointwise* distance to uniformity is strictly stronger than the guarantees of any of these papers. In particular, the previous subroutines do not return edges with two high degree endpoints in the case of highly irregular graphs. However, such edges may be of significant interest in practice. Further, the subgraph sampling strategy used in these subroutines is memory intensive—a fairly large set of vertices must be sampled (and stored), and a random edge incident to the set is sampled. In contrast, each query made by our algorithm depends only on a constant number of previous queries. Thus our algorithm can be implemented using poly-logarithmic space, and can easily be parallelized.

1.2 Overview of the algorithm

We treat each undirected edge $\{u, v\}$ as a pair of directed edges, (u, v) and (v, u). For each vertex u, let $d(u)$ denote its (undirected) degree, and let $m = \sum_{u \in V} d(u) = 2|E|$ be the number of directed edges. Consider the following two process for sampling edges:

Process 1 Choose a vertex u uniformly at random, and choose v uniformly from u's neighbors. Return (u, v).

Process 2 Choose a vertex u uniformly at random and i uniformly from $\{1, \ldots, n-1\}$. If $i \leq d(u)$, return (u, v) where v is u's i^{th} neighbor. Otherwise, fail.

In Process 1, each (directed) edge (u, v) is sampled with probability $1/(n\, d(u))$, thus biasing the sample towards edges originating from vertices with low degrees. Process 2 eliminates this bias, as each edge is sampled with probability $1/n(n-1)$. However, Process 2 only succeeds with probability $m/n(n-1)$. We can improve the success probability of Process 2 by sampling $i \in [\theta]$ for $\theta < n-1$, with the caveat that some edges incident with high-degree nodes will never be sampled. The idea of our algorithm is to choose $\theta = \sqrt{m/\varepsilon}$ and sample edges emanating from high and low degree vertices separately.

We call a vertex v *light* if $d(v) \leq \sqrt{m/\varepsilon}$, and *heavy* otherwise. Similarly a directed edge (u, v) is light (resp. heavy) if u is light (resp. heavy). We attempt to sample a light edge by using the modified Process 2 above with $\theta = \sqrt{m/\varepsilon}$, and failing if the sampled vertex u is heavy. Thus, each light edge (u, v) is sampled with probability $1/(n\sqrt{m/\varepsilon})$.

In order to sample a heavy edge we use the following procedure. We first sample a light edge (u, v) as described above. If v is heavy, we then query for a random neighbor w of v. The probability of hitting some specific heavy edge (v, w) is $\frac{d^{\mathcal{L}}(v)}{n\sqrt{m/\varepsilon}} \cdot \frac{1}{d(v)}$, where $d^{\mathcal{L}}(v)$ denotes the number of light neighbors of v. The threshold $\sqrt{m/\varepsilon}$ is set as to ensure that for every heavy vertex, at most an ε-fraction of its neighbors are heavy. It follows that $d^{\mathcal{L}}(v) \approx d(v)$, and thus each heavy edge (v, w) is chosen with probability roughly $1/(n\sqrt{m/\varepsilon})$.

Our algorithm invokes the procedures for sampling light and heavy edges, each with equal probability, sufficiently many times to ensure that an edge is returned with large constant probability. In our analysis, we show that the induced distribution on edges has bias at most ε.

1.3 Overview of the lower bound

The construction of the lower bound is similar to the lower bound construction of [4]. For an arbitrary graph G' on n' vertices with m' edges, let G be the graph obtained by adding a clique on $\Theta(\sqrt{m'})$ vertices to G'. Since the clique contains a constant fraction of G's edges, any almost-uniform edge sampler must return a clique edge with constant probability. The probability of sampling a clique vertex is $O(\sqrt{m}/n)$, so any algorithm that returns an edge according to an almost uniform distribution must perform $\Omega(n/\sqrt{m})$ queries.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices. We treat each undirected edge $\{u, v\}$ in E as pair of directed edges (u, v), (v, u) from u to v and from v to u, respectively. We denote the (undirected) degree of a vertex $v \in V$ by $d(v)$. Thus, the number of (directed) edges in G is $m = \sum_{v \in V} d(v) = 2|E|$. For $v \in V$, we denote the set of neighbors of v by $\Gamma(v)$. We assume that each v has an arbitrary but fixed order on $\Gamma(v)$ so that we may refer unambiguously to v's i^{th} neighbor for $i = 1, 2, \ldots, d(v)$.

We partition V and E into sets of light and heavy elements depending on their degrees.

▶ **Definition 4.** We say that a vertex u is a *light vertex* if $d(u) \leq \sqrt{m/\varepsilon}$ and otherwise we say that it is a *heavy vertex.* We say that an edge is a *light edge* if it originates from a light vertex. A *heavy edge* is defined analogously. Finally, we denote the sets of light and heavy vertices by $\mathcal{L}$ and $\mathcal{H}$ respectively, and the sets of light and heavy edges by $E_{\mathcal{L}}$ and $E_{\mathcal{H}}$ respectively.

Note that for a fixed *un*directed edge $\{u, v\}$, it may be the case that one of the corresponding directed edges, say (u, v), is light while the other, (v, u), is heavy. Specifically, this will occur if u is a light vertex and v is a heavy vertex.

The algorithms we consider access a graph G via queries. Our algorithm uses the following queries:

1. Vertex query: returns a uniformly random vertex $v \in V$.
2. Degree query: given a vertex $v \in V$, returns $d(v)$.
3. Neighbor query: given $v \in V$ and $i \in \mathbf{N}$, return v's i^{th} neighbor; if $i > d(v)$, this operation returns *fail*.

Our lower bounds apply additionally to a computational model that allows pair queries.

4. Pair query: given $v, w \in V$, returns *true* if $(v, w) \in E$; otherwise returns *false*.

The (expected) query cost of an algorithm $\mathcal{A}$ is the (expected) number of queries that $\mathcal{A}$ makes before terminating. We make no restrictions on the computational power of $\mathcal{A}$ except for the number of queries $\mathcal{A}$ makes to G. The query complexity of a task is the minimum query cost of any algorithm which performs the task.

We will require the following lemma, which gives sufficient conditions for a probability distribution P to have bias at most ε (recall Definition 1).

▶ **Lemma 5.** *Let P be a probability distribution over a finite set Ω which satisfies*

$$1 - \varepsilon \le \frac{P(x)}{P(y)} \le 1 + \varepsilon \quad \text{for all } x, y \in \Omega\,.$$

Then P has bias at most ε.

Proof. Suppose P satisfies the hypothesis of the lemma. Then

$$1 - \varepsilon \le \frac{P(x)}{P(y)} \le 1 + \varepsilon \implies (1 - \varepsilon)\frac{P(y)}{U(x)} \le \frac{P(x)}{U(x)} \le (1 + \varepsilon)\frac{P(y)}{U(x)}.$$

Summing the second expression over all $y \in \Omega$ gives

$$(1 - \varepsilon)\frac{1}{U(x)} \le \frac{P(x)}{U(x)} \cdot \frac{1}{U(x)} \le (1 + \varepsilon)\frac{1}{U(x)}\,.$$

The factor of $1/U(x)$ appears in the middle term because we sum over $|\Omega| = 1/U(x)$ terms. Hence P has bias at most ε. ◀

3 The Basic Algorithm

We start by presenting our main algorithm – Sample-edge-almost-uniformly– that samples a random edge in E almost uniformly with high probability. The algorithm is given query access to G and takes n, m and ε as parameters. For simplicity of presentation, we assume that m is known precisely. In Section 4, we show that this assumption is unnecessary.

We defer the statement of the lemma and proof regarding the correctness of the algorithm to the end of the section, and first present the two subroutines used in the algorithm Sample-light-edge and Sample-heavy-edge, for sampling a uniform light edge and an almost uniform heavy edge, respectively.

▶ **Lemma 6.** *The procedure* Sample-light-edge *performs a constant number of queries and succeeds with probability $|E_{\mathcal{L}}| / (n\sqrt{m/\varepsilon})$. In the case where* Sample-light-edge *succeeds, the edge returned is uniformly distributed in $E_{\mathcal{L}}$.*

Algorithm 1 Sample-edge-almost-uniformly(n, m, ε)

```
1. For $i = 1$ to $q = \frac{10n}{(1-\varepsilon)\sqrt{\varepsilon m}}$ do:
    a. With probability $1/2$ invoke Sample-light-edge$(m)$ and with probability $1/2$
       invoke Sample-heavy-edge$(m)$.
    b. If an edge $(u, v)$ was returned, then **return** $(u, v)$.
2. **Return** *fail*.
```

Algorithm 2 Sample-light-edge(m)

```
1. Sample a vertex $u \in V$ uniformly at random and query for its degree.
2. If $d(u) > \sqrt{m/\varepsilon}$ **return** *fail*.
3. Choose a number $j \in \left[\sqrt{m/\varepsilon}\right]$ uniformly at random.
4. Query for the $j^{\text{th}}$ neighbor of $u$.
5. If no vertex was returned then **return** *fail*. Otherwise, let $v$ be the returned
   vertex.
6. **Return** $(u, v)$.
```

Algorithm 3 Sample-heavy-edge(m)

```
1. Sample a vertex $u \in V$ uniformly at random and query for its degree.
2. If $d(u) > \sqrt{m/\varepsilon}$ **return** *fail*.
3. Choose a number $j \in \left[\sqrt{m/\varepsilon}\right]$ uniformly at random.
4. Query for the $j^{\text{th}}$ neighbor of $u$.
5. If no vertex was returned or if the returned vertex is light then **return** *fail*.
   Otherwise, let $v$ be the returned vertex.
6. Sample $w$ a random neighbor of $v$.
7. **Return** $(v, w)$.
```

Proof. Suppose a light vertex u is sampled in Step 1 of the procedure. Then the probability that we obtain a neighbor of u in Step 4 is $d(u)/\sqrt{m/\varepsilon}$. Hence,

$$\Pr[\text{ Success }] = \sum_{u \in \mathcal{L}} \frac{1}{n} \cdot \frac{d(u)}{\sqrt{m/\varepsilon}} = \frac{|E_{\mathcal{L}}|}{n\sqrt{m/\varepsilon}}.$$

In any invocation of the algorithm, the probability that a particular (directed) edge e is returned is $1/(n\sqrt{m/\varepsilon})$ if e is light, and 0 otherwise. Thus, each light edge is returned with equal probability. ◀

▶ **Lemma 7.** *The procedure* Sample-heavy-edge *performs a constant number of queries and succeeds with probability in* $\left[(1-\varepsilon)\frac{|E_{\mathcal{H}}|}{n\sqrt{m/\varepsilon}}, \frac{|E_{\mathcal{H}}|}{n\sqrt{m/\varepsilon}}\right]$. *In the case where* Sample-heavy-edge *succeeds, the edge returned is distributed according to a distribution P that has bias at most ε.*

Proof. Since each $v \in \mathcal{H}$ satisfies $d(v) > \sqrt{m/\varepsilon}$, we have $|\mathcal{H}| < m/\sqrt{m/\varepsilon} = \sqrt{\varepsilon m}$. For every vertex $v \in \mathcal{H}$, let $d^{\mathcal{L}}(v)$ denote the cardinality of $\Gamma(v) \cap \mathcal{L}$, the set of light neighbors of v. Similarly, let $d^{\mathcal{H}}(v)$ denote $|\Gamma(v) \cap \mathcal{H}|$. Thus, $d^{\mathcal{H}}(v) \le |\mathcal{H}| < \sqrt{\varepsilon m} < \varepsilon d(v)$.

Since $d^{\mathcal{L}}(v) + d^{\mathcal{H}}(v) = d(v)$ for every v, we have

$$d^{\mathcal{L}}(v) > (1-\varepsilon)\, d(v)\,. \tag{1}$$

Each vertex $v \in \mathcal{H}$ is chosen in Step 5 with probability $d^{\mathcal{L}}(v)/(n\sqrt{m/\varepsilon})$. Therefore, the probability that $e = (v, w)$ is chosen in Step 6 satisfies

$$\Pr[e \text{ is returned}] = \frac{d^{\mathcal{L}}(v)}{n\sqrt{m/\varepsilon}} \cdot \frac{1}{d(v)} > \frac{(1-\varepsilon)}{n\sqrt{m/\varepsilon}}\,, \tag{2}$$

where the inequality follows from Equation (1). On the other hand, $d^{\mathcal{L}}(v) \le d(v)$ implies that $\Pr[e \text{ is returned}] \le 1/\left(n\sqrt{m/\varepsilon}\right)$. Finally, we bound the success probability by

$$\Pr[\text{ Success }] = \Pr\left[\bigcup_{e \in E_{\mathcal{H}}} \{e \text{ is returned}\}\right] = \sum_{e \in E_{\mathcal{H}}} \Pr[e \text{ is returned}] > \frac{(1-\varepsilon)\,|E_{\mathcal{H}}|}{n\sqrt{m/\varepsilon}}\,.$$

The second equality holds because the events $\{e \text{ is returned}\}$ are disjoint, while the inequality holds by Equation (2). ◀

Assuming that m is known to the algorithm Sample-edge-almost-uniformly, Theorem 2 is an immediate consequence of the following lemma. In Section 4, we consider the case where m is not initially known to the algorithm and prove Theorem 2 in its full generality.

▶ **Lemma 8.** *For any ε satisfying $0 < \varepsilon < 1/2$,* Sample-edge-almost-uniformly *returns an edge with probability at least $2/3$. If an edge is returned, then it is distributed according to a distribution P that has bias at most 2ε.*

Proof. We first prove that the induced distribution on edges has bias at most 2ε. By Lemmas 6 and 7, the probability of successfully returning an edge in Step 1a satisfies

$$\begin{aligned}\Pr[\text{ Success }] &= \frac{1}{2}\Pr[\text{ Sample-light-edge succeeds }] + \frac{1}{2}\Pr[\text{ Sample-heavy-edge succeeds }]\\ &> \frac{1}{2}\cdot\frac{|E_{\mathcal{L}}|}{n\sqrt{m/\varepsilon}} + \frac{1}{2}\cdot\frac{(1-\varepsilon)\,|E_{\mathcal{H}}|}{n\sqrt{m/\varepsilon}} \ge (1-\varepsilon)\frac{m}{2n\sqrt{m/\varepsilon}}\,.\end{aligned}$$

The second inequality holds because $|E_{\mathcal{L}}| + |E_{\mathcal{H}}| = m$. Also by Lemmas 6 and 7, the probability, p_e, that a specific edge e is returned satisfies

$$\frac{1-\varepsilon}{2n\sqrt{m/\varepsilon}} \le p_e \le \frac{1}{2n\sqrt{m/\varepsilon}}.$$

Thus, the distribution on sampled edges satisfies

$$1-\varepsilon \le \frac{P(e)}{P(e')} \le \frac{1}{1-\varepsilon} \le 1+2\varepsilon\,, \quad \text{for all } e, e' \in E.$$

Therefore, P has bias at most 2ε by Lemma 5.

We now prove that the algorithm returns an edge with probability at least $2/3$. Let χ_i be the indicator variable for the event that an edge (u, v) was returned in the ith step of the for loop of the algorithm. By Lemmas 6 and 7,

$$\Pr[\chi_i = 0 \text{ for all } i] \le \left(1 - \frac{(1-\varepsilon)\sqrt{\varepsilon m}}{2n}\right)^{\frac{10n}{(1-\varepsilon)\sqrt{\varepsilon m}}} < 1/3.$$

Finally, since every invocation of Sample-light-edge and Sample-heavy-edge takes a constant number of queries, the query complexity and running time of the algorithm are $O(n/\sqrt{\varepsilon m})$. ◀

4 Sampling Edges with Unknown m

In the previous section, we assumed that the value of m (or more specifically, $\sqrt{m/\varepsilon}$) was known to the algorithm. In this section, we argue that such an assumption is unnecessary. In particular, it is sufficient to have any estimate $\widehat{m} \in [m, cm]$ for a fixed constant c. Such an estimate can be obtained with high probability using $\tilde{O}(n/\sqrt{m})$ expected queries by employing an algorithm of Goldreich and Ron [6].[3]

▶ **Theorem 9** (Goldreich & Ron [6]). *Let $G = (V, E)$ be a graph with n vertices and m edges. There exists an algorithm that uses $\tilde{O}(n/\sqrt{m})$ vertex, degree, and neighbor queries in expectation and outputs an estimate $\widehat{m}$ of m that with probability at least $2/3$ satisfies $m \leq \widehat{m} \leq 2m$.*

Analogues of Lemmas 6 and 7 hold for any estimate $\widehat{m}$ of m, with the threshold between light and heavy vertices redefined to be $\sqrt{\widehat{m}/\varepsilon}$. In particular, if $\widehat{m}$ is an overestimate—i.e., $\widehat{m} > m$—then the approximation guarantees of both lemmas are still satisfied. However, an overestimate results in a smaller success probability (by a factor of $\sqrt{m/\widehat{m}}$). It is straightforward to verify that so long as $m \leq \widehat{m} \leq 2m$, the conclusion of Lemma 8 still holds (for complete details please see the full version of the paper [3]). Using Lemma 8 and Theorem 9, we can prove Theorem 2 in its full generality.

Proof of Theorem 2. Let $\widehat{m}$ be an estimate of m. We call $\widehat{m}$ a *good* estimate if it satisfies $m \leq \widehat{m} \leq 2m$. By repeating the algorithm of Goldreich & Ron (Theorem 9) $O(\log(n/\varepsilon))$ times, then taking $\widehat{m}$ to be the median value reported, a straightforward application of Chernoff bounds guarantees that $\widehat{m}$ is good with probability at least $1 - \varepsilon/2n^2$. If $\widehat{m}$ is good, by Lemma 8, calling Sample-edge-almost-uniformly$(n, \widehat{m}, \varepsilon/4)$ will successfully return an edge e with probability at least $2/3$, and the returned edge is distributed according to a distribution P which has bias at most $\varepsilon/2$.

Let Q be the distribution of returned edges. If $\widehat{m}$ is not good, we have no guarantee of the success probability of returning an edge, nor of the distribution from which the edge is drawn. However, since $\widehat{m}$ is bad with probability at most $\varepsilon/2n^2$, for each e we can bound

$$|Q(e) - U(e)| \leq |Q(e) - P(e)| + |P(e) - U(e)| \leq \frac{\varepsilon}{2n^2} + \frac{\varepsilon}{2m} \leq \frac{\varepsilon}{m} = \varepsilon U(e).$$

Thus Q has bias at most ε.

We now turn to analyze the expected query cost of the algorithm. By Theorem 9, the expected query cost of Goldreich and Ron's algorithm is $\tilde{O}(n/\sqrt{m})$. By Lemmas 6 and 7, the procedures Sample-light-edge and Sample-heavy-edge each perform a constant number of queries per invocation. Hence, the query cost of the algorithm is $O(q) = \tilde{O}\left(n/\sqrt{\varepsilon\widehat{m}}\right)$. If $\widehat{m}$ is good then q is at most $\tilde{O}\left(n/\sqrt{\varepsilon m}\right)$, and otherwise it is at most $O(n)$. Since m is good with probability at least $1 - \varepsilon/2n^2$, it follows that the expected query cost is $\tilde{O}(n/\sqrt{\varepsilon m})$. ◀

5 A Lower Bound

In this section we prove a lower bound on the number of queries necessary to sample an edge from an almost-uniform distribution over E. Specifically we show that any algorithm

[3] An earlier result of Feige [4] would also suffice, but we found the result of Goldreich and Ron simpler to apply.

$\mathcal{A}$ that samples an edge almost uniformly must perform $\Omega\left(n/\sqrt{m}\right)$ queries, even if $\mathcal{A}$ is given m. Thus, the algorithm we present is asymptotically optimal (up to poly-logarithmic factors). Further our lower bound applies to (1) strictly weaker approximation to the uniform distribution (by total variational distance), and (2) to algorithms which are additionally allowed "pair queries" at unit cost.

We first recall the definition of the total variational distance between two distributions.

▶ **Definition 10.** Let P and Q be probability distributions over a finite set Ω. We denote the *total variational distance* or *statistical distance* between P and Q by

$$\mathrm{dist}_{\mathrm{TV}}(P,Q) = \frac{1}{2}\left\|P-Q\right\|_1 = \frac{1}{2}\sum_{x\in\Omega}\left|P(x)-Q(x)\right|.$$

Observe that if P and Q are pointwise ε-close, then $\mathrm{dist}_{\mathrm{TV}}(P,Q) \le \varepsilon$, but the converse is not true in general.

▶ **Theorem 11.** *Let $\varepsilon < 1/2$ be fixed and suppose $\mathcal{A}$ is an algorithm that performs $q = q(n,m)$ vertex, degree, neighbor, or pair queries and with probability at least $2/3$ returns an edge $e \in E$ sampled according to a distribution P over E. If for all $G = (V,E)$, P satisfies $\mathrm{dist}_{\mathrm{TV}}(P,U) < \varepsilon$, then $q = \Omega(n/\sqrt{m})$.*

Proof. The result is trivial if $m = \Omega(n^2)$, so we assume that $m = o(n^2)$. Suppose there exists an algorithm $\mathcal{A}$ that performs t queries and with probability at least $2/3$ returns an edge sampled from a distribution P satisfying $\mathrm{dist}_{\mathrm{TV}}(P,U) \le \varepsilon$. Let G' be an arbitrary graph, and let n' and m' denote the number of vertices and edges, respectively, in G'. Let K be a clique on $k = \sqrt{2m'}$ nodes. Let $G = G' \cup K$ be the disjoint union of G' and K, and let V_K and E_K denote the vertices and edges of K in G. Thus G has $n = n' + k$ nodes, $m > 2m'$ edges, and E_K contains at least $m/2$ edges. The remainder of the proof formalizes the intuition that since $\mathcal{A}$ makes relatively few queries, it is unlikely to sample vertices in V_K. Thus $\mathcal{A}$ must sample edges from E_K with probability significantly less than $1/2$.

Assume that the vertices are assigned distinct labels from $[n]$ uniformly at random, independently of any decisions made by the algorithm $\mathcal{A}$. Let $q_1,\ldots,q_t$ denote the set of queries that the algorithm performs, and let $a_1,\ldots,a_t$ denote the corresponding answers. We say that a query-answer pair (q_i,a_i) is a *witness pair* if (1) q_i is a degree query of $v \in V_K$, or (2) q_i is a neighbor query for some $v \in V_k$, or (3) q_i is a pair query for some $(v,w) \in E_K$. For $i \in [t]$ let NW_i denote that event that $(q_1,a_1),\ldots,(q_i,a_i)$ are not witness pairs, and let $\mathsf{NW} = \mathsf{NW}_t$. Let A_K be the event that $\mathcal{A}$ returns some edge $e \in E_K$. Since $\mathrm{dist}_{\mathrm{TV}}(P,U) < \varepsilon$, we must have $|\Pr_P[A_K] - \Pr_U[A_K]| \le \varepsilon$. Since $|E_K| \ge m/2$, we have $\Pr_U[A_K] \ge 1/2$, hence

$$\Pr_P[A_k] \ge \frac{1}{2} - \varepsilon. \tag{3}$$

The law of total probability gives

$$\Pr_P[A_K] = \Pr_P[\mathsf{NW}]\cdot\Pr_P[A_K \mid \mathsf{NW}] + \Pr_P[\mathsf{NW}^c]\cdot\Pr_P[A_K \mid \mathsf{NW}^c], \tag{4}$$

where NW^c denotes the complement of the event NW.

Claim. $\Pr_P[A_K \mid \mathsf{NW}] = o(1)$.

Proof of Claim. Suppose the event NW occurs, i.e., $\mathcal{A}$ does not observe a witness pair after t queries. Thus, if $\mathcal{A}$ returns an edge $e = (u,v) \in E_K$, it cannot have made queries involving u or v. We can therefore bound

$$\Pr_P[A_K \mid \mathsf{NW}] \le |E_K|\frac{1}{(n-2t)^2} \le \frac{m}{2(n-2t)^2} \le \frac{2m}{n^2}.$$

The first inequality holds because the identities of any $u, v \in V_K$ are uniformly distributed among the (at least) $n - 2t$ vertices not queried by $\mathcal{A}$. The second inequality holds assuming $t < n/4$. The claim follows from the assumption that $m = o(n^2)$.

Combining the result of the claim with equations (3) and (4) gives

$$\frac{1}{2} - \varepsilon \leq \Pr_P[A_k] = \Pr_P[\mathsf{NW}] \cdot \Pr_P[A_K \mid \mathsf{NW}] + \Pr_P[\mathsf{NW}^c] \cdot \Pr_P[A_K \mid \mathsf{NW}^c] \leq \Pr_P[\mathsf{NW}^c] + o(1)\,. \quad (5)$$

We bound $\Pr_P[\mathsf{NW}^c]$ by

$$\begin{aligned}\Pr[\mathsf{NW}^c] &= \Pr\left[\bigcup_{i \leq t} \{(q_i, a_i) \text{ is the first witness pair}\}\right] \\ &= \sum_{i \leq t} \Pr[(q_i, a_i) \text{ is a witness pair} \mid \mathsf{NW}_{i-1}] \\ &\leq \sum_{i \leq t} \frac{2k}{n - 2i} \leq \frac{4kt}{n} \leq t\frac{4\sqrt{2m}}{n}.\end{aligned}$$

Combining this bound with (5) and solving for t gives $\frac{2}{3}\left(\frac{1}{2} - \varepsilon - o(1)\right)\frac{n}{4\sqrt{2m}} < t$. The factor of $2/3$ is because $\mathcal{A}$ returns an edge with probability at least $2/3$. Thus, $t = \Omega(n/\sqrt{m})$, as desired. ◀

Acknowledgments. We are very grateful to Dana Ron and Oded Goldreich for valuable discussions and comments.

References

1 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM J. Comput.*, 46(5):1603–1646, 2017. `doi:10.1137/15M1054389`.

2 Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The degeneracy connection. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.ICALP.2017.7`.

3 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. *CoRR*, abs/1706.09748, 2017. `arXiv:1706.09748`.

4 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35(4):964–984, 2006. `doi:10.1137/S0097539704447304`.

5 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 406–415. ACM, 1997. `doi:10.1145/258533.258627`.

6 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008. `doi:10.1002/rsa.20203`.

7 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM J. Comput.*, 33(6):1441–1483, 2004. `doi:10.1137/S0097539703436424`.

8 Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, 2002. `doi:10.1002/rsa.10013`.

A Simple PTAS for the Dual Bin Packing Problem and Advice Complexity of Its Online Version[*]

Allan Borodin[1], Denis Pankratov[2], and Amirali Salehi-Abari[3]

1 **University of Toronto, Toronto, Canada**
bor@cs.toronto.edu
2 **University of Toronto, Toronto, Canada**
denisp@cs.toronto.edu
3 **Faculty of Business and IT, University of Ontario Institute of Technology, Oshawa, Canada**
abari@uoit.ca

Abstract

Recently, Renault (2016) studied the dual bin packing problem in the per-request advice model of online algorithms. He showed that given $O(1/\epsilon)$ advice bits for each input item allows approximating the dual bin packing problem online to within a factor of $1+\epsilon$. Renault asked about the advice complexity of dual bin packing in the tape-advice model of online algorithms. We make progress on this question. Let s be the maximum bit size of an input item weight. We present a conceptually simple online algorithm that with total advice $O\left(\frac{s+\log n}{\epsilon^2}\right)$ approximates the dual bin packing to within a $1+\epsilon$ factor. To this end, we describe and analyze a simple offline PTAS for the dual bin packing problem. Although a PTAS for a more general problem was known prior to our work (Kellerer 1999, Chekuri and Khanna 2006), our PTAS is arguably simpler to state and analyze. As a result, we could easily adapt our PTAS to obtain the advice-complexity result.

We also consider whether the dependence on s is necessary in our algorithm. We show that if s is unrestricted then for small enough $\epsilon > 0$ obtaining a $1+\epsilon$ approximation to the dual bin packing requires $\Omega_\epsilon(n)$ bits of advice. To establish this lower bound we analyze an online reduction that preserves the advice complexity and approximation ratio from the binary separation problem due to Boyar et al. (2016). We define two natural advice complexity classes that capture the distinction similar to the Turing machine world distinction between pseudo polynomial time algorithms and polynomial time algorithms. Our results on the dual bin packing problem imply the separation of the two classes in the advice complexity world.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases dual bin packing, PTAS, tape-advice complexity

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.8

1 Introduction

Given a sequence of items of weights $w_1, \ldots, w_n$ and m bins of unit capacity, the dual bin packing problem asks for the maximum number of items that can be packed into the bins without exceeding the capacity of any bin.[1] The search version of this problem is to find a good packing. In the online version of this problem, the items are presented one at a time in

[*] Research is supported by NSERC.
[1] This terminology is somewhat unfortunate, because the dual bin packing problem is not the dual to the natural integer programming formulation of the bin packing problem. For some early results on the latter see [1, 10].

some adversarial order and the algorithm needs to make an irrevocable decision into which (if any) bin to pack the current item. The dual bin packing problem has a substantial history in both the offline and online settings starting with Coffman et al. [14]. The performance of the online algorithm is measured by its competitive ratio; that is, the worst-case ratio between the value of an offline optimal solution and the value of the solution obtained by the algorithm. It is known that the online dual packing problem does not admit a constant competitive ratio even for randomized algorithms [8, 11]. The assumption that the online algorithm does not see the future at all is quite restrictive and in many cases impractical. It is often the case that some information about the input sequence is known in advance, e.g., its length, the largest weight of an item, etc. An information-theoretic way of capturing this side knowledge is given by the *tape-advice model* [3]. In this model, an all powerful oracle that sees the entire input sequence creates a short advice string. The algorithm uses the advice string in processing the online nodes. The main object of interest here is the tradeoff between the size of advice and the competitive ratio of an online algorithm. Often, a short advice string results in a dramatic improvement of the best competitive ratio that is achievable by an online algorithm. Of course, a short advice string can be computationally difficult to obtain since the oracle is allowed unlimited power.

A related advice model is the *per-request advice model* [13]. In this model, prior to seeing the ith input item, the algorithm receives the ith advice string. Unlike the tape-advice model, the overall length of advice is always lower bounded by n in this model. Both of these advice models have recently received considerable attention in the research community (see Boyer et al [5] for an extensive survey on this topic). Recently, Renault [16] studied the dual bin packing problem in the per-request advice model. He designed an algorithm that with 1 bit of advice per request achieves a 3/2 competitive ratio. He also showed that with $O(1/\epsilon)$ bits of advice per request it is possible to achieve a $1+\epsilon$ competitive ratio.[2] In [16] Renault explicitly asked, as an open problem, to analyze the advice complexity of the dual bin packing problem in the tape advice model. In this paper, we make progress on the advice complexity needed for achieving a $(1+\epsilon)$ competitive ratio for the online dual bin packing problem. Specifically, let s be the maximum bit size of a weight of an input item. In particular, the overall input size is $O(ns)$ bits. We present an online algorithm that with $O(\frac{s+\log n}{\epsilon^2})$ bits of advice achieves a $(1+\epsilon)$ competitive ratio for the dual bin packing problem. Note that it is trivial to achieve optimality with $n \log_2 m$ advice bits by specifying for each input item into which bin it should be placed. When stated in the tape advice model, Renault's bound for a $(1+\epsilon)$ competitive ratio is $\Theta(n/\epsilon)$. Our advice bound for achieving a $(1+\epsilon)$ approximation is exponentially smaller for the regime of constant ϵ and $s = O(\log n)$. When the n item weights have $s = n$ bits of precision, we show that the dependence on s is necessary by exhibiting an $\Omega_\epsilon(n)$ lower bound on the advice necessary to achieve a $(1+\epsilon)$ approximation.

Our main result heavily relies on a simple polynomial time approximation scheme (PTAS) for the dual bin packing problem, which constitutes the technical core of this paper. Dual bin packing is a special case of the multiple knapsack problem (MKP). In the MKP, each of the n items is described by its weight (number in $(0,1]$) and its value (an integer). There are m knapsacks each with their own capacity. The goal is to pack a subset of items such that

[2] Renault states the approximation as $1/(1-\epsilon)$ whereas we will use $(1+\epsilon)$ which is justified since $1/(1-\epsilon) \leq 1+\epsilon$ for all $\epsilon \leq 2/3$. Also without loss of generality, we will sometimes say that the approximation is $1+\Theta(\epsilon)$ since our advice bounds are asymptotic and we can replace ϵ by ϵ/c for some suitable c.

all items fit into the knapsacks without violating weight constraints and the total value of packed items is as large as possible. In the uniform MKP, capacities of the bins are equal, and, are taken to be 1 without loss of generality. Thus, the dual bin packing problem can be seen as the uniform MKP with all values being 1. It is known [9] that the dual bin packing, and consequently the MKP, is strongly NP-hard even for $m = 2$, which effectively rules out an FPTAS for these problems. This is in contrast to the standard knapsack problem and the makespan problem for a fixed number of machines where FPTAS are possible. Significant progress in the study of the MKP was made by Kellerer [15] who showed that the uniform MKP admits a PTAS. Subsequently, Chandra and Khanna [9] gave a PTAS for the general MKP. Clearly, these results also give PTAS algorithms for the dual bin packing problem. However, the PTAS algorithms provided by Kellerer, and Chekuri and Khanna, are relatively complicated algorithms with a technically detailed analysis of correctness. Our goal is to provide a simple online advice algorithm for the dual bin packing problem based on a simpler PTAS for the dual bin packing problem . Thus, as a first step, we provide a simpler PTAS and analysis for the case of the dual bin packing problem. In the second step, we use the simplified PTAS to derive our result for the tape advice-complexity of the online dual bin packing. One of the key steps in our PTAS is a dynamic programming algorithm for the dual bin packing problem with few distinct weights instead of the IP solver as in Kellerer's PTAS or the LP solver as in Chekuri and Khanna. This dynamic programming algorithm is essentially the same as the one used in the solution of the makespan problem with a bounded number of different processing times. Our PTAS and its analysis are self-contained and easy to follow. Our work highlights one of the important aspects of simple algorithms, namely, they are usually easier to modify and adapt to other problems and situations. In particular, we are able to easily adapt our simple PTAS to the setting of online tape-advice algorithms.

2 Preliminaries

The dual bin packing instance is specified by a sequence of n item weights $w_1, w_2, \ldots, w_n$ and $m \in \mathbb{N}$ bins, where $w_i \in (0, 1]$. The goal is to pack a largest subset of items into m bins such that for each bin the total weight of items placed in that bin is at most 1. The problem can be specified as an integer program as follows (notation $[n]$ stands for $\{1, \ldots, n\}$):

$$\begin{aligned}
\text{max.} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} & \\
\text{subj. to} \quad & \sum_{i=1}^{n} x_{ij} w_i \le 1 & \text{for all } j \in [m] \\
& \sum_{j=1}^{m} x_{ij} \le 1 & \text{for all } i \in [n] \\
& x_{ij} \in \{0, 1\} & \text{for all } i \in [n], j \in [m]
\end{aligned}$$

The online First Fit algorithm FF constructs a solution by processing items in the given order $w_1, w_2, \ldots, w_n$ and placing a given item into the first bin into which it fits. First Fit Increasing algorithm FFI first orders the items by increasing weight. Let $\sigma : [n] \to [n]$ be the corresponding permutation. Then FFI runs FF on the items in the order given by σ, i.e., $w_{\sigma(1)} \le w_{\sigma(2)} \le \cdots \le w_{\sigma(n)}$. It is easy to see that FF has an unbounded approximation (i.e., competitive) ratio whereas Coffman et al [14] show that FFI has a 4/3 approximation ratio for the dual packing problem. Note that the weights only enter the above integer program as constraints and are not part of the objective function. Thus, it is easy to see that the items in an optimal solution are, without loss of generality, a prefix of $w_{\sigma(1)}, w_{\sigma(2)}, \ldots, w_{\sigma(n)}$ packed into appropriate bins.

Throughout this paper we shall always write n to mean the number of input items, m the number of bins, and s the maximum bit size of an input item (s can be thought of as the "word size" of a computer, on which the given input sequence should be processed).

An online algorithm ALG is said to achieve a *competitive ratio* c for a maximization problem if there exists a constant α such that for all input sequences I we have $\mathrm{OPT}(I) \leq c\mathrm{ALG}(I) + \alpha$, where $\mathrm{ALG}(I)$ is the value of the objective that the algorithm achieves on I and $\mathrm{OPT}(I)$ is the value achieved by an offline optimal solution. If $\alpha \leq 0$, we say that ALG achieves a *strict* competitive ratio c.

3 A Simple PTAS for the Dual Bin Packing Problem

Fix $\epsilon > 0$. In what follows, for simplicity we shall assume that ϵ is "nice", i.e., $1/\epsilon$ is an integer, $m\epsilon$ is an integer, etc. We note that an arbitrary small nice ϵ can always be found. Let $S = \{i \mid w_i \leq \epsilon\}$ be the set of small input items, and let $L = \{i \mid w_i > \epsilon\}$ be the set of large input items. The goal is to pack as many items from $S \cup L$ into m bins as possible. Our first observation is that if the FFI algorithm fills m bins (i.e., does not allow any more items to be packed) using only small items then it already achieves a $1 + \epsilon$ approximation.

▶ **Lemma 1.** *Suppose that when the* FFI *algorithm terminates, it has filled all bins with items of weight at most ϵ. Then* FFI *achieves $1 + \epsilon$ approximation ratio on this instance.*

Proof. If FFI packs all items then it clearly finds an optimal solution. Suppose that FFI rejects some items. Let w be the smallest weight of a rejected item. Thus the total remaining free space among all m bins is $< wm$ in the FFI packing. Thus, OPT can pack at most $m - 1$ more items, since it can only add items of weight $\geq w$. Let N be the number of items packed by FFI. Then we have

$$\frac{\mathrm{OPT}}{\mathrm{FFI}} < \frac{N+m}{N} \leq \frac{m/\epsilon + m}{m/\epsilon} = 1 + \epsilon,$$

where the second inequality follows from $N \geq m/\epsilon$, since the FFI packing uses only items of weight $\leq \epsilon$. ◀

Thus, the whole difficulty in designing a PTAS for this problem lies in the handling of large items. If FFI terminates before packing all of S, then the condition of Lemma 1 holds and hence from now on, we consider the case when FFI packs all of S. In this case an optimal solution is to pack all of S together with some subset of smallest items from L. The strategy for our algorithm is to pack a largest subset F of L that still leaves enough room to pack all of S. This means that $w(F) \leq m - w(S)$, but we also want to pack all of S efficiently. This can be guaranteed by leaving slightly more room while packing F. Namely, $w(F) \leq m(1 - \epsilon) - w(S)$ guarantees that all of S can be packed efficiently after packing F.

▶ **Lemma 2** (Kellerer [15]). *Suppose that we have a packing of $F \subseteq L$ such that $w(F) \leq m(1 - \epsilon) - w(S)$. Then running* FFI *with the packing of F as a starting point results in packing all of S.*

Proof. Initially, we have $w(F) \leq m(1 - \epsilon) - w(S) \leq m(1 - \epsilon)$. Thus, by the pigeonhole principle there is a bin with $\geq \epsilon$ free space. Thus, we can pack the first item s_1 from S. Now, we have $w(F) \leq m(1-\epsilon) - w(S) \leq m(1-\epsilon) - w(s_1)$, i.e., $w(F) + w(s_1) \leq m(1-\epsilon)$. Again, by the pigeonhole principle there is a bin with $\geq \epsilon$ free space, so we can pack the second item from S, and so on. ◀

▶ Remark. Note that the argument in the above lemma does not use the *increasing* property of FFI. Therefore, even FF can be used to complete the partial packing F with all of S.

The next lemma shows that the extra "breathing room" that we leave to guarantee an efficient packing of S does not hurt the approximation ratio.

▶ **Lemma 3.** *Let F be the largest subset of L that can be packed into m bins with total weight $\leq m(1-\epsilon) - w(S)$. Then*

$$\frac{\text{OPT}}{|F| + |S|} \leq 1 + 3\epsilon.$$

Proof. If $F = L$ then we are done. Otherwise, let $w > \epsilon$ be the smallest weight of an item from $L \setminus F$. Then $|S| \geq w(S)/\epsilon \geq w(S)/w$ and $|F| \geq \frac{m(1-\epsilon)-w(S)}{w}$. Thus, $|F| + |S| \geq \frac{m(1-\epsilon)}{w}$. The total free space after packing $F \cup S$ is $\leq \epsilon m$. Thus, OPT can pack at most $\frac{\epsilon m}{w}$ more items than $|F| + |S|$. Combining all of the above, we have

$$\frac{\text{OPT}}{|F| + |S|} \leq \frac{|F| + |S| + \epsilon m/w}{|F| + |S|} \leq \frac{m(1-\epsilon)/w + \epsilon m/w}{m(1-\epsilon)/w} = \frac{1}{1-\epsilon} \leq 1 + 3\epsilon,$$

where the last inequality holds for small ϵ; i.e., $\epsilon \leq 2/3$. ◀

We shall refer to the problem of finding F as in the above lemma as the LFP ("the large F problem").

▶ Remark. Suppose that F is an approximation to the LFP with an additive ϵm term, i.e. $|F| \geq \text{OPT}_{LFP} - \epsilon m$. Then an argument similar to the one used in the above lemma shows that F together with S still gives $1 + \Theta(\epsilon)$ approximation to the original dual bin packing problem. Thus, it suffices to find a good enough F.

Before we show how to find a good approximation to the LFP, we show how to solve the dual bin packing optimally in polynomial time when the number of distinct weights of the input items is fixed. As previously stated, this follows from the known PTAS for the makespan problem. (See section 10.2 of the Vazirani text [17].)

▶ **Lemma 4.** *We can solve the dual bin packing problem optimally in time $O(n^{2k}m)$ where k is the number of distinct weights of the input items.*

Proof. The algorithm is a simple dynamic programming. Let $w_1, \ldots, w_k$ be the distinct weights appearing in the input. The entire input sequence can be described by a k-tuple $(n_1, \ldots, n_k)$, where n_i is the number of items of weight w_i and $n = \sum_i n_i$. Note that the number of different possible k-tuples with n items is $O(n^k)$. Let $\mathcal{K}$ be the set of distinct k-tuples such that each of its element fits entirely in a single bin, i.e., $(\ell_1, \ldots, \ell_k)$ such that $\sum_i \ell_i w_i \leq 1$. The dynamic programming table D is going to be indexed by the number of available bins m' and a possible k-tuple $(\ell_1, \ldots, \ell_k)$ such that $0 \leq \ell_i \leq n_i$. The value $D[(\ell_1, \ldots, \ell_k), m']$ is going to indicate the maximum number of items that can be packed from the input sequence described by the state $(\ell_1, \ldots, \ell_k)$ in m' bins. Let $\mathcal{L}(\ell_1, \ldots, \ell_k) = \{(\ell'_1, \ldots, \ell'_k) \mid \forall i\ 0 \leq \ell'_i \leq \ell_i\}$. An optimal solution to the subproblem indexed by $(\ell_1, \ldots, \ell_k)$ and m' consists of a packing of some element from $\mathcal{L}$ into a single bin, and packing the remaining input items into $m' - 1$ bins:

$$D[(\ell_1, \ldots, \ell_k), m'] = \max_{(\ell'_1, \ldots, \ell'_k) \in \mathcal{K} \cap \mathcal{L}(\ell_1, \ldots, \ell_k)} \sum_i \ell'_i + D[(\ell_1 - \ell'_1, \ldots, \ell_k - \ell'_k), m' - 1].$$

Algorithm 1 Our PTAS for the dual bin packing.

```
procedure DUAL BIN PACKING PTAS($w_1 \leq \cdots \leq w_n, m, \epsilon$)
    Let $S = \{i \mid w_i \leq \epsilon\}$
    if FFI($S, m$) packs $< |S|$ items then return FFI($S, m$)
    Let $L' = \{|S| + 1, \ldots, |S| + \ell\}$ be the indices of the largest subset of $L$ such that $w(L') \leq m(1 - \epsilon) - w(S)$
    Let $k = \ell/(m\epsilon)$
    Let $\widetilde{w}$ denote new weights where items with indices $\{|S| + (i-1)m\epsilon + 1, \ldots, |S| + im\epsilon\}$ all receive weight $w_{|S|+im\epsilon}$
    Use the algorithm of Lemma 4 to obtain a packing of items $F'$ with modified weights $\widetilde{w}$
    Regard $F'$ as a packing with the original weights
    Run $FF$ on $S$ with the packing of $F'$ as a starting point
return the resulting packing
```

The base case is given by the states where either $m' = 0$ or $\sum_i \ell_i = 0$, in which case we cannot pack any items. The overall runtime of this algorithm is $O(n^{2k}m)$ since the dynamic programming table has $O(n^k m)$ entries and each entry can be computed in time $O(n^k)$ with appropriate preprocessing of the input data. As usual, this dynamic program can be easily modified to return the actual packing rather than the number of packed items. ◀

Let L' be the subset of the smallest items from L such that $|L'|$ is as large as possible subject to $w(L') \leq m(1 - \epsilon) - w(S)$. We would like to find F by running the dynamic programming algorithm on L'. Unfortunately, L' can have too many distinct inputs. The idea is to group items of L' into few groups depending only on ϵ, reassign all weights of elements within a single group to the weight of the largest element in that group, and run the dynamic programming algorithm on the new problem instance. Then, we will need to argue that the resulting solution is an additive ϵm approximation to the LFP.

Let $\ell = |L'|$. We can assume $\ell > m$ otherwise there is a trivial way to pack ℓ items into m bins. Assume for simplicity that $m\epsilon$ is an integer and that $k = \ell/(m\epsilon)$ is also an integer. Then, we split L' into k groups of $m\epsilon$ elements each. Let $w_{j_1} \leq w_{j_2} \leq \cdots \leq w_{j_\ell}$ be the weights of elements in L'. Define L_i to be the ith group consisting of items of weights $w_{j_{1+(i-1)m\epsilon}}, \ldots, w_{j_{im\epsilon}}$. Reassign the weights of elements in L_i to be $w_{j_{im\epsilon}}$. Let $\widetilde{w}$ denote the modified weights. Thus, we get an instance with k distinct weights, where $k = \ell/(m\epsilon)$. Note that $\ell \leq m/\epsilon$ since we are dealing with large items $\geq \epsilon$. We conclude that $k \leq 1/\epsilon^2$. Thus, we can solve this instance in time $O(n^{2/\epsilon^2}m)$ by Lemma 4. Let F' denote this solution. Let F denote an optimal solution to LFP with the original weights. Then, we have the following.

► **Lemma 5.** *F' is feasible with respect to weights w and $|F'| \geq |F| - \epsilon m$.*

Proof. Since F' is feasible with weights $\widetilde{w}$ and $\widetilde{w} \geq w$, we immediately conclude that F' is feasible with respect to w. Rather than directly showing $|F'| \geq |F| - \epsilon m$, we show how to construct a set F'' from F such that $|F''| \geq |F| - \epsilon m$ and F'' is feasible with respect to $\widetilde{w}$. This will prove the lemma, since F' is a maximum cardinality set that satisfies the feasibility constraints (i.e., $|F'| \geq |F''|$). To construct such F'', we can simply drop all items from $F \cap L_1$ and replace all items from $F \cap L_i$ by arbitrary items from L_{i-1} for $i \geq 2$. Note that $|F''| = |F \setminus L_1| \geq |F| - \epsilon m$. Moreover, F'' is feasible with respect to $\widetilde{w}$ since we are always replacing large weight items by smaller weight items. ◀

This completes the argument that approximately solving the LFP using the reassigned weights and the dynamic programming followed by FFI on small items gives a $1 + \Theta(\epsilon)$ approximation. The running time of the dynamic programming is $O(n^{2/\epsilon^2} m)$. One can run FFI in $O(n \log n + nm)$ time. The overall running time of our PTAS algorithm is $O(n^{2/\epsilon^2} m + n \log n)$, which is clearly polynomial when ϵ is fixed. Algorithm 1 describes this PTAS.

Summarizing, in this section we proved the following theorem.

▶ **Theorem 6.** *Algorithm 1 is a PTAS for the dual bin packing problem.*

4 Advice Complexity of the Online Dual Bin Packing Problem for Bounded Bit Size of Input Items

In this section, we consider the online version of the dual bin packing problem in the tape-advice model. Let s be the maximum bit-size of an input item weight. Then the input bit-length is $O(sn)$. Based on the PTAS in Algorithm 1, we develop an online algorithm that achieves $1+\epsilon$ approximation to the dual bin packing problem with $O\left(\frac{s+\log n}{\epsilon^2}\right)$ bits of advice. Before we prove the main result of this section, we need to modify Lemma 1 to work in the online setting. Recall that Lemma 1 detects when FFI is already successful enough that we don't need to do any extra work to obtain a $1 + \epsilon$ approximation. An online algorithm does not have the ability to sort the input items, thus we would like to obtain a version of Lemma 1 that detects when FF obtains a $1 + \epsilon$ approximation. The restricted subsequence first fit (RSFF) algorithm given by Renault [16] is what we need. Let $W = w_1, \ldots, w_n$ be the sequence of weights given to the online algorithm. For a value η we define W_η to be the subsequence $(w_i \mid w_i \le \eta)$. The RSFF algorithm finds the largest value of η such that FF packs all items in W_η and then returns $\mathrm{FF}(W_\eta)$. Without loss of generality, we may assume that η is one of the w_i.

▶ **Lemma 7** (Implicit in Renault [16]). *If* RSFF *identifies an η such that $\eta \le \epsilon$ then* RSFF *achieves a $1 + \epsilon$ approximation ratio.*

By replacing FFI with RSFF in the first step of Algorithm 1, we obtain the main result of this section.

▶ **Theorem 8.** *There is an online algorithm achieving a $1 + \Theta(\epsilon)$ strict competitive ratio for the dual bin packing problem with $O\left(\frac{s+\log n}{\epsilon^2}\right)$ bits of advice, where s is the maximum bit-size of an input item.*

Proof. The advice is obtained by slightly modifying the PTAS from Section 3. At first, the oracle writes down the value of η identified by running RSFF on the input sequence. For later convenience, we rename η by $\widetilde{w}_0$. This takes $O(s)$ bits of advice. This is analogous to running FFI in the original PTAS. Recall that the PTAS creates $k \le 1/\epsilon^2$ groups of large input items L_i for $i \in [k]$ with the corresponding rounded weights $\widetilde{w}_i$ for $i \in [k]$. The oracle appends $|L_i|$ together with $\widetilde{w}_i$ for $i \in [k]$ to the advice string. This completes the specification of the advice string. The length of the advice string is $O(s + k \log |L_i| + ks) = O\left(\frac{s+\log n}{\epsilon^2}\right)$.

It is left to see that with this advice string an online algorithm can compute a $1 + \Theta(\epsilon)$ approximate solution to the instance of the dual bin packing problem. Observe that if $\widetilde{w}_0 \le \epsilon$ then by Lemma 7 the solution obtained by running FF on all items of weight $\le \widetilde{w}_0$ achieves $1 + \epsilon$ approximation, since this gives us exactly the packing produced by RSFF. From now on, we consider the case $\widetilde{w}_0 > \epsilon$. Then an optimal solution might use large items.

Recall that the PTAS creates a solution to the rounded instance encoded by $(|L_1|, \dots, |L_k|)$ and weights $(\widetilde{w}_1, \dots, \widetilde{w}_k)$, replaces this solution with actual weights of the corresponding items and fills the rest in FF fashion with the rest of the items (see the remark immediately following Lemma 2). Thus, knowing $(|L_1|, \dots, |L_k|)$ and weights $(\widetilde{w}_1, \dots, \widetilde{w}_k)$ from the advice, our online algorithm can reserve place holders for items in bins according to the dynamic programming solution. We refer to this space as the preallocated space, and we refer to the complement of it as the remaining space. For example, if dynamic programming solution says that bin 1 contains ℓ_i items of weight $\widetilde{w}_i$ then the online algorithm reserves ℓ_i slots of weight $\widetilde{w}_i$ in bin 1. The preallocated space in bin 1 is $\sum_i \ell_i \widetilde{w}_i$ and the remaining space in bin 1 is $1 - \sum_i \ell_i \widetilde{w}_i$. Now, the algorithm is ready to process the items in the online fashion. When the algorithm receives an input item of weight $\le \epsilon$ it packs it in the remaining space in FF fashion. When the algorithm receives an item of weight $\in (\widetilde{w}_{i-1}, \widetilde{w}_i]$, it packs it into the first available preallocated slot of weight $\widetilde{w}_i$. By the construction of advice, we are guaranteed that when the algorithm is done processing the inputs, all preallocated slots are occupied and all small items are packed. By Theorem 6 this solution is a $1 + \epsilon$ approximation. ◀

5 Advice Complexity of the Online Dual Bin Packing Problem for General Weights

In this section we show that the online dual bin packing without any restrictions on s requires $\Theta_\epsilon(n)$ advice to approximate OPT within $1 + \epsilon$. Observe that the upper bound $O(n/\epsilon)$ immediately follows from the result of Renault [16] in the per-request advice model. A somewhat stronger upper bound, $(1 - \Omega(\epsilon))n$, follows by observing that the dual bin packing belongs to the advice complexity class AOC defined by Boyar et al. [6] and then using the results from [6]. Thus, we only need to prove that in the case of unrestricted s the lower bound of $\Omega_\epsilon(n)$ holds. For sufficiently small ϵ, we show a nearly matching lower bound of $(1 - O(\epsilon \log(1/\epsilon)))n = \Omega_\epsilon(n)$ in the tape-advice model. We establish our lower bound by providing a reduction (that preserves the precision, advice and competitive ratio) from an online problem known to require a lot of advice to the dual bin packing problem. The starting point is the binary separation problem defined by Boyar et al. [7].

▶ **Definition 9** (Boyar et al. [7]). The *binary separation problem* is the online problem with input $I = (n_1, y_1, \dots, y_n)$ consisting of $n = n_1 + n_2$ positive values which are revealed one by one. There is a fixed partitioning of the set of items into a subset of n_1 large items and a subset of n_2 small items, so that all large items are greater than all small items. Upon receiving an item y_i, an online algorithm for the problem must guess if y belongs to the set of small or large items. After the algorithm has made a guess, it is revealed whether the guess was correct. The goal is to maximize the number of correct guesses.

Boyar et al. [7] establish a lower bound on the advice needed to achieve competitive ratio c for the binary separation problem.

▶ **Theorem 10** (Boyar et al. [7]). *Assume that an online algorithm solves the binary separation problem on sequences $I = (n_1, y_1, \dots, y_n)$ where the y_i are n bit numbers and does so using at most $b(n)$ bits of advice while making at most $r(n)$ mistakes. Set $\alpha = (n - r(n))/n$. If $\alpha \in [1/2, 1)$ then $b(n) \ge (1 - H(\alpha))n$ where $H(p) = p \log(1/p) + (1 - p) \log(1/(1 - p))$.*

Moreover, Boyar et al. [7] provide a reduction from the binary separation problem to the standard bin packing problem to show that achieving competitive ratio $< 9/8$ requires an online algorithm to receive $\Omega(n)$ bits of advice. A simple adaptation of this reduction allows us to derive a similar result for the dual bin packing problem. We present the details below for completeness.

▶ **Theorem 11.** *An online algorithm achieving a competitive ratio* $1+\epsilon$ *for the dual bin packing problem with unrestricted bit size of input weights requires* $(1-O(\epsilon\log(1/\epsilon)))n = \Omega_\epsilon(n)$ *bits of advice, provided* $\epsilon < 1/19$.

Proof. We show how to reduce the binary separation problem to the dual bin packing problem while preserving the size of advice and the competitive ratio.

Let ALG be an algorithm for the dual bin packing problem that achieves competitive ratio c and uses advice $b(n)$. Let $I = (n_1, (y_1, \ldots, y_n))$ be an input to the binary separation problem. We define ALG' for solving I as follows. ALG' constructs an instance of the dual bin packing problem in the online fashion. It will use decisions and the advice string of ALG to make decisions about its own inputs y_i. Let $\delta_{\max} > \delta_{\min} > 0$ be small enough numbers. Suppose that we have a strictly decreasing function $f : \mathbb{R} \to (\delta_{\min}, \delta_{\max})$. ALG' invokes ALG with n bins and $2n$ items. ALG' constructs input weights to ALG in three phases.

Phase 1 (preprocessing): the first n_1 weights are defined as $1/2+\delta_{\min}$. This is generated by ALG' prior to any inputs seen from I.

Phase 2 (online): when y_i arrives, ALG' defines a new input item to ALG of weight $1/2 - f(y_i)$. In this phase ALG' uses decisions of ALG to handle its own inputs. If ALG packs the current item into a bin that contains $1/2+\delta_{\min}$ item from phase 1 then ALG' declares y_i to be large. Otherwise, ALG' declares the item to be small. We shall refer to $1/2 - f(y_i)$ weight items corresponding to truly small (large) y_i as small items (large items).

Phase 3 (post processing): once ALG' has processed the entire sequence I, it appends weights $1/2 + f(y_i)$ for all truly small y_i from I. We refer to these weights as the complementary weights of small items.

First observe that OPT for the constructed instance of the dual bin packing packs all $2n$ items into n bins: the n_1 weights corresponding to the large items can be paired up with the n_1 items from phase 1, and the n_2 weights corresponding to the small items can be paired up with their complementary weights from phase 3 in the remaining $n_2 = n - n_1$ bins.

Clearly, the advice complexity and the precision of the input items are preserved by this reduction. Thus, to finish the argument we need to analyze how many mistakes ALG' does. We bound the number of mistakes in terms of the number of items unpacked by ALG. We define the following variables.

- Let p_1 be the number of items from phase 1 that were not packed by ALG.
- Let ℓ_2 be the number of large items from phase 2 that were not packed by ALG.
- Let s_2 be the number of small items from phase 2 that were not packed by ALG.
- Let p_3 be the number of items from phase 3 that were not packed by ALG.

The overall number of items that were not packed by ALG is $p_1 + \ell_2 + s_2 + p_3 \le \frac{c-1}{c} 2n$. Observe that the complementary weights can only be paired up with the corresponding small item weights, and phase 1 items can only be paired up with large or small phase 2 items.

The number of bins containing phase 1 items is $n_1 - p_1$. The number of bins containing phase 3 items is $n_2 - p_3$. Due to the above observations, all these bins have to be distinct. Thus, the number of bins that do not contain either phase 1 or phase 3 items is $p_1 + p_3$. Call these the leftover bins. The are two types of mistakes that ALG' can do: (1) it classifies a large item as being small, and (2) it classifies a small item as being large. Since large items can only be paired either with phase 1 items or be placed in the leftover bins, type (1) mistakes occur only when large items are placed in the leftover bins or when large items remain unpacked. There can be at most $2(p_1 + p_3)$ large items in the leftover bins. Thus, ALG' makes at least $g_1 := n_1 - p_1 - 2(p_1 + p_3) - \ell_2$ correct guesses for large items. A type (2) mistake happens only when a small item is paired up with a phase 1 item. Since there can be at most $n_1 - p_1 - g_1 = 2(p_1 + p_3) + \ell_2$ phase 1 items not paired up with large items, there can be at most that many type (2) mistakes. Thus, ALG' makes at least $g_2 = n_2 - s_2 - 2(p_1 + p_3) - \ell_2$ correct guesses for small items. Overall, ALG' makes $g_1 + g_2 = n_1 + n_2 - s_2 - p_1 - 4(p_1 + p_3) - 2\ell_2 \geq n_1 + n_2 - 5(p_1 + \ell_2 + s_2 + p_3) \geq n - 10\frac{c-1}{c}n$ good guesses. The fraction of good guesses is then $\frac{n-10(c-1)n/c}{n} = \frac{10-9c}{c}$. By Theorem 10, it follows that $b(n) \geq (1 - H((10 - 9c)/c))n$. Observe that $(10 - 9c)/c \in (1/2, 1)$ provided that $c \in (1, 20/19)$. In particular, if ϵ is a small positive constant, then achieving a competitive ratio $c = 1 + \epsilon$ for the dual bin packing problem requires $(1 - H((1 - 9\epsilon)/(1 + \epsilon)))n = (1 - H(O(\epsilon)))n = (1 - O(\epsilon \log(1/\epsilon)))n = \Omega_\epsilon(n)$ bits of advice. ◀

All in all, the dual bin packing problem admits short advice in case of s bounded by a slowly growing function of n, but requires long advice when s is unrestricted. This is akin to the distinction between the polynomial time vs pseudo-polynomial time in the regular Turing machine world. One of the conceptual contributions of this paper is a demonstration that "pseudo-short" advice and "truly short" advice are provably different. To make this idea precise, we introduce two natural classes of efficient advice problems.

▶ **Definition 12.** The class *EAC (efficient advice complexity)* consists of online problems P such that an input to P is given by n items, and the advice complexity of achieving $1 + \epsilon$ competitive ratio for P is $O_\epsilon(\mathsf{poly}(\log n))$.

Denoting the maximum bit size of an input item to P by s, we define a superclass *WEAC (weakly efficient advice complexity)* of EAC to consist of those online problems P such that the advice complexity of achieving $1 + \epsilon$ competitive ratio for P is $O_\epsilon(\mathsf{poly}(\log n, s))$.

EAC class is defined by analogy with communication complexity where $O(\mathsf{poly}(\log n))$ communication is considered efficient (see Babai et al. [2]). WEAC class is also natural. The advice length bound of algorithms for WEAC problems suggests that the advice can consist of a short description of combinatorial parameters of a problem (e.g., length of a stream, index into a stream, which take $O(\log n)$ bits to describe) plus a small (polylogarithmic) number of actual data items from the stream.

In light of the above definitions and the main result of this section and Section 4, the dual bin packing problem witnesses the following class separation theorem.

▶ **Theorem 13.** *WEAC≠EAC.*

6 Conclusion

We presented a simple PTAS for the dual bin packing problem. Although a PTAS for a more general multiple knapsack problem was already known, our PTAS is arguably simpler to state and analyze. Its simplicity helped us to adapt it to the tape-advice model of online algorithms. We showed that a $1 + \epsilon$ competitive ratio for the dual bin packing problem is achievable with $O\left(\frac{s+\log n}{\epsilon^2}\right)$ bits of tape advice. We showed that the dependence on s is necessary to obtain such small advice, as the dual bin packing problem requires $\Omega_\epsilon(n)$ when $\epsilon > 0$ is small enough, s is unrestricted, and m is part of the input. We introduced two natural advice complexity classes EAC and WEAC. The conceptual distinction between the classes WEAC and EAC is similar to the Turing machine world distinction between pseudo-polynomial time and strongly polynomial time. EAC captures problems that can be approximated to within $1+\epsilon$ with $O_\epsilon(\mathsf{poly}\log n)$ bits of advice, whereas WEAC captures problems that can be approximated to within $1+\epsilon$ with $O_\epsilon(\mathsf{poly}(\log n, s))$ bits of advice. Our results on the dual bin packing problem imply that WEAC≠EAC.

One immediate question left open by our work is whether there is an small advice algorithm for small s which requires less advice bits. More specifically, does there exist a $1+\epsilon$ approximation using $o_\epsilon(s) + O_\epsilon(\log n)$ advice bits for $s = o(n)$? In this paper we exclusively studied the dual bin packing in the regime of obtaining $1+\epsilon$ competitive ratio when ϵ is small and m is part of the input. Are there sublinear advice algorithms for large ϵ, e.g., $\epsilon = 1/2$? Also, does the dual bin packing admit sublinear advice algorithms when m is a small constant? It is also interesting to see whether or not results for the dual bin packing problem can be extended to more general problems such as when bins have different capacities, and more generally to the multiple knapsack problem, while preserving conceptual simplicity. Last and perhaps a most important question is whether or not there exist online algorithms with efficiently computable (i.e., linear or even online computable as in [12, 4]) advice for the dual bin packing problem achieving a constant competitive ratio.

Acknowledgements. We thank the anonymous reviewers, especially reviewer 2, for their helpful comments.

References

1. Susan F. Assmann, David S. Johnson, Daniel J. Kleitman, and Joseph Y.-T. Leung. On a dual version of the one-dimensional bin packing problem. *J. Algorithms*, 5(4):502–525, 1984. `doi:10.1016/0196-6774(84)90004-X`.
2. Laszlo Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proc. of the 27th Symp. on Found. of Comput. Sci.*, SFCS '86, pages 337–347, 1986.
3. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. *Algorithms and Computation*, pages 331–340, 2009.
4. Allan Borodin, Denis Pankratov, and Amirali Salehi-Abari. On conceptually simple algorithms for variants of online bipartite matching. In *WAOA'17: The 15th workshop on approximation and online algorithms (To appear)*, 2017.
5. Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: a survey. *ACM SIGACT News*, 47(3):93–129, 2016.
6. Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen. The advice complexity of a class of hard online problems. *Theory of Comput. Sys.*, 2016.
7. Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, Jan 2016.
8. Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. The accommodating function: A generalization of the competitive ratio. *SIAM J. on Comput.*, 31(1):233–258, 2001.
9. Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. on Comput.*, 35(3):713–728, 2005.
10. János Csirik and V. Totik. Online algorithms for a dual version of bin packing. *Discrete Applied Mathematics*, 21(2):163–167, 1988. `doi:10.1016/0166-218X(88)90052-2`.
11. Marek Cygan, Lukasz Jez, and Jirí Sgall. Online knapsack revisited. *Theory Comput. Sys.*, 58(1):153–190, 2016.
12. Christoph Dürr, Christian Konrad, and Marc Renault. On the Power of Advice and Randomization for Online Bipartite Matching. In *Proc. of ESA*, pages 37:1–37:16, 2016.

13 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.
14 Edward G. Coffman Jr., Joseph Y.-T. Leung, and D. W. Ting. Bin packing: Maximizing the number of pieces packed. *Acta Inf.*, 9:263–271, 1978.
15 Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Proc. of RANDOM-APPROX*, volume 1671, pages 51–62. Springer, 1999.
16 Marc P Renault. Online algorithms with advice for the dual bin packing problem. *Central Eur. J. of Op. Res.*, pages 1–14, 2016.
17 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

Simple and Efficient Leader Election

Petra Berenbrink[1], Dominik Kaaser[2], Peter Kling[3], and Lena Otterbach[4]

1 **Universität Hamburg, Germany**
`berenbrink@informatik.uni-hamburg.de`
2 **Universität Hamburg, Germany**
`dominik.kaaser@uni-hamburg.de`
3 **Universität Hamburg, Germany**
`peter.kling@uni-hamburg.de`
4 **Universität Hamburg, Germany**
`otterbach@informatik.uni-hamburg.de`

Abstract

We provide a simple and efficient population protocol for leader election that uses $\mathrm{O}(\log n)$ states and elects exactly one leader in $\mathrm{O}\big(n \cdot (\log n)^2\big)$ interactions with high probability and in expectation. Our analysis is simple and based on fundamental stochastic arguments. Our protocol combines the tournament based leader elimination by Alistarh and Gelashvili, ICALP'15, with the synthetic coin introduced by Alistarh et al., SODA'17.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.3.16 Stochastic Processes

Keywords and phrases population protocols, leader election, distributed, randomized

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.9

1 Introduction

We consider the *leader election* problem for *population protocols* introduced by [4], where one seeks a simple, distributed protocol that establishes a leader in a system of n initially identical agents. In this problem, in each round a pair of randomly chosen agents interact. The interacting agents observe each other's state and update their own state according to a simple deterministic rule, which is identical for each agent. A protocol's quality is measured by the number of interactions until a unique leader is found and by the number of states per agent required by the protocol. A key aspect of this model is that a unique leader *must* be found eventually. In particular, the protocol may not fail even with negligible probability.

Related Work. We give an overview of recent results in population protocols, with a focus on the leader election problem. We refer to [6] or the more recent [1] for a general survey on population protocols.

[4] introduce the population protocol model. They present protocols that stably compute any predicate definable via Pressburger arithmetic, which includes fundamental distributed tasks like leader election or consensus. [5, 6] show that predicates stably computable by population protocols are semi-linear. These early results restrict the number of states per agent to a constant and focus on what can and cannot be computed (in contrast to what can be computed efficiently). [8] prove that any population protocol that elects a leader with a constant number of states requires an expected number of $\Omega\big(n^2\big)$ interactions. Under some natural protocol assumptions (met, as far as we know, by all known population protocols), [1]

strengthen this lower bound by showing that population protocols using less than $1/2 \cdot \log\log n$ states need an expected number of $\Omega\big(n^2/\operatorname{polylog} n\big)$ interactions to elect a leader (their lower bound holds also for a broader class of problems).

To beat this polynomial lower bound on the time to elect a leader, recent results consider population protocols with polylogarithmically many states. [3] present a tournament based protocol that elects a leader in $\mathrm{O}\big(n \cdot (\log n)^3\big)$ expected interactions using $\mathrm{O}\big((\log n)^3\big)$ states. This protocol is quite intuitive and simple: Each leader candidate has a counter that is increased whenever it interacts with another agent. When a leader candidate meets an agent with a larger counter, it becomes a *minion*. Minions copy the largest counter seen so far. The main idea of the analysis is to show that, after $\mathrm{O}\big(n \cdot (\log n)^3\big)$ interactions, one of the remaining leaders, say v, has a counter that exceeds any other leader's counter by $\Theta(\log n)$. This head start allows v to broadcast its counter to all other remaining leaders before their counters catch up.

[1] decrease the number of states to $\mathrm{O}\big((\log n)^2\big)$ at the cost of an increased number of $\mathrm{O}\big(n \cdot (\log n)^{5.3} \cdot \log\log n\big)$ expected interactions and a much more involved protocol. A key part of their protocol is the use of *synthetic coins*, which allow agents to access a random bit. More precisely, each agent has a bit that is flipped at the end of each interaction. One can show that, after roughly a linear number of interactions, about half of the agents have their bit set. Thus, by accessing the bit of its interaction partner (which is chosen uniformly at random), an agent has access to an almost uniformly random bit.

Three very recent, yet unpublished results [7, 2, 9] further improve upon these bounds. [7] present a protocol that requires $\mathrm{O}\big(n \cdot (\log n)^2\big)$ interactions in expectation and $\mathrm{O}\big((\log n)^2\big)$ states. [2] reduce the number of states to $\mathrm{O}(\log n)$ while maintaining the number of required interactions. Finally, [9] further reduce the number of states to $\mathrm{O}(\log\log n)$, matching the lower bound mentioned above. All these protocols and analysis are rather involved. In particular, [2, 9] are based on a phase clock to actively synchronize the behavior of the agents.

Our Contribution. We introduce a natural and simple leader election protocol that elects a single leader in $\mathrm{O}\big(n \cdot (\log n)^2\big)$ expected interactions and uses $\mathrm{O}(\log n)$ states. Our analysis is simple and based on fundamental stochastic arguments. It combines the tournament based leader elimination from [3] with the synthetic coin introduced in [1]. Using the synthetic coin, we initially mark $n/\log n$ agents. Since an agent's interaction partners are chosen uniformly at random, this effectively gives each agent access to a $(1/\log n)$-coin. This allows agents participating in the tournament to increase their counter only with a probability of $1/\log n$. As a result, we can show that an agent only needs a constant head start to broadcast its counter to all other remaining leaders before their counters can catch up. Our analysis relies on a simplified and slightly stronger analysis of the synthetic coin from [1].

Formally, we show the following theorem.

▶ **Theorem 1.** *With high probability*[1] *and in expectation, the protocol defined in Algorithm 1 elects exactly one leader in $\mathrm{O}\big(n \cdot (\log n)^2\big)$ interactions. Furthermore, the protocol eventually reaches and stays in a configuration where exactly one leader contender is left with probability* 1.

2 Model and Protocol

We consider a population of n agents[2], also referred to as nodes. A *population protocol* specifies a set of possible *states*, the initial state of each agent, and an *update rule*. The

[1] The expression *with high probability* refers to a probability of $1 - n^{-\Omega(1)}$.
[2] All our results assume n to be larger than a suitable constant.

(deterministic) update rule is defined from the perspective of a single agent that knows its own state and the state of its communication partner. All agents start in the same initial state and use the same update rule. In every round a pair of agents is *activated* uniformly at random. In such an *interaction* the two activated nodes observe each other's state and apply the update rule. The goal is to reach a configuration where exactly one agent's state labels the agent as a (potential) leader and all other agents know that they are not a leader. Additionally, we require that every following configuration also have exactly one leader.

Protocol

We start with an informal description of our protocol. Every node has a *counter* and uses it to compete with other nodes. In the beginning some nodes will be marked. Leader candidates only increment their counter if they interact with a marked node. To initially mark a small fraction of nodes the protocol is split into two phases: the *marking phase* and the *tournament phase*.

Marking Phase. In the first phase, $\Theta(n/\log n)$ nodes get marked, see Section 3. To derive this, each node is equipped with an additional bit, the *flip bit*, that is flipped at the end of each interaction. After its first $3 \log\log n$ activations, a node starts to study the flip bits of its interaction partners. It marks itself if and only if all of its next $\log\log n$ interaction partners have their flip bits set. We refer to these at most $\log\log n$ crucial interactions as a node's *marking trials*. After a node's marking trials, it enters the tournament.

Tournament Phase. The second phase is responsible for electing a unique leader, see Section 4. At the beginning of its tournament phase every node is a possible leader and regards itself as a *contender*. Contenders count the number of their interactions with marked nodes. Whenever a contender interacts with another agent having a larger counter, it sets its role to *minion*. Minions carry the largest counter seen so far. Since the counter values never decrease we always have at least one contender left (see Lemma 6). The single remaining contender will be the unique leader.

Below, we summarize the parameters that constitute the state of a node v.

- **role** $r(v) \in \{\,\texttt{contender},\ \texttt{minion}\,\}$**.** Each node starts as a contender.
- **flip bit** $f(v) \in \{\,0, 1\,\}$**.** Initially the flip bit is set to 0. The flip bit will be used to approximate a random coin which is zero or one with probability 1/2.
- **marker** $m(v) \in \{\,0, 1\,\}$**.** Initially the marker is set to 0 for *unmarked*. Nodes that mark themselves after their marking trials set this marker to 1. The marked nodes will be used in the tournament phase to approximate a random coin with a probability $1/\log n$ to be one.
- **phase** $p(v) \in \{\,\texttt{marking},\ \texttt{tournament}\,\}$**.** Each node starts in the marking phase.
- **counter** $c(v) \in \{\,0, \dots, \mathrm{O}(\log n)\,\}$**.** The counter, initialized to 0, is used in both phases: In the marking phase to skip the first $3 \log\log n$ activations and then count the marking trials. In the tournament phase, the counter is used to determine the *winner* of an encounter.

In the following we assume $c(v)$ to be a variable that may count up to $\mathrm{O}(\log n)$. Hence, $c(v)$ can assume $\mathrm{O}(\log n)$ different values. Each of the parameters role, flip bit, marker, and phase only doubles the state space. Therefore, the total number of states per node is $\mathrm{O}(\log n)$.

For the case that all nodes reach the maximal possible counter value before all but one contender are eliminated, we let contenders with equal counter compete via their flip bits.

```
Algorithm leader-election(node $v$, node $u$)
    ——————————— marking phase ———————————
    if phase $p(v) =$ marking then
        if counter $c(v) \geq 3 \log\log n$ and flip bit $f(u) = 0$ then
            phase $p(v) \leftarrow$ tournament;                 /* leave phase unmarked */
        else
            increment counter $c(v) \leftarrow c(v) + 1$;
        if counter $c(v) = 4 \log\log n$ then
            marker $m(v) \leftarrow 1$;
            phase $p(v) \leftarrow$ tournament;                   /* leave phase marked */
    ——————————— tournament phase ———————————
    if phase $p(v) =$ tournament then
        if role $r(v) =$ contender then
            if marker $m(u) = 1$ and counter $c(v) \leq U \log n$ then
                increment counter $c(v) \leftarrow c(v) + 1$;
            if $c(v) < c(u)$ then
                role $r(v) \leftarrow$ minion;          /* lose the duel due to the counter */
            if $r(u) =$ contender and $c(v) = c(u)$ and $f(v) < f(u)$ then
                role $r(v) \leftarrow$ minion;         /* lose the duel due to the flip bit */
        update counter $c(v) \leftarrow \max\{c(u), c(v)\}$;      /* adopt the maximum counter */
    ————————————————————————————————————————
    flip the flip bit $f(v) = 1 - f(v)$;
```

Algorithm 1 The leader election algorithm from the perspective of a single node v upon an interaction with communication partner u. Here, U is a large enough constant.

The complete update rule from the viewpoint of a node v interacting with a node u is formally defined in Algorithm 1. To avoid concurrency issues, we assume that node v operates on values of node u as they were *before* the interaction.

3 Analysis of the Marking Phase

In the first part of the analysis, our goal is to prove the following proposition, which states that after the marking phase, roughly $n/\log n$ nodes are marked. We use this in Section 4 to prove our main result.

▶ **Proposition 2.** *With high probability, after $4 \ln n$ interactions $\Theta(n/\log n)$ nodes are marked and all nodes are in the tournament phase.*

The proof of Proposition 2 works as follows: A node marks itself if and only if all communication partners of its $\log\log n$ marking trials have their bit set. If the bit of an interaction partner were set with probability $1/2$, this would imply that v marks itself with probability $1/2^{\log\log n} = 1/\log n$ and the desired result would follow via Chernoff bounds. The major difficulty is to show that, when a node starts its marking trials, the probability that a flip bit is set is close to $1/2$. We prove this in Section 3.1. Additionally, we have to show that not too many nodes start their marking trials before the balancing of the flip bits has finished. This is done in Section 3.2. The proof of Proposition 2 is finally given in Section 3.3.

3.1 Concentration of the $1/2$-Coin

As described in the overview, the major technical tool to prove Proposition 2 is the following concentration result for the number of flip bits set:

▶ **Lemma 3.** *Let $a > 0$ and consider an interaction t with $n \cdot \ln(\log\log n)/2 \leq t \leq n^a$. The number of flip bits that equal zero at the beginning of interaction t lies with probability at least $1 - n^{-a}$ in $(1 \pm 1/\log\log n) \cdot n/2$.*

The flip bit of a node is set if and only if the node attended an odd number of interactions. Observe that the number of interactions a node attended can be modeled by a balls into bins game: Nodes correspond to bins and activations to balls. For each interaction, we throw two balls into two random bins. Nodes with flip bit equal zero correspond to bins with an even load. Analyzing the number of such bins directly is difficult, since the bins' loads are correlated. However, we can use the Poisson approximation technique (see, e.g., the textbook [11, Chap. 5.4]).

More formally, assume we throw m balls independently and uniformly at random into n bins. Let X_i be the resulting load of bin i for $i \in \{1, 2, \ldots, n\}$. Additionally, let Y_i for $i \in \{1, 2, \ldots, n\}$ denote independent Poisson random variables with parameter m/n. We call $(X_1, \ldots, X_n)$ the *exact case* and $(Y_1, \ldots, Y_n)$ the *Poisson case*. The following well-known result relates these processes:

▶ **Known Result 1** (Corollary 5.9, [11]). *Any event that takes place with probability p in the Poisson case takes place with probability at most $pe\sqrt{m}$ in the exact case.*

Recall that we are interested in the number of bins with even load. This can be easily bounded in the Poisson case:

▶ **Lemma 4.** *Let $a > 0$ and $Y_1, \ldots, Y_n$ be independent Poisson random variables, each with parameter $\lambda \geq 2$. Define $\alpha := \min\{\lambda, \ln n/8\}$. The number of variables that are even lies with probability at least $1 - n^{-a}$ in $(1 \pm e^{-\alpha}) \cdot n/2$.*

Proof. One easily verifies that the probability for Y_i to be even is

$$\Pr[Y_i \text{ is even}] = \frac{1}{2} \cdot \left(1 + e^{-2\lambda}\right), \tag{1}$$

see Appendix A. Let the indicator random variable Z_i be 1 if and only if Y_i is even and 0 otherwise. By construction $Z_1, \ldots Z_n$ are independent 0-1 random variables and $Z = \sum_{i=1}^{n} Z_i$ is the number of variables that are even. By Equation (1), $\mathbb{E}[Z] = n \cdot \left(1 + e^{-2\lambda}\right)/2$. Set $\delta := e^{-2\alpha}$ and note that, since $\lambda \geq \max\{2, \alpha\}$, we have $\left(1 + e^{-2\lambda}\right) \cdot (1 + \delta) \leq (1 + e^{-\alpha})$ and $\left(1 + e^{-2\lambda}\right) \cdot (1 - \delta) \geq (1 - e^{-\alpha})$. Thus, standard Chernoff bounds (Lemma 11) yield

$$\begin{aligned} &\Pr\left[Z \geq \frac{n}{2}(1 + e^{-\alpha})\right] \leq \Pr[Z \geq (1 + \delta) \cdot \mathbb{E}[Z]] \leq e^{-\mathbb{E}[Z]\delta^2/3} \leq \frac{n^{-a}}{2} \quad \text{and} \\ &\Pr\left[Z \leq \frac{n}{2}(1 - e^{-\alpha})\right] \leq \Pr[Z \leq (1 - \delta) \cdot \mathbb{E}[Z]] \leq e^{-\mathbb{E}[Z]\delta^2/2} \leq \frac{n^{-a}}{2} \ . \end{aligned} \tag{2}$$

Combining both bounds gives the desired statement. ◀

Proof of Lemma 3. Fix an interaction t with $n \cdot \ln(\log\log n)/2 \leq t \leq n^a$ and set $\alpha := \min\{2t/n, \ln n/8\}$. Note that $\alpha \geq \ln(\log\log n)$. Let X denote the number of nodes that have their flip bit equal 0. As mentioned above, X also equals the number of bins with an even load when we throw $2t$ balls into n bins chosen independently and uniformly at random in the exact case. Let Y be the number of bins with an even load in the Poisson case (that is, each of the independent n Poisson random variables has parameter $\lambda = 2t/n$). By Known Result 1, we know that for any set $\mathcal{A} \subseteq \{0, 1, \dots, n\}$ $\Pr[X \in \mathcal{A}] \leq e\sqrt{2t} \cdot \Pr[Y \in \mathcal{A}]$. Let $\mathcal{A} := [0, (1 - e^{-\alpha}) \cdot n/2) \cup ((1 + e^{-\alpha}) \cdot n/2, n]$. By Lemma 4, $e\sqrt{2t} \cdot \Pr[Y \in \mathcal{A}] \leq n^{-a}$. Using that $e^{-\alpha} \leq 1/\log\log n$, we get the desired statement. ◀

3.2 Bounding the Number of Early Marking Trials

By Lemma 3 we know that if a node starts its marking trials after (global) interaction $n \cdot \ln(\log\log n)/2$, the fraction of flip bits equal zero in the system is very close to 1/2. In the following, we bound the number of nodes that start their marking trials earlier.

▶ **Lemma 5.** *Let $a > 0$. With probability $1 - n^{-a}$, at most $n/\log n$ nodes start their marking trials before the $\big(n \cdot \ln(\log\log n)/2\big)$-th (global) interaction.*

Proof. Fix interaction $T_0 := n \cdot \ln(\log\log n)/2$ and consider the number of nodes that start their marking trials before T_0. We analyze this number using the Poisson approximation technique.

Performing T_0 global interactions corresponds to throwing $2T_0$ balls. Hence, we consider independent Poisson random variables $Y_1, \dots, Y_n$, each with parameter $\lambda := 2T_0/n = \ln(\log\log n)$. A node starts its marking trials once it got activated $t := 3\log\log n$ times. The Chernoff bound for Poisson random variables (Lemma 12) gives

$$\begin{aligned}\Pr[Y_i \geq t] &\leq \frac{e^{-\lambda}(e\lambda)^t}{t^t} = \frac{1}{\log\log n}\left(\frac{e\ln(\log\log n)}{3\log\log n}\right)^{3\log\log n} \\ &\leq \frac{1}{\log\log n}\left(\frac{1}{2}\right)^{3\log\log n} \leq \frac{1}{(\log n)^3},\end{aligned} \tag{3}$$

where the second inequality follows from $\ln(x)/x \leq 1/2$ for any $x > 0$. Now consider binary random variables Z_i that are 1 if and only if $Y_i \geq t$ and let $Z := \sum_{i=1}^{n} Z_i$. It is $\mathbb{E}[Z] \leq n/(\log n)^3 \leq n/(2\log n)$. Lemma 11 implies for $\delta := 1$

$$\Pr\left[Z \geq \frac{n}{\log n}\right] \leq \Pr[Z \geq (1+\delta) \cdot \mathbb{E}[Z]] \leq e^{-n/(3(\log n)^3)} \leq e^{-(a+2)\ln n} = n^{-(a+2)}. \tag{4}$$

As in the proof of Lemma 3, we can now apply Known Result 1 to get the same guarantee for the exact case with probability n^{-a}. Therefore, with probability $1 - n^{-a}$, at most $n/\log n$ nodes have been activated more than $3\log\log n$ times before the T_0-th interaction, finishing the proof. ◀

3.3 Proof of Proposition 2

We show that, with high probability, after $T := 4n\ln n$ interactions all nodes are in their tournament phase and at least $n \cdot (1 - 1/\log n)$ of them have marking probability $\Theta(1/\log n)$. We conclude that the expected number of marked nodes is $\Theta(n/\log n)$ and use Chernoff to get the same result with high probability.

A node v enters the tournament phase at the latest when it was activated $4\log\log n$ times. Let N denote the number of interactions in which v was activated at the end of interaction T. Then $\mathbb{E}[N] = 2T/n = 8\ln n$. Set $\delta := 1/\sqrt{2}$ and note that $4\log\log n \leq 2\ln n \leq (1 - \frac{1}{\sqrt{2}})8\ln n = (1-\delta) \cdot \mathbb{E}[N]$. Hence, Lemma 11 implies

$$\Pr[N \leq 4\log\log n] \leq e^{-\mathbb{E}[N] \cdot \delta^2/2} = e^{-2\ln n} = n^{-2}. \tag{5}$$

A union bound over all nodes yields that, with high probability, all nodes are in the tournament phase after interaction T.

Lemma 3 implies that at the end of any interaction t with $T \geq t \geq T_0 := n \cdot \ln(\log\log n)/2$ the number of flip bits set lies in $n - (1 \pm 1/\log\log n) \cdot n/2$ with probability at least $1 - n^{-3}$. Thus, via a union bound over the $T - T_0 < 4n \ln n$ many interactions in the interval $[T_0, T]$, with high probability the number of flip bits set lies in $n - (1 \pm 1/\log\log n) \cdot n/2$ during the whole interval $[T_0, T]$. We use $\mathcal{E}$ to denote this event. Now consider a node v that has all of its marking trials during $[T_0, T]$ and let M_v be the event that v leaves its marking phase marked. Using that $(1 - 1/\log\log n)^{\log\log n} \geq 1/(2e)$ and $(1 + 1/\log\log n)^{\log\log n} \leq e$ we get

$$\frac{1}{2e \log n} \leq \Pr[M_v \mid \mathcal{E}] \leq \frac{e}{\log n}. \tag{6}$$

By Lemma 5, with high probability there are no more than $n/\log n$ nodes that start their marking trials before interaction T_0. Denote this high probability event with $\mathcal{E}'$. The two worst case scenarios are that all or non of the early nodes get marked. Let M be the number of marked nodes after T interactions. With the above argumentation we obtain

$$\frac{n}{6\log n} \leq \left(n - \frac{n}{\log n}\right) \cdot \frac{1}{2e \log n} \leq \mathbb{E}[M \mid \mathcal{E}, \mathcal{E}'] \leq n \cdot \frac{e}{\log n} + \frac{n}{\log n} \leq \frac{4n}{\log n}. \tag{7}$$

Applying Lemma 11 with $\delta = 1/4$ and $\delta = 5/6$, respectively, yields

$$\Pr\left[M \geq \frac{5n}{\log n} \,\middle|\, \mathcal{E}, \mathcal{E}'\right] \leq e^{-\frac{n}{288 \log n}} \qquad \text{and} \qquad \Pr\left[M \leq \frac{n}{\log n} \,\middle|\, \mathcal{E}, \mathcal{E}'\right] \leq e^{-\frac{25n}{432 \log n}}. \tag{8}$$

Thus, using the law of total probability and the fact that the events $\mathcal{E}$, $\mathcal{E}'$ happen with high probability, we get that $M \in \Theta(n/\log n)$ with high probability, finishing the proof. ◀

4 Analysis of the Main Algorithm

In the following section we analyze the tournament phase of our protocol. First we show that, at the beginning of any interaction, at least one node will have the contender role.

▶ **Lemma 6.** *At the beginning of any interaction, there will be at least one contender.*

Proof. Let $M = \max\{\, c(v) \,\}$ be the maximum counter value of all nodes. We observe that from the definition of the protocol in Algorithm 1 it follows that a node's counter cannot decrease. We show by an induction over the number of interactions that there always exists at least one contender which has the largest counter.

Initially, all nodes have the role contender and counter value 0. Therefore the base of the induction holds. For the induction step consider an arbitrary but fixed interaction between v and u. W.l.o.g. assume that $c(v) \geq c(u)$. We distinguish the following two cases, depending on the status of v and u before the interaction.

Case 1: $c(v) = M$**.**
If node v is a contender, it can only become a minion upon interaction with another contender u with $c(u) = M$ and $f(u) = 1$ while $f(v) = 0$. In this case, however, u remains a contender with maximal counter. It might also happen that the maximal counter value increases to $M + 1$ while v still has $c(v) = M$. In that case, however, u will have $c(u) = M + 1$ and thus u will be a contender with maximal counter value.

Case 2: $c(v) < M$.
Since both v and u do not have maximal counter value, the number of contenders having maximal counters cannot decrease.

Together, these two cases yield the induction step and the lemma follows. ◀

It is easy to see that the maximum counter values are spread through the system like messages in the case of push/pull broadcasting (see, e.g., [10]). The following observation is an adaption of the results for randomized broadcasting algorithms to our setting.

For any interaction t let $\overline{C}(t)$ and $\underline{C}(t)$ denote the maximal and minimal counter value of all nodes after interaction t.

▶ **Observation 7.** *Fix an interaction t. With probability at least $1-n^{-3}$ the maximal counter is broadcast to all nodes in $4n\log n$ interactions: $\underline{C}(t+4n\log n) \geq \overline{C}(t)$.*

We use this observation to obtain the following corollary.

▶ **Corollary 8.** *Fix an interaction t. Let $\mathcal{E}_t$ be the event that $\underline{C}(t+4n\log n) < \overline{C}(t)$. We have $\Pr\left[\bigcup_{t=1}^{\Theta(n\cdot(\log n)^2)} \mathcal{E}_t\right] \leq 1/n$.*

Proof. Note that for an interaction t the event $\mathcal{E}_t$ is precisely the complementary event from the one characterized in Observation 7. Therefore, $\Pr[\mathcal{E}_t] \leq n^{-3}$. The corollary follows from union bound over the $\Theta(n \cdot (\log n)^2)$ interactions. ◀

From Observation 7 and Corollary 8 we obtain that whenever a contender increments its counter, after at most $4n\log n$ interactions, all nodes have at least the same value.

▶ **Lemma 9.** *Let v_1, v_2 with $v_1 \neq v_2$ be two contenders which are both in the tournament phase. With constant probability $p_{\mathrm{L9}} = \Theta(1)$ one of the two contenders becomes a minion after $8n\log n$ interactions.*

Proof. W.l.o.g. assume that $c(v_1) \geq c(v_2)$. We split the $8n\log n$ interactions into two parts and show that with constant probability v_1 increments its counter in the first part, while v_2 does not increment its counter at all in both parts.

In each interaction, the probability that v_1 is selected interacts with a marked node is $\Theta(1/(n\log n))$. This follows directly from the number of marked nodes, see Proposition 2. Let p_1 be the probability that v interacts with a marked node in $4n\log n$ interactions. For the complementary event we get

$$\overline{p_1} = 1 - p_1 = \left(1 - \Theta\left(\frac{1}{n\log n}\right)\right)^{4n\log n} = e^{-\Theta(1)}$$

and thus $p_1 = \Theta(1)$.

Let p_2 be the probability that v_2 does not interact with a marked node and thus does not increment its counter in all $8n\log n$ interactions. The numbers of interactions of node v_1 and node v_2 are not independent, they are negatively correlated. To obtain a lower bound on p_2, we assume that in the worst case v_1 does not interact at all. Under this assumption, the probability that v_2 is selected is $2/(n-1)$ and thus the probability that v_2 does not interact with a marked node is at least

$$p_2 \geq \left(1 - \frac{2}{n-1} \cdot \Theta\left(\frac{1}{\log n}\right)\right)^{8n\log n} = \Theta(1).$$

From Corollary 8 we obtain that in the second part of the $8n \log n$ interactions v_2 will *see* a counter value which is as least as large as the counter value of v_1 after the first part of the interactions, with high probability. We use union bound on above probabilities and the result from Corollary 8 and conclude that with constant probability $p_{\text{L9}} \geq p_1 \cdot p_2 - 1/n = \Theta(1)$ the node v_2 becomes a minion. ◀

Together with Lemma 6 and Corollary 8, the above lemma forms the basis for the proof of our main theorem, Theorem 1. Our main result is restated as follows.

▶ **Theorem 1.** *With high probability and in expectation, the protocol defined in Algorithm 1 elects exactly one leader in* $\mathrm{O}\big(n \cdot (\log n)^2\big)$ *interactions. Furthermore, the protocol eventually reaches and stays in a configuration where exactly one leader contender is left with probability* 1.

Proof. Let v_1 and v_2 be an arbitrary but fixed pair of contenders. We denote the first interaction when both v_1 and v_2 have entered the tournament phase as t_0. From Proposition 2 we obtain that with high probability $t_0 = \mathrm{O}(n \log n)$. Starting with interaction t_0, we consider $5/\log(1/(1-p_{\text{L9}})) \cdot \log n = \Theta(\log n)$ so-called *periods* consisting of $8n \log n$ interactions each. More precisely, the i-th period consists of interactions in $[t_{i-1}, t_i)$ for $1 \leq i \leq 5/\log(1/(1-p_{\text{L9}})) \cdot \log n$, where $t_i = t_0 + i \cdot 8n \log n$.

From Lemma 9 we know that with constant probability p_{L9} either v_1 or v_2 becomes a minion in each period. Therefore, with constant probability $1 - p_{\text{L9}}$ both nodes v_1 and v_2 remain contenders in one period. After $5/\log(1/(1-p_{\text{L9}})) \cdot \log n = \Theta(\log n)$ periods, the probability that v_1 and v_2 both remain contenders is at most $1/n^5$.

We take the union bound over all n^2 pairs of nodes and obtain a probability of at least $1 - 1/n^3$ that from each pair at least one node becomes a minion. From Lemma 6 we know that we always have at least one contender. Obviously, in any pair of nodes this contender cannot be the one to become a minion. Together, this implies that we have at least one contender and after $t_0 + \Theta(\log n) \cdot 8n \log n = \mathrm{O}\big(n \cdot (\log n)^2\big)$ interactions we have exactly one remaining contender, with high probability. All other nodes become minions with high probability, which shows the first part of the theorem.

To argue that the claimed run time also holds in expectation, we observe that the definition of the algorithm includes a *backup protocol* based on the flip bits. Observe that a similar approach to use a backup protocol has also been described in [3] and in [7]. Intuitively, whenever two contenders with the same counter value interact, there is a constant probability that one of them becomes a minion due to the flip bits. This backup protocol reduces the number of contenders to one in $\mathrm{O}\big(n^2 \log n\big)$ interactions in expectation. This follows from the coupon collector's problem. Since the probability that our main protocol fails and thus the backup protocol is actually needed is at most $\mathrm{O}\big(1/n^3\big)$, we obtain that our result also holds in expectation.

Finally, to show that the protocol eventually reaches a state where exactly one contender – the *leader* – is left, we observe the following. From any state of the system which is reachable over a sequence of interactions from the initial configuration, it is straight forward to specify a finite sequence of interactions such that all but one nodes become a minion. That means, at any time we have a positive probability to reach a stable state within finitely many interactions, and thus with probability 1 eventually only one contender will be left. ◀

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-Space Trade-offs in Population Protocols. In *Proc. SODA*, pages 2560–2579, 2017.

2 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. *CoRR*, abs/1704.04947, 2017.

3 Dan Alistarh and Rati Gelashvili. Polylogarithmic-Time Leader Election in Population Protocols. In *Proc. ICALP*, pages 479–491, 2015.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

5 Dana Angluin, James Aspnes, and David Eisenstat. Stably Computable Predicates Are Semilinear. In *Proc. PODC*, pages 292–299, New York, NY, USA, 2006.

6 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

7 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. *CoRR*, abs/1705.01146, 2017.

8 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *CoRR*, abs/1502.04246, 2015.

9 Leszek Gasieniec and Grzegorz Stachowiak. Fast Space Optimal Leader Election in Population Protocols. *CoRR*, abs/1704.07649, 2017.

10 Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized Rumor Spreading. In *Proc. FOCS*, pages 565–574, 2000.

11 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

A Appendix

Let $\mathbb{N}$ be the natural numbers including zero.

▶ **Definition 10** (Poisson Random Variable)**.** A discrete *Poisson random variable* Y with parameter λ is given by the following probability distribution on $\mathbb{N}$: $\Pr[Y = k] = e^{-\lambda}\lambda^k/k!$ for all $k \in \mathbb{N}$.

▶ **Lemma 11** (Chernoff Bounds [11] (Th. 4.4, Th. 4.5))**.** *Let $Z_1, \ldots, Z_n$ be independent Poisson trials such that $Pr(Z_i) = p_i$. Let $Z = \sum_{i=1}^{n} Z_i$ and $\mu_L \le \mathbb{E}[Z] \le \mu_U$.*[3] *Then,*

- $\Pr[Z \ge (1+\delta)\mu_U] \le e^{-\mu_U \delta^2/3}$ *for* $0 < \delta \le 1$ *and*
- $\Pr[Z \le (1-\delta)\mu_L] \le e^{-\mu_L \delta^2/2}$ *for* $0 < \delta < 1$.

▶ **Lemma 12** (Chernoff for Poisson Variables [11] (Th. 5.4))**.** *Let X be a Poisson random variable with parameter λ.*

- *If $x > \lambda$, then* $\Pr(X \ge x) \le \frac{e^{-\lambda}(e\lambda)^x}{x^x}$.
- *If $x < \lambda$, then* $\Pr(X \le x) \le \frac{e^{-\lambda}(e\lambda)^x}{x^x}$.

[3] While [11] states these bounds in terms of $\mu = \mathbb{E}[Z]$, it is easy to see and also mentioned in [11] that the Chernoff bounds hold also for suitable lower and upper bounds on $\mathbb{E}[Z]$.

Proof of Equation (1). Let Y be a Poisson random variable with parameter λ. Using the Taylor series $e^x = \sum_{k\in\mathbb{N}} \frac{x^k}{k!}$, we obtain

$$
\begin{aligned}
\Pr[Y \text{ is even}] &= \sum_{k\in\mathbb{N}} \Pr[Y = 2k] = \sum_{k\in\mathbb{N}} \frac{e^{-\lambda}\lambda^{2k}}{(2k)!} \\
&= \frac{e^{-\lambda}}{2}\left(\sum_{k\in\mathbb{N}} \frac{\lambda^{2k}}{(2k)!} + \sum_{k\in\mathbb{N}} \frac{\lambda^{2k}}{(2k)!}\right) \\
&= \frac{e^{-\lambda}}{2}\left(\sum_{k\in\mathbb{N}} \frac{\lambda^{2k}}{(2k)!} + \sum_{k\in\mathbb{N}} \frac{\lambda^{2k+1}}{(2k+1)!} + \sum_{k\in\mathbb{N}} \frac{\lambda^{2k}}{(2k)!} - \sum_{k\in\mathbb{N}} \frac{\lambda^{2k+1}}{(2k+1)!}\right) \\
&= \frac{e^{-\lambda}}{2}\left(\sum_{k\in\mathbb{N}} \frac{\lambda^{k}}{(k)!} + \sum_{k\in\mathbb{N}} \frac{(-\lambda)^{k}}{(k)!}\right) \\
&= \frac{e^{-\lambda}}{2}(e^{\lambda} + e^{-\lambda}) \\
&= \frac{1}{2}(1 + e^{-2\lambda}) \ .
\end{aligned}
$$

◀

A Simple Algorithm for Approximating the Text-To-Pattern Hamming Distance[*]

Tsvi Kopelowitz[1] and Ely Porat[2]

1 **University of Waterloo, Waterloo, Canada**
`kopelot@gmail.com`

2 **University of Bar-Ilan, Ramat Gan, Israel**
`porately@cs.biu.ac.il`

Abstract

The algorithmic task of computing the *Hamming distance* between a given pattern of length m and each location in a text of length n, both over a general alphabet Σ, is one of the most fundamental algorithmic tasks in string algorithms. The fastest known runtime for exact computation is $\tilde{O}(n\sqrt{m})$. We recently introduced a complicated randomized algorithm for obtaining a $1 \pm \epsilon$ approximation for each location in the text in $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon} \log n \log m \log |\Sigma|)$ total time, breaking a barrier that stood for 22 years. In this paper, we introduce an elementary and simple randomized algorithm that takes $O(\frac{n}{\epsilon} \log n \log m)$ time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Pattern Matching, Hamming Distance, Approximation Algorithms

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.10

1 Introduction

One of the most fundamental family of problems in string algorithms is to compute the distance between a given pattern P of length m and each location in a given larger text T of length n, both over alphabet Σ, under some string distance metric (See [24, 20, 2, 25, 8, 6, 3, 7, 29, 12, 28, 26, 9, 11, 31, 27, 19, 10, 15, 18, 17, 16, 5, 4, 30]). One of the most useful distance metrics in this setting is the *Hamming Distance* of two strings, which is the number of aligned character mismatches between the strings. Let $\texttt{HAM}(X, Y)$ denote the Hamming distance of two strings X and Y. Abrahamson [1] showed an algorithm whose runtime is $O(n\sqrt{m \log m})$. The task of obtaining a faster upper bound seems to be very challenging, and indeed there is a folklore matching conditional lower bound for combinatorial algorithms based on the hardness of combinatorial Boolean matrix multiplication (see [14]). However, for constant sized alphabets the runtime is solvable in $O(n \log m)$ using a constant number of convolution computations (which are implemented via the FFT algorithm) [20].

The challenge in beating Abrahamson's algorithm naturally lead to approximation algorithms for computing the Hamming distance in this setting, which is the problem that we consider here and is defined as follows. Denote $T_j = T[j, \ldots, j + m - 1]$. In the *pattern-to-text approximate Hamming distance problem* the input is a parameter $\epsilon > 0$, T, and P. The goal is to compute for all locations $i \in [1, n - m + 1]$ a value δ_i such that $(1-\epsilon)\texttt{HAM}(T_i, P) \leq \delta_i \leq (1+\epsilon)\texttt{HAM}(T_i, P)$. For simplicity we assume without loss of generality that Σ is the set of integers $\{1, 2, \ldots, |\Sigma|\}$.

[*] This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 683064).

Karloff in [22] utilized the efficiency of the algorithm for constant sized alphabets to introduce a beautiful randomized algorithm for solving the pattern-to-text approximate Hamming distance problem, by utilizing projections of Σ to binary alphabets. Karloff's algorithm runs in $\tilde{O}(\frac{n}{\epsilon^2})$ time, and is correct with high probability.

Communication complexity lower bounds

One of the downsides of Karloff's algorithm is the dependence on $\frac{1}{\epsilon^2}$. In particular, if one is interested in a one percent approximation guarantee, then this term becomes 10000, which is extremely large for many applications. Remarkably, many believed that beating the runtime of Karloff's algorithm is not possible, mainly since there exist qualitatively related lower bounds for estimating the Hamming distance of two equal length strings (for a single alignment). In particular, Woodruff [32] and later Jayram, Kumar and Sivakumar [21] showed that obtaining a $(1 \pm \epsilon)$ approximation for two strings in the one-way communication complexity model requires sending $\Omega(1/\epsilon^2)$ bits of information. The lower bounds were extended to the two-way communication complexity model as well [13].

In [23] we showed that **this belief was flawed**, by introducing an $\tilde{O}(\frac{n}{\epsilon})$ time algorithm that succeeds with high probability. In particular, we proved the following.

▶ **Theorem 1** ([23]). *There exists an algorithm that with high probability solves the pattern-to-text approximate Hamming distance problem and runs in $O(\frac{n}{\epsilon}\log\frac{1}{\epsilon}\log n \log m \log|\Sigma|)$ time.*

Our algorithm in [23] turned out to be rather complicated and borrows ideas from sparse recovery and constructions of specialized families of hash functions.

A simpler algorithm

In this paper we show how to solve the pattern-to-text approximate Hamming distance problem faster (in terms of logarithmic factors) and simpler. The rest of this paper is devoted to proving the following theorem.

▶ **Theorem 2** ([23]). *There exists an algorithm that with high probability solves the pattern-to-text approximate Hamming distance problem and runs in $O(\frac{n}{\epsilon}\log n \log m)$ time.*

2 The Algorithm

For a function $h : \Sigma \to \Sigma'$ and for any string $S = s_1 s_2 \ldots s_k$, let $h(S) = h(s_1)h(s_2)\ldots h(s_k)$.

Local versus global operations

The operations that our algorithm performs during the approximation of the Hamming distance at some location j are partitioned into two types. The first type is *local* operations which are independent of the computations performed for other locations in T. The second type is *global* operations, which are operations that for efficiency purposes are computed as a batch for all of the alignments in T. In particular, all of the global operations in our algorithm are to compute $HAM(h(T_j), h(P))$ where $h : \Sigma \to \left[\frac{2}{\epsilon}\right]$. Such a computation will make use of the following Theorem (which uses the FFT algorithm; see [20]), and is summarized in Corollary 4.

Algorithm 1 The new algorithm.

```
ApproxHAM($T_j, P, \epsilon$)
1  for $i = 1$ to $c \log n$
2          do Pick a random $h : \Sigma \to \{1, 2, \ldots, \frac{2}{\epsilon}\}$.
3             compute $x_i =$ HAM$(h(T_j), h(P))$
4  return $\max_{1 \le i \le c \log n}\{x_i\}$
```

▶ **Theorem 3.** *Given a binary text T of size n and a binary pattern P of size m, there exists an $O(n \log m)$ time algorithm that computes for all locations j in T the number of times that a 1 in T_j is aligned with a 1 in P.*

▶ **Corollary 4.** *Given a text T of size n and a pattern P of size m both over alphabet Σ, there exists an $O(|\Sigma| \cdot n \log m)$ time algorithm that computes $HAM(T_j, P)$ for all locations j in T.*

The algorithm for Corollary 4 is implemented by considering a separate binary text and binary pattern for each character $\sigma \in \Sigma$. For character σ in this set, occurrences of σ in T are assigned to 1, while occurrences of other characters are assigned to 0. On the other hand, occurrences of σ in P are assigned to 0, while occurrences of other characters are assigned to 1. Applying Theorem 3 on the binary text and pattern defined by σ enables computing for every location j the number of times character σ in T_j contributes to HAM(T_j, P). A summation over all the mismatches for all the characters in Σ completes the computation of HAM(T_j, P). Since Theorem 3 is applied $|\Sigma|$ times, the Corollary follows. Notice that when the algorithm of Corollary 4 is applied, then each location in the text is charged an $O(|\Sigma| \log m)$ (global) time cost.

The algorithm

With the goal of easing the presentation of our algorithm, we focus on estimating the Hamming distance between T_j and P, and count the cost of global and local operations for this location. Since we are interested in algorithms that succeed with high probability (at least $1 - \frac{1}{n^{\Theta(1)}}$) then it suffices to show that with high probability the algorithm succeeds at location j. The pseudo-code for the algorithm is given in Algorithm 1.

Time complexity

Computing the Hamming distance between the projected text and projected pattern in Line 3 takes place by applying the algorithm from Corollary 4 where the alphabet is $\left[\frac{2}{\epsilon}\right]$. Thus, the total time cost for location j is $O(\frac{1}{\epsilon} \log n \log m)$, and so the overall time cost for all locations is $O(\frac{n}{\epsilon} \log n \log m)$.

Correctness

Let $d = HAM(T_j, P)$. The goal of the algorithm is to approximate d. Notice that the expected value of x_i is $E[x_i] = (1 - \frac{\epsilon}{2})d$, since each mismatch in the original text and pattern remains a mismatch after the projection obtained by applying h with probability $1 - \frac{\epsilon}{2}$. Thus,

$E[d - x_i] = \frac{\epsilon d}{2}$, and so by the Markov inequality,

$$\Pr[x_i < (1-\epsilon)d] = \Pr[d - x_i > \epsilon d] \leq \frac{E[d - x_i]}{\epsilon d} = \frac{1}{2}.$$

Since the algorithm returns the largest x_i value, the only way in which the algorithm fails is if all of the x_i values are less than $(1-\epsilon)d$. Since the choices of the projections is independent, this happens with probability at most n^{-c}.

References

1 K. Abrahamson. Generalized string matching. In *SIAM J. Computing 16 (6)*, page 1039–1051, 1987.

2 A. Amir, O. Lipsky, E. Porat, and J. Umanski. Approximate matching in the l1 metric. In *CPM*, pages 91–103, 2005.

3 Amihood Amir, Yonatan Aumann, Gary Benson, Avivit Levy, Ohad Lipsky, Ely Porat, Steven Skiena, and Uzi Vishne. Pattern matching with address errors: rearrangement distances. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1221–1229, 2006.

4 Amihood Amir, Yonatan Aumann, Piotr Indyk, Avivit Levy, and Ely Porat. Efficient computations of l_1 and l_{infinity} rearrangement distances. In *String Processing and Information Retrieval, 14th International Symposium, SPIRE 2007, Santiago, Chile, October 29-31, 2007, Proceedings*, pages 39–49, 2007.

5 Amihood Amir, Yonatan Aumann, Oren Kapah, Avivit Levy, and Ely Porat. Approximate string matching with address bit errors. In *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008, Pisa, Italy, June 18-20, 2008, Proceedings*, pages 118–129, 2008.

6 Amihood Amir, Estrella Eisenberg, and Ely Porat. Swap and mismatch edit distance. In *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, pages 16–27, 2004.

7 Amihood Amir, Tzvika Hartman, Oren Kapah, Avivit Levy, and Ely Porat. On the cost of interchange rearrangement in strings. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 99–110, 2007.

8 Amihood Amir, Moshe Lewenstein, and Ely Porat. Approximate swapped matching. In *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS 2000 New Delhi, India, December 13-15, 2000, Proceedings.*, pages 302–311, 2000.

9 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386, 2010.

10 A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Accepted to 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

11 Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 550–559, 2004.

12 Ayelet Butman, Noa Lewenstein, Benny Porat, and Ely Porat. Jump-matching with errors. In *String Processing and Information Retrieval, 14th International Symposium, SPIRE 2007, Santiago, Chile, October 29-31, 2007, Proceedings*, pages 98–106, 2007.

13 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012.

14 Raphael Clifford. Matrix multiplication and pattern matching under hamming norm. `http://www.cs.bris.ac.uk/Research/Algorithms/events/BAD09/BAD09/Talks/BAD09-Hammingnotes.pdf`. Retrieved August 2015.

15 Raphaël Clifford, Klim Efremenko, Benny Porat, Ely Porat, and Amir Rothschild. Mismatch sampling. *Information and Computation*, 214:112–118, 2012.

16 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 778–784, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496855`.

17 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *Journal of Computer System Science*, 76(2):115–124, 2010.

18 Raphaël Clifford and Ely Porat. A filtering algorithm for k-mismatch with don't cares. *Inf. Process. Lett.*, 110(22):1021–1025, 2010. `doi:10.1016/j.ipl.2010.08.012`.

19 Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 667–676, 2002.

20 M.J. Fischer and M.S. Paterson. String matching and other products. r.m. karp (ed.), complexity of computation. In *SIAM–AMS Proceedings, vol. 7,*, page 113–125, 1974.

21 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

22 H. Karloff. Fast algorithms for approximately counting mismatches. In *Inf. Process. Lett. 48 (2)*, pages 53–60, 1993.

23 Tsvi Kopelowitz and Ely Porat. Breaking the variance: Approximating the hamming distance in $1/\epsilon$ time per alignment. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS*, pages 601–613, 2015.

24 Vladimir Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. In *Probl. Inf. Transmission 1*, page 8–17, 1965.

25 O. Lipsky and E. Porat. Approximated pattern matching with the l1, l2 and linfinit metrics. In *SPIRE*, pages 212–223, 2008.

26 R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *J. of the ACM*, pages 177–183, 1975.

27 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 315–323, 2009.

28 Benny Porat, Ely Porat, and Asaf Zur. Pattern matching with pair correlation distance. In *String Processing and Information Retrieval, 15th International Symposium, SPIRE 2008, Melbourne, Australia, November 10-12, 2008. Proceedings*, pages 249–256, 2008.

29 Ely Porat and Klim Efremenko. Approximating general metric distances between a pattern and a text. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 419–427, 2008.

30 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 173–182, 2007.

31 Ariel Shiftan and Ely Porat. Set intersection and sequence matching. In *String Processing and Information Retrieval, 16th International Symposium, SPIRE 2009, Saariselkä, Finland, August 25-27, 2009, Proceedings*, pages 285–294, 2009.

32 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 167–175, 2004.

Compact LP Relaxations for Allocation Problems*

Klaus Jansen[1] **and Lars Rohwedder**[2]

1 **Christian-Albrechts-Universität, Kiel, Germany**
`kj@informatik.uni-kiel.de`

2 **Christian-Albrechts-Universität, Kiel, Germany**
`lro@informatik.uni-kiel.de`

Abstract

We consider the restricted versions of Scheduling on Unrelated Machines and the Santa Claus problem. In these problems we are given a set of jobs and a set of machines. Every job j has a size p_j and a set of allowed machines $\Gamma(j)$, i.e., it can only be assigned to those machines. In the first problem, the objective is to minimize the maximum load among all machines; in the latter problem it is to maximize the minimum load. For these problems, the strongest LP relaxation known is the configuration LP. The configuration LP has an exponential number of variables and it cannot be solved exactly unless P = NP.

Our main result is a new LP relaxation for these problems. This LP has only $O(n^3)$ variables and constraints. It is a further relaxation of the configuration LP, but it obeys the best bounds known for its integrality gap (11/6 and 4).

For the configuration LP these bounds were obtained using two local search algorithm. These algorithms, however, differ significantly in presentation. In this paper, we give a meta algorithm based on the local search ideas. With an instantiation for each objective function, we prove the bounds for the new compact LP relaxation (in particular, for the configuration LP). This way, we bring out many analogies between the two proofs, which were not apparent before.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Linear programming, unrelated machines, makespan, max-min, restricted assignment, santa claus

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.11

1 Introduction

We consider the problem of allocating jobs $\mathcal{J}$ to machines $\mathcal{M}$. A popular variation is the restricted case, where $j \in \mathcal{J}$ has a size p_j and can only be assigned to $\Gamma(j) \subseteq \mathcal{M}$. Two natural objective functions are to minimize the maximum load or to maximize the minimum load among all machines, where the load of a machine is defined as the sum of the sizes over the jobs assigned to it. The first objective will be referred to as Makespan and the latter as Max-min. These problems are special cases of Scheduling on Unrelated Machines and the Santa Claus problem.

Recent breakthroughs in both problems can be attributed to the study of the exponential size configuration LP which started with [4]. It was shown for the Max-Min problem that the LP has an integrality gap of at most 4 [3], which was the first constant factor guarantee there. Later, Svensson transferred these ideas to the Makespan problem and proved an upper bound of 33/17 for the integrality gap in this case [11], thereby giving the first improvement over

* Research was supported by German Research Foundation (DFG) project JA 612/15-2

the classical 2-approximation (see [9]). This has since been improved to 11/6 by us [7]. The known lower bounds for the approximation ratio assuming $P \neq NP$ are are 3/2 (Makespan) [9] and 2 (Max-min) [5]. This matches the known instances with the highest integrality gap for the configuration LP. Such instances can be derived by easy modification of the lower bounds given in this paper later on. Note that all of the upper bounds mentioned above are non-constructive, i.e., they do not give an efficient algorithm to compute the integral solution of the respective quality. A significant amount of research has gone into making these proofs constructive [10, 2, 8, 1], but this is not the focus of this paper.

In this paper, we show that these upper bounds can already be achieved by a weaker LP relaxation, which has polynomial size. First, we show a necessary condition for the existence of a fractional solution of a certain value. Then, we use this condition together with a local search algorithm similar to those used to prove the bounds for the configuration LP. This local search algorithm might not terminate in polynomial time; hence the result is non-constructive. We present the local search algorithms and their analysis in a unified way for both problems. In previous literature, many similarities between the problems were hidden by the different presentations of the algorithms. The algorithm in this paper is given in a natural way like in some physical system and uses much less technical definitions than in the previous papers. Starting with an arbitrary allocation, jobs are repelled or attracted by certain machines. This depends for example on whether a machine has too much or too little load. Based on these rules, they are moved away from machines by which they are repelled or towards machines by which they are attracted. The allocation eventually converges to one that has the desired properties. This is also significant change in the technical aspects, in particular compared to the previous algorithm for Max-min. There, jobs were always considered in large sets and such a set can only be exchanged altogether for a disjoint set of jobs. Our approach is more fine-grained by arguing over the jobs individually, which is also how it was traditionally done in the Makespan case.

One advantage of the small linear program is that it is much simpler to solve. An optimal solution can be computed directly and efficiently by an LP solver. The configuration LP on the other hand cannot be solved exactly in polynomial time, unless P=NP (see Appendix A), and even approximating it requires non-trivial techniques. A detailed description can be found in [4].

Furthermore, this paper improves the understanding of the configuration LP by pointing out which properties are necessary and which are not to obtain the currently known bounds for the integrality gap. This points to aspects of the configuration LP that should be investigated in order to reduce the bound on the integrality gap further. The compact linear program we give in this paper works by enforcing some properties of the configuration LP only on jobs greater than a certain threshold. It is intuitive that the integrality gap approaches that of the configuration LP as this threshold tends to 0, but it is surprising that we get the exact same bounds and this even with a rather large threshold.

It is also a direction of further research to investigate if efficient rounding procedures for the weaker linear program exist, since now it is clear that it has the potential for them.

Notation

Throughout the paper we will encounter numerous occasions, where one inequality (e.g., $a \leq b$) holds for Makespan objective and the opposite holds for Min-max (e.g., $a \geq b$). To save space, we will write $a \leq (\geq)\ b$ in that case. The first symbol always refers to the Makespan objective and the latter one to Max-min.

For a set of jobs $A \subseteq \mathcal{J}$, we write $p(A)$ in place of $\sum_{j \in A} p_j$. For other variables indexed by jobs, we may do the same. An allocation is a function $\sigma : \mathcal{J} \to \mathcal{M}$, where $\sigma(j) \in \Gamma(j)$ for all $j \in \mathcal{J}$. We write $\sigma^{-1}(i)$ for the set of all jobs j which have $\sigma(j) = i$.

1.1 Linear programming relaxations

All of the LPs presented below do not have an objective function. Instead, they are parameterized by a value $T \in [0, n \cdot \max_{j \in \mathcal{J}} p_j]$ and the optimum is the lowest T (Makespan) or highest T (Max-min) for which the LP is feasible. If the LP can be solved in polynomial time, such a T can be found in polynomial time using a binary search.

First, we define the allocation polytope, which captures every legal (fractional) allocation of jobs.

▶ **Definition 1** (Allocation polytope).

$$\sum_{i \in \Gamma(j)} x_{i,j} \geq (\leq)\ 1 \qquad \forall j \in \mathcal{J} \qquad (1)$$

$$\sum_{i \notin \Gamma(j)} x_{i,j} = 0 \qquad \forall j \in \mathcal{J} \qquad (2)$$

$$x_{i,j} \in [0,1] \qquad \forall j \in \mathcal{J}, i \in \mathcal{M}$$

Here the variable $x_{i,j}$ specifies if job j is assigned to machine i. Note that it would perhaps be more intuitive to enforce equality in (1), but this would make certain upcoming arguments more lengthy than necessary. In general, a solution that does not satisfy equality can be converted without loss to one that does.

It remains to add constraints that guarantee every machine has a load of at most T (Makespan) or at least T (Max-min). We give these constraints in an indirect form. This is to improve comparability between these relaxations. In Section 2 we will give an explicit version of LP$_r$, which is the polynomial linear program this paper focuses on.

▶ **Definition 2** (Assignment LP). The straight forward method is to ensure for all $i \in \mathcal{M}$ that $\sum_{j \in \mathcal{J}} p_j x_{i,j} \leq (\geq)\ T$ holds. This can also be written as

$$(x_{i,j})_{j \in \mathcal{J}} \in \{\chi \in [0,1]^{\mathcal{J}} : p^T \chi \leq (\geq)\ T\} \qquad \forall i \in \mathcal{M}. \qquad (3)$$

This basic relaxation goes back to [9]. For Makespan it has an integrality gap of exactly 2; for Max-min the integrality gap is unbounded.

▶ **Definition 3** (Configuration LP). For the configuration LP it is required that the assignment of jobs to a particular machine is a convex combination of so-called configurations (sets of jobs that do not exceed T in size or have size of at least T), i.e.,

$$(x_{i,j})_{j \in \mathcal{J}} \in \operatorname{conv}\{\chi \in \{0,1\}^{\mathcal{J}} : p^T \chi \leq (\geq)\ T\} \qquad \forall i \in \mathcal{M}. \qquad (4)$$

Note that it is not necessary to require $\chi_j = 0$ for all $j \in \mathcal{J}$ with $i \notin \Gamma(j)$ (which is typical for the definition of the configurations), since this is already implied by the allocation polytope. The common definition of the configuration LP uses an exponential number of variables. Wiese and Verschae observed that the definition above is equivalent [12]. Clearly these constraints imply those from the assignment LP. Hence, the configuration LP is the stronger of the two.

▶ **Definition 4** (LP$_r$)**.** As a natural intermediate between assignment LP and configuration LP, for a constant $r \in \mathbb{N}_0$ we propose the following constraint.

$$(x_{i,j})_{j\in\mathcal{J}} \in \operatorname{conv}\{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \leq (\geq)\ T, \chi_j \in \{0,1\} \text{ if } p_j \cdot r > T\} \qquad \forall i \in \mathcal{M}. \tag{5}$$

To our best knowledge, the cases where $0 < r < \infty$ have not been considered in literature.

1.2 Other related work

The Graph Balancing problem is the special case of Makespan minimization where $|\Gamma(j)| = 2$ for all $j \in \mathcal{J}$. For this problem a strong polynomial size LP relaxation is already known. This is the assignment LP with the additional constraint that $\sum_{j\in\mathcal{J}:p_j>T/2} x_{i,j} \leq 1$ for all $i \in \mathcal{M}$. It was shown to have an integrality gap of exactly 1.75 for Graph Balancing, but for arbitrary restrictions it only gives 2 [6]. This LP can be written as the points in the allocation polytope that satisfy

$$(x_{i,j})_{j\in\mathcal{J}} \in \operatorname{conv}\{\chi \in [0,\infty)^{\mathcal{J}} : p^T\chi \leq T \text{ and } \chi_j \in \{0,1\} \text{ if } p_j \cdot 2 > T\} \qquad \forall i \in \mathcal{M}.$$

In this form we see clearly the similarities to LP$_2$. Interestingly, LP$_2$ remains strong even for arbitrary restrictions.

1.3 Our contribution

▶ **Theorem 5.** *For $r \geq 2$ there is a linear program with $O(n^{r+1})$ variables and constraints (Makespan) or $O(n^{r+2})$ (Max-min) that is equivalent to* LP$_r$*, where $n = |\mathcal{J}| + |\mathcal{M}|$.*

▶ **Theorem 6.** LP$_2$ *for Makespan has an integrality gap between* $10/6$ *and* $11/6$*.*

▶ **Theorem 7.** LP$_4$ *for Max-min has an integrality gap between* 2.5 *and* 4*.*

With some optimization in the Max-min case, we can further reduce the size of LP$_4$ (see Appendix B).

▶ **Corollary 8.** *For Makespan (Max-min) objective there is a linear programming relaxation with $O(n^3)$ variables and constraints that approximates the problem with a ratio of* $11/6$ *(respectively,* 4*).*

Notable is that the lower bounds are higher than those known for the configuration LP. There, the worst instances known give an integrality gap of 2 (Max-min) and 9/6 (Makespan). This means for LP$_2$ / LP$_4$ we are closer to a full understanding. It also shows that, assuming that the integrality gap of the configuration LP is indeed the respective lower bound, proving it will require utilizing constraints that are not already implied by LP$_2$ (Makespan) or LP$_4$ (Max-min).

▶ **Corollary 9.** *There exists an* $11/6$*-estimation (*4*-estimation) algorithm for the Makespan objective (respectively, the Max-min objective).*

The estimation algorithm is based on computing the optimum of the relaxation. This improves on the $11/6 + \epsilon$ $(4 + \epsilon)$ rate previously known. The error of ϵ in the previous result comes from the fact that the configuration LP can only be solved approximately.

2 Compact linear program

In this section, we present a polynomial size linear program, which is feasible if and only if LP$_r$ is feasible. Note that in order to meet the claimed size, we need to eliminate unnecessary variables, which is discussed in 2.1.

For simplicity of notation, define big jobs $\mathcal{J}_B := \{j \in \mathcal{J} : r \cdot p_j > T\}$ and small jobs $\mathcal{J}_S := \mathcal{J} \setminus \mathcal{J}_B$. In the first part, we write an LP for the big jobs and only then deal with the small ones. We write the set of big configurations as $C_B(T) = \{\chi \in \{0,1\}^{\mathcal{J}_B}\}$.

The convexity constraint for LP$_r$ implies that $(x_{i,j})_{j\in\mathcal{J}_B} \in \mathrm{conv}(C_B(T))$ for every $i \in \mathcal{M}$. In other words, there exist $a_{i,\chi} \geq 0$ $(i \in \mathcal{M}, \chi \in C_B(T))$ such that $\sum_{\chi\in C_B(T)} a_{i,\chi} = 1$ $(*)$ for every $i \in \mathcal{M}$ and $x_{i,j} = \sum_{\chi\in C_B(T)} \chi_j a_{i,\chi}$ for every $i \in \mathcal{M}, j \in \mathcal{J}$. With this idea in mind, we construct an LP by using variables $a_{i,\chi}$, the constraint $(*)$, and the allocation LP where we substitute every occurrence of $x_{i,j}$ for $\sum_{\chi\in C_B(T)} \chi_j a_{i,\chi}$.

$$\sum_{\chi\in C_B(T)} a_{i,\chi} = 1 \qquad \forall i \in \mathcal{M} \tag{6}$$

$$\sum_{i\in\Gamma(j)} \sum_{\chi\in C_B(T)} \chi_j a_{i,\chi} \geq (\leq)\ 1 \qquad \forall j \in \mathcal{J}_B \tag{7}$$

$$\sum_{i\notin\Gamma(j)} \sum_{\chi\in C_B(T)} \chi_j a_{i,\chi} = 0 \qquad \forall j \in \mathcal{J}_B \tag{8}$$

$$a_{i,\chi} \geq 0$$

In the following, we will show how to cope with small jobs. For every $j \in \mathcal{J}_S$, $i \in \mathcal{M}$, and $\chi \in C_B(T)$ we use a variable $b_{j,i,\chi}$ that describes how much of j is used on machine i together with χ. Here $b_{j,i,\chi} = a_{i,\chi}$ means it is fully used and $b_{j,i,\chi} = 0$ means it is not used at all.

$$\sum_{i\in\Gamma(j)} \sum_{\chi\in C_B(T)} b_{j,i,\chi} \geq (\leq)\ 1 \qquad \forall j \in \mathcal{J}_S \tag{9}$$

$$\sum_{i\notin\Gamma(j)} \sum_{\chi\in C_B(T)} b_{j,i,\chi} = 0 \qquad \forall j \in \mathcal{J}_S \tag{10}$$

$$\sum_{j\in\mathcal{J}_S} p_j b_{j,i,\chi} \leq (\geq)\ (T - \sum_{j\in\mathcal{J}_B} p_j \chi_j) a_{i,\chi} \qquad \forall i \in \mathcal{M}, \chi \in C_B(T) \tag{11}$$

$$0 \leq b_{j,i,\chi} \leq a_{i,\chi}$$

2.1 Restricting the variables

We denote by supp(χ) the non-zero components of $\chi \in C_B(T)$. Observe that in the makespan case, a configuration $\chi \in C_B(T)$ with $|\mathrm{supp}(\chi)| \geq r$ cannot be used, i.e., $a_{i,\chi} = 0$ must hold for a feasible solution. Otherwise, the right hand side of (11) is negative. Hence, we can throw away such variables and the number of remaining configurations is at most $\sum_{k=0}^{r-1} \binom{n}{k} = O(n^{r-1})$. It is easy to see that $O(n^{r+1})$ variables and constraints are left.

For Max-min, we notice that when a configuration $\chi \in C_B(T)$ has $|\mathrm{supp}(\chi)| \geq r$, then (11) is trivially satisfied. If a configuration $\chi \in C_B(T)$ with $|\mathrm{supp}(\chi)| > r$ is used, then we can shift its value to any $\chi' \leq \chi$ (component-wise) where $|\mathrm{supp}(\chi)| = r$. Hence, if there is a feasible solution, then there is also one that uses only configurations $\chi \in C_B(T)$ with $|\mathrm{supp}(\chi)| \leq r$. This gives a total of $O(n^r)$ relevant configurations and thus $O(n^{r+2})$ variables and constraints.

2.2 Equivalence to LP_r

▶ **Lemma 10.** *If the compact linear program is feasible, then* LP_r *is feasible.*

Proof. Consider a feasible solution a, b. Recall, we have that each machine is assigned a combination of configurations in $C_B(T)$. We will extend these configurations by adding small jobs to them. For this purpose, define for every $i \in \mathcal{M}$ and $\chi \in C_B(T)$ a vector $f(i,\chi) \in [0,1]^{\mathcal{J}}$ by

$$f(i,\chi)_j := \begin{cases} \chi_j & \text{if } j \in \mathcal{J}_B, \\ b_{j,i,\chi}/a_{i,\chi} & \text{if } j \in \mathcal{J}_S \text{ and } a_{i,\chi} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Note that since $b_{j,i,\chi} \le a_{i,\chi}$, we have $f(i,\chi)_j \in [0,1]$. We define the solution x for LP_r on every machine as a convex combination of these vectors, more formally

$$x_{i,j} := \sum_{\chi \in C_r(T)} a_{i,\chi} \cdot f(i,\chi)_j.$$

It follow directly from the constraints in the compact linear program that x is in the allocation polytope. Let us verify that x satisfies the convexity constraint for i. Since $\sum_{\chi \in C_r(T)} a_{i,\chi} = 1$, it is sufficient to show that for every $i \in \mathcal{M}, \chi \in C_B(T)$ with $a_{i,\chi} > 0$,

$$f(i,\chi) \in \{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \le (\ge)\ T \text{ and } \chi_j \in \{0,1\} \text{ if } p_j \cdot r > T\}.$$

By definition we have $f(i,\chi)_j \in \{0,1\}$ if $p_j \cdot r > T$ (i.e., $j \in \mathcal{J}_B$) and using constraint (11) we find,

$$p^T f(i,\chi) = \sum_{j \in \mathcal{J}_B} p_j\chi_j + \sum_{j \in \mathcal{J}_S} p_j b_{j,i,\chi}/a_{i,\chi} \le (\ge) \sum_{j \in \mathcal{J}_B} p_j\chi_j + (T - \sum_{j \in \mathcal{J}_B} p_j\chi_j) a_{i,\chi}/a_{i,\chi} = T. \blacktriangleleft$$

▶ **Lemma 11.** *If* LP_r *is feasible, then the compact linear program is feasible.*

Proof. Let x be a solution for LP_r and define

$$K = \{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \le (\ge)\ T, \chi_j \in \{0,1\} \text{ if } p_j \cdot r > T\}.$$

Let $i \in \mathcal{M}$. Then by the convexity constraint, there exist $(\lambda_\chi)_{\chi \in K}$ non-negative with $\sum_{\chi \in K} \lambda_\chi = 1$ and $x_{i,j} = \sum_{\chi \in K} \lambda_\chi \chi_j$ for all $j \in \mathcal{J}$.

Let $\psi \in C_B(T)$ and let $K(\psi) \subseteq K$ denote those $\chi \in K$ which have $\chi_j = \psi_j$ for all $j \in \mathcal{J}_B$. We define the variables for i and ψ as $a_{i,\psi} = \sum_{\chi \in K(\psi)} \lambda_\chi$ and $b_{i,j,\psi} = \sum_{\chi \in K(\psi)} \lambda_\chi \chi_j$ for all $j \in \mathcal{J}_S$. Note that

$$\sum_{\psi \in C_B(T)} b_{i,j,\psi} = \sum_{\psi \in C_B(T)} \sum_{\chi \in K(\psi)} \lambda_\chi \chi_j = \sum_{\chi \in K} \lambda_\chi \chi_j = x_{i,j}.$$

With that in mind, the constraints except for (11) are straight-forward. Moreover, we show said constraint as follows.

$$\begin{aligned} \sum_{j \in \mathcal{J}_B} p_j\psi_j + \frac{1}{a_{i,\psi}} \sum_{j \in \mathcal{J}_S} p_j b_{j,i,\psi} &= \sum_{j \in \mathcal{J}_B} p_j\psi_j + \frac{1}{a_{i,\psi}} \sum_{j \in \mathcal{J}_S} [p_j \sum_{\chi \in K(\psi)} \lambda_\chi \chi_j] \\ &= \underbrace{\sum_{\chi \in K(\psi)} [\frac{\lambda_\chi}{a_{i,\psi}}}_{=1} \sum_{j \in \mathcal{J}_B} p_j \underbrace{\chi_j}_{=\psi_j}] + \sum_{\chi \in K(\psi)} [\frac{\lambda_\chi}{a_{i,\psi}} \sum_{j \in \mathcal{J}_S} p_j\chi_j] \\ &= \sum_{\chi \in K(\psi)} \frac{\lambda_\chi}{a_{i,\psi}} p^T\chi \le (\ge)\ T. \end{aligned}$$

◀

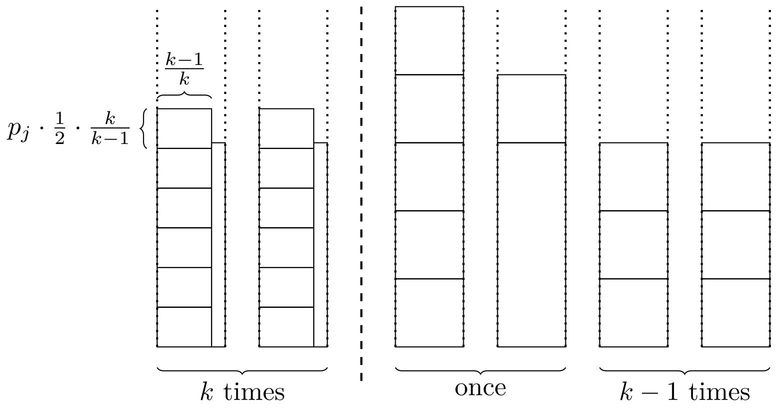

Figure 1 Fractional and integral solution for lower bound (Makespan)

3 Lower bound (Makespan)

Here, we give a lower bound of 5/3 for LP_3 (in particular, for the weaker LP_2) in the Makespan case. A similar construction for the Max-min case is given in the appendix.

Let k be an even number and consider an instance with k machines, i.e., $k/2$ pairs of machines, and $3k+1$ jobs. For each of the $k/2$ pairs (i_1, i_2) let there be 6 jobs j with $p_j = 1/3$ and $\Gamma(j) = \{i_1, i_2\}$. Furthermore let there be one job j_B with $p_{j_B} = 1$ and $\Gamma(j_B) = \mathcal{M}$, i.e., it can be assigned anywhere.

Assume toward contradiction that there is a schedule with makespan strictly less than 5/3. j_B has to be assigned somewhere and we denote this machine by i. There can be at most one job of size 1/3 that is assigned to i as well. Hence, 5 jobs of size 1/3 must be assigned to the other machine in this pair; thus the load on that machine is at least 5/3. A contradiction.

Next, we show that LP_2 is feasible for $T = k/(k-1)$. For every $i \in \mathcal{M}$ let $x_{i,j_B} = 1/k$, i.e., the big job is distributed evenly across all machines. For every other job j, split it across the two machines it is allowed on. More formally, let $x_{i_1,j} = x_{i_2,j} = 1/2$ with $\{i_1, i_2\} = \Gamma(j)$. Clearly x is in the allocation polytope. Let $i \in \mathcal{M}$. We need to verify that

$$(x_{i,j})_{j\in\mathcal{J}} \in \mathrm{conv}\{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \le k/(k-1) \text{ and } \chi_j \in \{0,1\} \text{ if } p_j \cdot 3 > k/(k-1)\}.$$

We define one vector for the big job and one for each machine:

$$\chi_j^{(j_B)} := \begin{cases} 1 & \text{if } j = j_B, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and } \chi_j^{(i)} := \begin{cases} \frac{1}{2}\cdot\frac{k}{k-1} & \text{if } p_j = 1/3 \text{ and } i \in \Gamma(j), \\ 0 & \text{otherwise.} \end{cases}$$

Note that we have $p^T\chi^{(j_B)} = p_{j_B} = 1 \le \frac{k}{k-1} = T$ as well as $p^T\chi^{(i)} = 6\cdot\frac{1}{3}\cdot\frac{1}{2}\cdot\frac{k}{k-1} = \frac{k}{k-1} = T$. The integrality constraint is satisfied for j_B and since $1/3 \cdot 3 = 1 \le k/(k-1)$, it is not necessary for the small jobs. Also, it holds that $(x_{i,j})_{j\in\mathcal{J}} = 1/k \cdot \chi^{(j_B)} + (k-1)/k \cdot \chi^{(i)}$, as shown below.

$$x_{i,j} = \begin{cases} 1/k \cdot 1 + (1-1/k)\cdot 0 = 1/k \cdot \chi_j^{(j_B)} + (k-1)/k \cdot \chi_j^{(i)} & \text{if } j = j_B, \\ 1/k \cdot 0 + \frac{k-1}{k}\cdot\frac{1}{2}\cdot\frac{k}{k-1} = 1/k \cdot \chi_j^{(j_B)} + (k-1)/k \cdot \chi_j^{(i)} & \text{if } p_j = 1/3 \text{ and } i \in \Gamma(j), \\ 1/k \cdot 0 + (1-1/k)\cdot 0 = 1/k \cdot \chi_j^{(j_B)} + (k-1)/k \cdot \chi_j^{(i)} & \text{otherwise.} \end{cases}$$

For this instance, we get a gap that approaches 5/3 as k tends to infinity. This construction does not work for LP_4 and higher, since in those cases integrality constraints are enforced for the jobs of size 1/3 as well. However, similar constructions work when using 10 jobs of size 1/5, 14 jobs of size 1/7, etc. The lower bound becomes weaker the smaller the size is chosen, but it always stays strictly above 3/2, the best lower bound for the configuration LP.

4 Proof of integrality gap

The proof for the integrality gap is by using a potentially exponential time approximation algorithm. The algorithm takes as an input T and returns a solution of value αT, where α is the bound on the integrality gap, or proves that LP$_2$ (Makespan) or LP$_4$ (Max-min) is infeasible w.r.t. T.

In 4.1, we prove a criterion for the infeasibility of LP$_r$. In previous literature, for the configuration LP a similar criterion was derived from its dual. In fact, if r tends to infinity (i.e., $p_j \cdot r > T$ for all $j \in \mathcal{J}$), our criterion is equivalent to that one.

In the proofs for the configuration LP, our criterion can replace the previous one in a straight-forward way and give the integrality gap for LP$_2$ (Makespan) or LP$_4$ (Max-min).

4.1 Criterion for infeasibility

The criterion is derived from the LP$_r$ in another equivalent representation and by using the duality theorem (e.g., unbounded dual implies infeasible primal). For this purpose, we construct a representation of LP$_r$, where we do not care about its size, but the goal is to obtain a rather simple dual.

▶ **Lemma 12.** LP$_r$ *is infeasible w.r.t.* T *if there are* $y \in \mathbb{R}^{\mathcal{M}}_{\geq 0}$ *and* $z \in \mathbb{R}^{\mathcal{J}}_{\geq 0}$ *with* $\sum_{j\in\mathcal{J}} z_j > (<) \sum_{i\in\mathcal{M}} y_i$ *such that for every* $i \in \mathcal{M}$, $\chi \in [0,1]^{\mathcal{J}}$ *with*

1. $p^T\chi \leq (\geq)\ T$,
2. $\chi_j \in \{0,1\}$ *for every* $j \in \mathcal{J}$ *with* $p_j \cdot r > T$, *and*
3. $\chi_j = 0$ *for every* $j \in \mathcal{J}$ *with* $i \notin \Gamma(j)$,

it holds that $z^T\chi \leq (\geq)\ y_i$.

For this lemma even equivalence holds. To conserve space we will only show this direction.

Proof of Lemma 12. Recall for the compact linear program, the big jobs were assigned to machines in configurations. We want to include the (fractional) allocation of small jobs in those configurations as well. The natural approach is to define

$$C(T) := \{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \leq (\geq)\ T, \chi_j \in \{0,1\} \text{ if } p_j \cdot r > T\}.$$

Then a representation of LP$_r$ is the following.

$$\min(\max)\ 0 \tag{12}$$

$$\sum_{\chi\in C(T)} a_{\chi,i} = 1 \qquad \forall i \in \mathcal{M} \tag{13}$$

$$\sum_{i\in\Gamma(j)} \sum_{\chi\in C(T)} \chi_j \cdot a_{\chi,i} \geq (\leq)\ 1 \qquad \forall j \in \mathcal{J} \tag{14}$$

$$\sum_{i\notin\Gamma(j)} \sum_{\chi\in C(T)} \chi_j \cdot a_{\chi,i} = 0 \qquad \forall j \in \mathcal{J} \tag{15}$$

$$a_{\chi,i} \geq 0$$

There is, however, an issue with this definition. Since $C(T)$ can have infinitely many elements, the dimension of the LP is potentially infinite. This means, we cannot simply apply results from LP duality. Thus, we will first show that a finite number of variables is suffices.

Recall that the constraint for LP_r is $(x_{i,j})_{j\in\mathcal{J}} \in \mathrm{conv}(C(T))$. We will show that $\mathrm{conv}(C(T)) = \mathrm{conv}(V(T))$ for some finite $V(T) \subseteq C(T)$. This means, we can substitute $C(T)$ for $V(T)$.

Observe that $C(T)$ is a union of polytopes, where each polytope corresponds to one integral allocation of the big jobs. More formally, let $\chi^B \in \{0,1\}^{\mathcal{J}_B}$. Then the set of vectors in $C(T)$ where the values for big jobs equal χ^B are exactly

$$\{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \le (\ge)\ T \text{ and } (\chi_j)_{j\in\mathcal{J}_B} = \chi^B\}.$$

For each of these χ^B, this is clearly a polytope, which can be written as the convex hull of finitely many basic solutions. Let $V(T)$ be the set of all basic solutions for all allocations χ_B of big jobs. Then $(x_{i,j})_{j\in\mathcal{J}} \in \mathrm{conv}(C(T))$ is equivalent to $(x_{i,j})_{j\in\mathcal{J}} \in \mathrm{conv}(V(T))$

We substitute $C(T)$ for $V(T)$ in the LP above and, for an easier dual, we multiply (13) by -1. Then the dual is the following:

$$\max(\min) \sum_{j\in\mathcal{J}} z_j - \sum_{i\in\mathcal{M}} y_i$$

$$\sum_{j\in\mathcal{J}:i\in\Gamma(j)} \chi_j \cdot z_j + \sum_{j\in\mathcal{J}:i\notin\Gamma(j)} \chi_j \cdot \overline{z}_j \le (\ge)\ y_i \quad \forall i \in \mathcal{M}, \chi \in V(T)$$

$$z_j \ge 0$$

$$\overline{z}_j, y_i \in \mathbb{R}$$

Now consider the values z, y from Lemma 12 and set $\overline{z}_j$ to a negative (Makespan) or positive (Max-min) number of very large magnitude for all $j \in \mathcal{J}$. Then this is a feasible solution for the dual: Let $i \in \mathcal{M}$ and $\chi \in V(T)$. If $\chi_j = 0$ for all $j \in \mathcal{J}$ with $i \notin \Gamma(j)$, then the constraint is satisfied by definition of z and y. Otherwise, the constraint holds when $\overline{z}_j$ is chosen sufficiently small (large).

The solution has a positive (negative) objective value and we can scale it by any constant and construct a new feasible solution. This way we can obtain any positive (negative) objective value, i.e., the dual is unbounded. By duality this implies the primal is infeasible. ◀

4.2 Local search algorithm

▶ **Definition 13** (Good and bad machines). Given an allocation $\sigma : \mathcal{J} \to \mathcal{M}$, we call a machine i bad, if $\sum_{j\in\sigma^{-1}(i)} p_j > 11/6 \cdot T$ (Makespan) or $\sum_{j\in\sigma^{-1}(i)} p_j < 1/4 \cdot T$ (Max-min). A machine is good, if it is not bad.

The local search algorithm starts with an arbitrary allocation and moves jobs until all machines are good, or it can prove that LP_2 (Makespan) or LP_4 (Max-min) is infeasible w.r.t. T. During this process, a machine that is already good will never be made bad.

The central data structure of the algorithm is an ordered list of moves $L = (L_1, L_2, \dots, L_\ell)$. Here, every component $L_k = (j, i)$, $j \in \mathcal{J}$ and $i \in \Gamma(j)$, stands for a move the algorithm wants to perform. It will not perform the move, if this would create a bad machine, i.e., $p(\sigma^{-1}(i)) + p_j > 11/6 \cdot T$ (Makespan) or $p(\sigma^{-1}(\sigma(j))) - p_j < 1/4 \cdot T$ (Max-min). If it does not create a bad machine, we say that the move (j, i) is valid. For every $0 \le k \le \ell$ define $L_{\le k} := (L_1, \dots, L_k)$, the first k elements of L (with $L_{\le 0}$ being the empty list).

Algorithm 1 Local search meta-algorithm

```
1. Let $\sigma$ be an arbitrary allocation;
2. $\ell \leftarrow 0$; // length of the list of moves $L$
3. while there is a bad machine do
   a. if there exists a valid move $(j, i) \in L$ then
      i. Let $0 \leq k \leq \ell$ be minimal such that
         (Makespan) $j$ is repelled by $\sigma(j)$ w.r.t. $L_{\leq k}$;
         (Max-min) $j$ is attracted by $i$ w.r.t. $L_{\leq k}$;
      ii. $\sigma(j) \leftarrow i$;
      iii. $L \leftarrow L_{\leq k}$; $\ell \leftarrow k$; // Forget moves $L_{k+1}, \dots, L_\ell$
   b. else
      i. Choose a move $(j, i) \notin L$, $j \in \mathcal{J}$ and $i \in \Gamma(j)$, with $p_j$ minimal and
         (Makespan) $j$ is repelled by $\sigma(j)$ and not repelled by $i$ w.r.t. $L$;
         (Max-min) $j$ is not attracted by $\sigma(j)$ and attracted by $i$ w.r.t. $L$;
      ii. $L_{\ell+1} \leftarrow (j, i)$; $\ell = \ell + 1$; // Append $(j, i)$ to $L$
```

Depending on the current schedule σ and list of moves L, we define for every machine i which jobs are repelled or not repelled (Makespan). In the Max-min case we use the term attracted instead of not repelled; not attracted instead of repelled. This is only to make the definitions easier to read. The definition of repelled/attracted jobs differs in the two algorithms and is given in Section 4.2.1. The algorithm will only add a new move (j, i) to the current list L, if j is repelled (not attracted) by its current machine and not repelled (attracted) by the target i w.r.t. L.

4.2.1 Repelled and attracted jobs

Here, we will start with the Max-min case, since it is the simpler one.

Max-min

Depending on the current schedule σ and the current set of moves $L = L_{\leq \ell}$, we define which jobs are attracted by which machines. We call a job $j \in \mathcal{J}$ big, if $p_j > 1/4 \cdot T$ and small otherwise. The first two rules are that bad machines attract all jobs and that rules are propagated from prefixes of the list.
(initialization) If $\ell = 0$, every bad machine i attracts every job j.
(monotonicity) If $\ell > 0$ and i attracts j w.r.t. $L_{\leq \ell-1}$, then i attracts j w.r.t. $L_{\leq \ell}$.
For the remaining rules, assume $\ell > 0$, and let $(j_\ell, i_\ell) := L_\ell$ be the last move added. We will define which new rules this move adds to the existing ones.

We can assume that all moves in $L_{\leq \ell-1}$ are not valid, since otherwise the algorithm would execute them instead of adding a new one. Since i_ℓ tries to steal j_ℓ from $\sigma(j_\ell)$, but the move is not valid, the machine $\sigma(j_\ell)$ should attract jobs in order to make (j_ℓ, i_ℓ) valid. More precisely,
(small-all) if j_ℓ is small, $\sigma(j_\ell)$ attracts all jobs and
(big-big) if j_ℓ is big, $\sigma(j_\ell)$ attracts all big jobs.

This misses one important case. Suppose that j_ℓ is big. Intuitively, $\sigma(j_\ell)$, should also collect small jobs to make the move (j_ℓ, i_ℓ) valid. However, the straight-forward way ($\sigma(j)$ attracts all small jobs as well) does not work out in the analysis. Hence, a more sophisticated strategy is required.

For $i \in \mathcal{M}$ define $S_i(L)$ to be all small jobs j which have either $\sigma(j) = i$ or which have $i \in \Gamma(j)$ and are not attracted by their current machine, $\sigma(j)$, w.r.t. the rules above. The intuition behind $S_i(L)$ is the following. In the best case, i could collect all jobs from $S_i(L)$. If $p(S_i(L)) < 1/4 \cdot T$, then we cannot expect i to satisfy its demand only using small jobs. On the other hand, if it gets a big job, then this job alone satisfies its demand. Hence, i should not attract small jobs. More formally,
(big-all) if j_ℓ is big and $p(S_{\sigma(j_\ell)}(L)) \geq 1/4 \cdot T$, then $\sigma(j_\ell)$ attracts all jobs.

Makespan

Here, we define big jobs to be those $j \in \mathcal{J}$ that have $p_j > 1/2 \cdot T$ and small jobs all others.
(initialization) If $\ell = 0$, every bad machine i repels every job j.
(monotonicity) If $\ell > 0$ and i repels j w.r.t. $L_{\leq \ell-1}$, then i repels j w.r.t. $L_{\leq \ell}$.
Again, the remaining rules regard $\ell > 0$ and we define $(j_\ell, i_\ell) := L_\ell$, i.e., the last move added. In order to make space for j_ℓ, the machine i_ℓ should repel jobs.
(small-all) If j_ℓ is small, i_ℓ repels all jobs.
This still leaves one case to resolve. If j_ℓ is big, which jobs does i_ℓ repel? It helps to imagine that the algorithm is a lazy one: It repels jobs only if it is really necessary.

For $i \in \mathcal{M}$ define $S_i(L)$ to be those small jobs j which have $\sigma(j) = i$ and which are repelled by all other potential machines, i.e., $\Gamma(j) \setminus \{i\}$, w.r.t. the rules above. The intuition behind $S_i(L)$ is that we do not expect that i can get rid of any of the jobs in $S_i(L)$.

Next, define a threshold $t(L, (j_\ell, i_\ell))$ as the minimum $t \geq 0$ such that S_i and all big jobs below this threshold are already too large to add j_ℓ:

$$p(\{j \in \sigma^{-1}(i_\ell) : j \in S_{i_\ell}(L) \text{ or } 1/2 < p_j \leq t\}) + p_{j_\ell} > 11/6 \cdot T,$$

or $t(L, (j_\ell, i_\ell)) = \infty$ if no such t exists. In order to make (j_ℓ, i_ℓ) valid, it is necessary (although not always sufficient) to remove one of the big jobs with size at most $t(L, (j_\ell, i_\ell))$. Hence, we define,
(big-all) if j_ℓ is big and $t(L, (j_\ell, i_\ell)) = \infty$, then i_ℓ repels all jobs and
(big-big) if j_ℓ is big and $t(L, (j_\ell, i_\ell)) < \infty$, then i_ℓ repels $S_{i_\ell}(L)$ and all jobs j with $1/2 < p_j \leq t(L, (j_\ell, i_\ell))$.
Note that repelling $S_{i_\ell}(L)$ seems unnecessary, since those jobs do not have any machine to go to. However, this definition simplifies the analysis. It is also notable that the special case where $t(L, (j_\ell, i_\ell)) = 0$ is equivalent to $p(S_{i_\ell}(L)) + p_{j_\ell} > 11/6 \cdot T$ and here the algorithm gives up making (j_ℓ, i_ℓ) valid. Finally, we want to highlight the following counter-intuitive (but intentional) aspect of the algorithm. It might happen that some job of size greater than $t(L, (j_\ell, i_\ell))$ is moved to i_ℓ, only to be removed again later on, when $t(L, (j_\ell, i_\ell))$ has increased.

4.3 Analysis (Max-min)

For completeness, we give the analysis for the Makespan case in the appendix. It is not included here so as to avoid repetitive arguments.

To verify the correctness of the algorithm, it has to be shown that (1) it terminates and (2) in each iteration of the main loop, there is either a valid move in L or some move that can be added to L.

▶ **Theorem 14.** *The algorithm terminates after finitely many iterations of the main loop.*

Proof. As before, let ℓ denote the current length of L. We define a potential function

$$\Phi(L) = (b, s_0, s_1, s_2, \dots, s_\ell, \infty),$$

where b is the number of bad machines and s_k is the number of pairs $(i, j) \in \mathcal{M} \times \mathcal{J}$ where i attracts j w.r.t. $L_{\leq k}$ and $\sigma(j) \neq i$. Intuitively, the algorithm makes progress when this number decreases.

Note that the length of $\Phi(L)$ is bounded by $|\mathcal{M}| \cdot |\mathcal{J}| + 2$ and every component can only have $|\mathcal{M}| \cdot |\mathcal{J}| + 1$ many different values. Thus, the range of the potential function is finite.

We will show that the vector decreases lexicographically after every iteration of the main loop; hence the running time is bounded by the number of such vectors times the maximum number of operations in one iteration and, in particular, is finite. For the lexicographic decrease consider two cases. Either a new move is added, which decreases the potential function by replacing the last component with some finite value, or a move is performed. If a move turns a bad machine good, b decreases and so does the lexicographic value of $\Phi(L)$. Otherwise, let $\ell' \leq \ell$ be the length of the list after a move (j, i) is performed. Recall the algorithm prevents jobs attracted by their current machine from being moved, i.e., j is not attracted by its previous machine w.r.t. $L_{\leq \ell'}$. Moreover, observe that the attracted jobs w.r.t. $L_{\leq 0}, \dots, L_{\leq \ell'}$ do not change. This can be seen from the definition of attracted jobs. Therefore $s_0, \dots, s_{\ell'}$ do not increase. Finally, since j is attracted by i w.r.t. $L_{\leq \ell'}$ and after the move $\sigma(j) = i$ holds, the value of $s_{\ell'}$ has decreased. ◀

▶ **Theorem 15.** *If* LP_4 *is feasible w.r.t.* T*, the algorithm always has an operation it can perform.*

Proof. As in the algorithm, call a job j small, if $p_j < 1/4 \cdot T = 1/r \cdot T$ and big otherwise. Suppose toward contradiction, there are bad machines, no move in L is valid, and no move can be added to L. We will construct values $(z_j)_{j \in \mathcal{J}}$, $(y_i)_{i \in \mathcal{M}}$ with the properties as in Lemma 12 and thereby show that LP_4 is infeasible w.r.t. T.

Define $y_i = 3/4$ for every $i \in \mathcal{M}$ where i is bad or $(j', i') \in L$ for some $j' \in \mathcal{J}$ with $\sigma(j') = i$. For all other machines i define $y_i = 0$. Furthermore, define $z_j = 3/4$ if j is big and attracted by $\sigma(j)$; define $z_j = p_j/T$ if j is small and attracted by $\sigma(j)$; and $z_j = 0$ if j is not attracted by $\sigma(j)$. We proceed to show that these values indeed satisfy the properties as in Lemma 12.

▶ **Fact 16.** *Let* j *be a job attracted by some machine* $i \in \Gamma(j)$ *(not necessarily* $\sigma(j)$*). Then* $z_j = 3/4$ *if* j *is big and* $z_j = p_j/T$*, otherwise.*

We need to show that j is attracted by $\sigma(j)$. If $\sigma(j) = i$, then this holds trivially. If $\sigma(j) \neq i$, then either j is attracted by $\sigma(j)$ or $(j, i) \in L$, since no moves can be added. In the latter case $\sigma(j)$ attracts at least all big jobs if j is big and all jobs if j is small. In either case $\sigma(j)$ attracts j and therefore Fact 16 holds.

Let $i \in \mathcal{M}$ and $\chi \in [0, 1]^{\mathcal{J}}$ with $p^T \chi \geq T$, $\chi_j \in \{0, 1\}$ for every big job, and $\chi_j = 0$ if $i \notin \Gamma(j)$. We must show that $z^T \chi \geq y_i$.

If $y_i = 0$ then $z^T \chi \geq y_i$, since $z^T \chi$ is non-negative. Hence, assume w.l.o.g. that $y_i = 3/4$. By definition of attracted jobs, this means i attracts at least all big jobs. Moreover, the inequality holds trivially if $\chi_j = 1$ for some big job j, since $z_j = 3/4$ (Fact 16). Because for big jobs j we have $\chi_j \in \{0, 1\}$ and the case $\chi_j = 1$ is trivial, the only interesting case is where $\chi_j = 0$ for all big jobs j. We note that this is the only argument in which we use the integrality of some component in χ.

Define $S_i(L_{\leq k})$ for all $0 \leq k \leq \ell$ as in the algorithm. Because $y_i = 3/4$, we know that there is a $(j_k, i_k) = L_k$ $(k \leq \ell)$ such that $\sigma(j_k) = i$. Then there are two cases.

1. **Case: i attracts all jobs.** Since $\chi_j = 0$ is assumed for all big jobs j and i attracts all jobs, by Fact 16 we get $z^T\chi = p^T\chi/T \geq 1 > y_i$.
2. **Case: i attracts only big jobs.** Then j_k must be big and $p(S_i(L_{\leq k})) < 1/4 \cdot T$ (rule big-big). Note that $z_j = p_j/T$ for every small job $j \notin S_i(L_{\leq \ell}))$ with $i \in \Gamma(j)$, since by definition j is attracted by $\sigma(j)$ w.r.t. $L_{\leq k}$ (in particular, w.r.t. $L_{\leq \ell}$). Hence,

$$z^T\chi \geq \sum_{j\in\mathcal{J}\setminus S_i(L)} z_j\chi_j = \sum_{j\in\mathcal{J}\setminus S_i(L)} p_j\chi_j/T \geq (p^T\chi - p(S_i(L)))/T > (T - 1/4\cdot T)/T = y_i.$$

It remains to show that $\sum_{j\in\mathcal{J}} z_j < \sum_{i\in\mathcal{M}} y_i$. We show that, with amortization, good machines satisfy $z(\sigma^{-1}(i)) \leq y_i$ and for bad machines strict inequality holds.

Let i be a bad machine. A bad machine cannot have a big job assigned to it. Moreover, all jobs (in particular those in $\sigma^{-1}(i)$) are attracted by i. Hence,

$$z(\sigma^{-1}(i)) = p(\sigma^{-1}(i))/T < 1/4 < y_i.$$

Let i be a good machine. If $y_i = 0$, then i attracts no jobs and therefore $z(\sigma^{-1}(i)) = 0$. For the remaining part, assume that $y_i = 3/4$, that is to say, there exists a move $(j_k, i_k) = L_k$ $(k \leq \ell)$ such that $\sigma(j_k) = i$.

Case (big-big): i attracts only big jobs. Then j_k must be big and since (j_k, i_k) is not valid, it must be the only big job on i. Thus,

$$z(\sigma^{-1}(i)) = z_{j_B} = 3/4 = y_i.$$

Case (big-all): i attracts all jobs and j_k is big. Since (j_k, i_k) is not valid, we have $p(\sigma^{-1}(i) \setminus \{j_k\}) < 1/4 \cdot T$. In particular, $\sigma^{-1}(i) \setminus \{j_k\}$ does not contain another big job. Thus,

$$z(\sigma^{-1}(i)) = z_{j_k} + z(\sigma^{-1}(i) \setminus \{j_k\}) = z_{j_B} + p(\sigma^{-1}(i) \setminus \{j_k\})/T < 3/4 + 1/4 = y_i + 1/4.$$

Case (small-all): i attracts all jobs and j_k is small. Again, (j_k, i_k) is not valid. In particular, $\sigma^{-1}(i)$ cannot contain a big job. Hence,

$$z(\sigma^{-1}(i)) = p(\sigma^{-1}(i))/T = (p(\sigma^{-1}(i)) - p_{j_k})/T + p_{j_k}/T < 1/4 + 1/4 = y_i - 1/4.$$

It is easy to see that cases (big-all) and (small-all) are disjoint: Once a machine attracts all jobs, no new move will be added for a job assigned to it.

▶ **Fact 17.** *There are at least as many machines of case (small-all) as there are of case (big-all).*

The proof of Fact 17 is postponed to the end. With Fact 17, we can amortize those two cases and get

$$\sum_{j\in\mathcal{J}} z_j = \sum_{i\in\mathcal{M}} z(\sigma^{-1}(i)) < \sum_{i\in\mathcal{M}} y_i. \qquad ◀$$

Proof of Fact 17. Let $L_{b_1}, L_{b_2}, \ldots, L_{b_h}$ be the moves that correspond to case (big-all) machines, i.e., for each $(j_k, i_k) = L_{b_k}$ $(k \leq h)$, $\sigma(j_k)$ attracts all jobs and j_k is big.

We argue that there are $L_{s_1}, L_{s_2}, \ldots, L_{s_h}$ such that $b_1 < s_1 < b_2 < s_2 < \ldots b_h < s_h$, where for each $(j_k, i_k) = L_{s_k}$ $(k \leq h)$, j_k is small and therefore $\sigma(j_k)$ is a case (small-all) machine.

Let $k \leq h$. A critical argument is that the algorithm prefers moves of small jobs over those of big jobs. In particular, when $L_{b_{k+1}}$ is added, there does not exist a small job move that it can add instead. However, we have that $p(S_{\sigma(j_k)}(L_{\leq b_k})) \geq 1/4 \cdot T$ and since (j_k, i_k) is and was not valid, the load of small jobs on $\sigma(j_k)$ is strictly less than $1/4 \cdot T$. Hence, there exists a small job j in $S_{\sigma(j_k)}(L_{\leq b_k})$ which is not assigned to $\sigma(j_k)$. If no small move was added after L_{b_k} and before $L_{b_{k+1}}$, then $(j, \sigma(j_k))$ would have been preferred over $L_{b_{k+1}}$. ◀

References

1 Chidambaram Annamalai. Lazy local search meets machine scheduling. *CoRR*, abs/1611.07371, 2016. arXiv:1611.07371.

2 Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1357–1372. SIAM, 2015. doi:10.1137/1.9781611973730.90.

3 Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Trans. Algorithms*, 8(3):24:1–24:9, 2012. doi:10.1145/2229163.2229168.

4 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 31–40. ACM, 2006. doi:10.1145/1132516.1132522.

5 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005. doi:10.1145/1120680.1120683.

6 Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.

7 Klaus Jansen and Lars Rohwedder. On the configuration-lp of the restricted assignment problem. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2670–2678. SIAM, 2017. doi:10.1137/1.9781611974782.176.

8 Klaus Jansen and Lars Rohwedder. A quasi-polynomial approximation for the restricted assignment problem. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2017. doi:10.1007/978-3-319-59250-3_25.

9 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. doi:10.1007/BF01585745.

10 Lukás Polácek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. *ACM Trans. Algorithms*, 12(2):13:1–13:13, 2016. doi:10.1145/2818695.

11 Ola Svensson. Santa claus schedules jobs on unrelated machines. *SIAM J. Comput.*, 41(5):1318–1341, 2012. doi:10.1137/110851201.

12 José Verschae and Andreas Wiese. On the configuration-lp for scheduling on unrelated machines. *J. Scheduling*, 17(4):371–383, 2014. doi:10.1007/s10951-013-0359-4.

A NP-Hardness of the configuration-LP

The following reduction uses a typical idea for these kind of problems and we doubt that it is novel. We include it here only for the sake of completeness.

▶ **Theorem 18.** *Solving the configuration LP is NP-hard.*

Proof. We give the proof w.r.t. the Makespan objective. For Max-min the proof is analogous. Consider the NP-hard Partition problem: Given a set of natural numbers $a_1, \dots, a_n$, decide if there exists $A \subseteq \{a_1, \dots, a_n\}$ with $\sum_{j\in A} a_j = 1/2 \cdot \sum_{j=1}^n a_j$.

We construct the following instance for Makespan minimization and show that solving the configuration LP it is equivalent to the problem above.

Let there be two machines i_1, i_2 and for each a_j a job j with $\Gamma(j) = \{i_1, i_2\}$ and $p_j = a_j$. There exists such a subset if and only if the optimum of the configuration LP is $1/2 \cdot \sum_{j=1}^n a_j$. Recall that the configuration LP for makespan T assigns to each machine a convex combination of configurations, i.e., sets of jobs that do not exceed T in size. If there exists a solution for the Partition problem, then the optimum of the configuration LP is $1/2 \cdot \sum_{j=1}^n a_j$. We will assign one configuration with the solution of the Partition problem to one machine and one configuration with all remaining jobs to the other. If the optimum of the configuration LP is $1/2 \cdot \sum_{j=1}^n a_j$, then take the biggest configuration that is used. It must have a size of $1/2 \cdot \sum_{j=1}^n a_j$, or else the optimum would be lower. This configuration is a solution for the Partition problem. ◀

B Reducing the size of LP$_4$ (Max-min)

We have shown that LP$_4$ is strong for the Max-min case. In the Makespan case, already LP$_2$ gives the best bounds, which is much smaller than LP$_4$. In order to achieve a similar size, we will show that after a simple preprocessing step of the Max-min instance, the linear program can be reduced to $O(n^3)$ variables and constraints.

Let I be an instance of Max-min with job sizes p_j. Construct a new instance I^T by changing the size of each job $j \in \mathcal{J}$ to

$$p_j^T := \begin{cases} T & \text{if } p_j > T/4, \\ p_j & \text{otherwise.} \end{cases}$$

Recall that in the LP, the configurations for big jobs $\mathcal{J}_B$ (that have size $> T/4$) are defined as $C_B(T) := \{\chi \in \{0,1\}^{\mathcal{J}_B}\}$.

The argument for restricting the support of these configurations was that we do not need a configuration χ if there is a $\chi' \le \chi$ (component-wise) with $|\mathrm{supp}(\chi')| < |\mathrm{supp}(\chi)|$ and $p^T\chi' \ge T$. It is easy to see that after rounding the sizes, the relevant configuration have at most one non-zero component. Hence, there are only $O(n)$ many, which gives a total size of $O(n^3)$ for the compact LP. Now we need to show that after rounding the sizes, we still get a ratio of 4. In other words,

$$\frac{1}{4}\mathrm{OPT}(\mathrm{LP}_4(I^T)) \le \mathrm{OPT}(I) \le \mathrm{OPT}(\mathrm{LP}_4(I^T)).$$

Since the sizes have only increased, the relaxation can only have gotten weaker, i.e., $\mathrm{OPT}(\mathrm{LP}_4(I^T)) \ge \mathrm{OPT}(\mathrm{LP}_4(I)) \ge \mathrm{OPT}(I)$.

Let $T > 4 \cdot \mathrm{OPT}(I)$. We need to show that $\mathrm{LP}_4(I^T)$ is infeasible w.r.t. T. Notice how $\mathrm{OPT}(I^T) = \mathrm{OPT}(I)$: Given the optimal allocation for I^T, the machine that has the minimum load cannot have any jobs j with $p_j \ge T/4 > \mathrm{OPT}(I)$. Otherwise this allocation would yield a higher value for I than $\mathrm{OPT}(I)$. Hence, on the machine with minimum load the jobs have the same size in I and I^T. This means the solution has the same value for I. Since LP$_4$ has an integrality gap of at most 4, we get

$$\frac{1}{4}\mathrm{OPT}(\mathrm{LP}_4(I^T)) \le \mathrm{OPT}(I^T) = \mathrm{OPT}(I).$$

In other words, the highest value for which $\mathrm{LP}_4(I^T)$ is feasible is $4 \cdot \mathrm{OPT}(I^T) < T$.

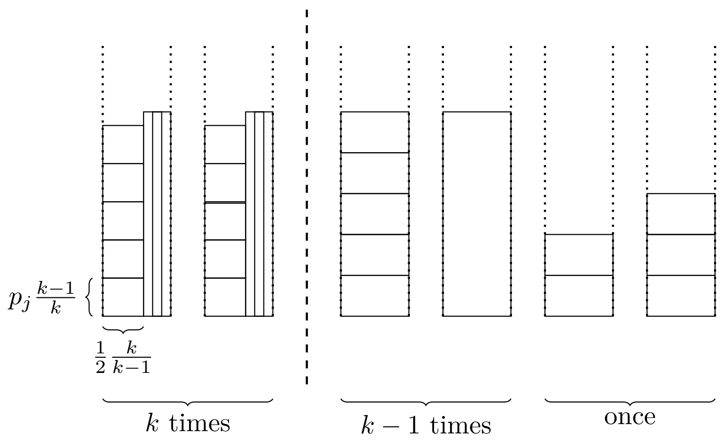

Figure 2 Fractional and integral solution for lower bound (Max-min)

C Lower bound (Max-min)

Here, we give a lower bound of 2.5 for LP$_4$ (in particular, LP$_3$, LP$_2$) for Max-min.

Let k be an even number and consider an instance with k machines, i.e., $k/2$ pairs of machines, and $6k-1$ jobs. For each of the $k/2$ pairs (i_1, i_2) let there be 5 jobs j with $p_j = 1/5$ and $\Gamma(j) = \{i_1, i_2\}$. Furthermore let there be $k/2-1$ jobs j with $p_j = 1$ and $\Gamma(j) = \mathcal{M}$, i.e., they can be assigned anywhere.

Assume toward contradiction that there is a schedule with makespan strictly more than 2/5. There must be at least one pair of machines (i_1, i_2) that do not have a job of size 1 assigned to them. Since there are only 5 jobs of size 1/5 that are allowed on i_1 and i_2, one of the two machines has at most two. A contradiction.

Next, we show that LP$_4$ is feasible for $T = (k-1)/k$. For every $i \in \mathcal{M}$ and every job j_B of size 1, let $x_{i,j_B} = 1/k$, i.e., the big job is distributed evenly across all machines. For every other job j, split it across the two machines it is allowed on. More formally, let $x_{i_1,j} = x_{i_2,j} = 1/2$ with $\{i_1, i_2\} = \Gamma(j)$. Clearly x is in the allocation polytope. Let $i \in \mathcal{M}$. We need to verify that

$$(x_{i,j})_{j\in\mathcal{J}} \in \operatorname{conv}\{\chi \in [0,1]^{\mathcal{J}} : p^T\chi \geq (k-1)/k \text{ and } \chi_j \in \{0,1\} \text{ if } p_j \cdot 4 > (k-1)/k\}.$$

We define one vector for every $j_B \in \mathcal{J}$ with $p_{j_B} = 1$ and one vector for the small jobs, which depends on the machine it is used for.

$$\chi_j^{(j_B)} := \begin{cases} 1 & \text{if } j = j_B, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and } \chi_j^{(i)} := \begin{cases} \frac{k-1}{k} & \text{if } p_j = 1/5 \text{ and } i \in \Gamma(j), \\ 0 & \text{otherwise.} \end{cases}$$

Note that we have $p^T\chi^{(j_B)} = p_{j_B} = 1 \geq \frac{k-1}{k} = T$ as well as $p^T\chi^{(i)} = 5 \cdot \frac{1}{5} \cdot \frac{k-1}{k} = \frac{k-1}{k} = T$. The integrality constraint is satisfied for all jobs of size 1 and since $1/5 \cdot 4 \leq (k-1)/k$, for sufficiently large k, it is not necessary for the small jobs. Also, it holds that $(x_{i,j})_{j\in\mathcal{J}} = \sum_{j_B\in\mathcal{J}:p_{j_B}=1} 1/k \cdot \chi^{(j_B)} + 1/2 \cdot k/(k-1) \cdot \chi^{(i)}$, as shown below.

Let j_B be a job of size 1. Then all vectors but $\chi^{(j_B)}$ have 0 for j_B, hence $x_{i,j_B} = 1/k \cdot \chi_{j_B}^{(j_B)}$. Next, let j be a job of size 1/5. Then $x_{i,j} = 1/2 = 1/2 \cdot k/(k-1) \cdot \chi_j^{(i)}$ if $i \in \Gamma(j)$ and $x_{i,j} = 0 = 1/2 \cdot k/(k-1) \cdot \chi_j^{(i)}$, otherwise.

For this instance, we get a gap that approaches 5/2 as k tends to infinity. Similar constructions work when using 14 jobs of size 1/7, 18 jobs of size 1/9, etc. The lower bound becomes weaker the smaller the size is chosen, but it always stays strictly above 4, the best lower bound known for the configuration LP.

D Analysis (Makespan)

▶ **Theorem 19.** *The algorithm terminates after finitely many iterations of the main loop.*

Proof. Let ℓ be the length of L. We define a potential function

$$\Phi(L) = (g, s_0, s_1, s_2, \dots, s_\ell, -1),$$

where g is the number of good machines and s_k is the number of pairs $(i,j) \in \mathcal{M} \times \mathcal{J}$ where i repels j w.r.t. $L_{\le k}$ and $\sigma(j) \neq i$. Note that the length of $\Phi(L)$ is bounded by $|\mathcal{M}| \cdot |\mathcal{J}| + 2$ and every component is an integer between -1 and $|\mathcal{J}| \cdot |\mathcal{M}|$. Thus, the range of the potential function is finite.

We will show that the vector increases lexicographically after every iteration of the main loop; hence the running time is bounded by the number of such vectors times the maximum number of operations in an iteration and is in particular finite. For the lexicographic increase, consider two cases. Either a new move is added, which increases the potential function by replacing the last component with some non-negative value, or a move is performed.

If the move turns a bad machine good, g increases and thereby $\Phi(L)$ increases lexicographically as well. Otherwise, let $\ell' \le \ell$ be the length of the list after a move (j,i) is performed. Recall the algorithm prevents jobs repelled by a machine from being moved there, i.e., j is not repelled by i machine w.r.t. $L_{\le \ell'}$. Moreover, the set of repelled jobs by some machine w.r.t. $L_{\le 0}, \dots, L_{\le \ell'}$ can only grow. This can be observed from the definition of repelled jobs. It follows that $s_0, \dots, s_{\ell'}$ do not decrease. Finally, since j is repelled by its previous machine i' w.r.t. $L_{\le \ell'}$ and after the move $\sigma(j) \neq i'$ holds, the value of $s_{\ell'}$ has increased. ◀

▶ **Theorem 20.** *If* LP$_2$ *is feasible, the algorithm always has an operation to perform.*

Proof. As in the algorithm, call a job j small if $p_j < 1/2 \cdot T = 1/r \cdot T$ and big otherwise. Suppose toward contradiction, there are bad machines, no move in L is valid, and no move can be added to L. We will construct values $(z_j)_{j \in \mathcal{J}}$, $(y_i)_{i \in \mathcal{M}}$ with the properties as in Lemma 12 and thereby show that LP$_2$ is infeasible.

For every $j \in \mathcal{J}$ let $z_j = \min\{p_j/T, 5/6\}$ if j is repelled by $\sigma(j)$ and $z_j = 0$ otherwise. Let $y_i := 1$ if $i \in \mathcal{M}$ repels all jobs and $y_i = z(\sigma^{-1}(i))$ otherwise.

▶ **Fact 21.** *Let j be a small job not repelled by $i \in \Gamma(j)$. Then $z_j = 0$.*

If $i = \sigma(j)$, this is by definition. In the other case, assume toward contradiction $z_j \neq 0$, i.e. j is repelled by $\sigma(j)$. (j,i) cannot be in L or else i would repel small jobs. Since (j,i) also cannot be added to L, i must repel j. A contradiction.

Let $i \in \mathcal{M}$ and $\chi \in [0,1]^{\mathcal{J}}$ with $p^T\chi \le T$, $\chi_j \in \{0,1\}$ for every big job, and $\chi_j = 0$ if $i \notin \Gamma(j)$. We must show that $z^T\chi \le y_i$.

If $y_i = 1$ it holds because of $z^T\chi \le p^T\chi/T \le 1$. We assume w.l.o.g. that i does not repel all jobs and thus $y_i = z(\sigma^{-1}(i))$. In particular, i does not repel small jobs that are on other machines. If $\chi_j = 0$ or $z_j = 0$ for all big jobs j, then with Fact 21 we get $z^T\chi \le z(\sigma^{-1}(i))$. Also, there can be at most one big job j_B with $\chi_{j_B} = 1$, since it has $p_{j_B} > 1/2 \cdot T$ and thus $p^T\chi$ would be greater than T, otherwise.

We recap: The only interesting case is when $y_i = z(\sigma^{-1}(i))$, there is one big job j_B with $\chi_{j_B} z_{j_B} = \min\{p_{j_B}/T, 5/6\}$, and all other big jobs j'_B have $\chi_{j'_B} z_{j'_B} = 0$.

1. **Case: j_B is repelled by i.** This must be because there is some move $(j_k, i_k) = L_k$ such that $t(L_{\leq k}, (j_k, i_k)) \geq p_{j_B}$ (big-big). Recall that $t(L_{\leq k}, (j_k, i_k))$ is the minimum t with

   $$p(\{j \in \sigma^{-1}(i) : j \in S_i(L_{\leq k}) \text{ or } 1/2 < p_j \leq t\}) + p_{j_k} > 11/6.$$

   In particular, there must be a big job $j'_B \in \sigma^{-1}(i)$ with $t(L_{\leq k}, (j_k, i_k)) = p_{j'_B} \geq p_{j_B}$ and j'_B is also repelled by i (big-big). Using Fact 21 we get

   $$z^T\chi = \sum_{j \in \mathcal{J}} z_j \chi_j \leq z(S_i(L)) + z_{j_B} \leq z(S_i(L)) + z_{j'_B} \leq z(\sigma^{-1}(i)) = y_i.$$

2. **Case: j_B is not repelled by i.** Then (j_B, i) must already be in L, i.e., $L_k = (j_B, i)$ for some $k \leq \ell$, and since i does not repel all jobs, rule (big-big) must apply for this move. Let $R = \{j \in \sigma^{-1}(i) : j \in S_i(L_{\leq k}) \text{ or } 1/2 < p_j \leq t(L_{\leq k}, (j_B, i))\}$. Then

   $$p(R) + p_{j_B} > 11/6 \cdot T \geq p^T\chi + 5/6 \cdot T.$$

   Furthermore, all jobs in R are also repelled by i. If $z_{j'_B} = 5/6$ for some $j'_B \in R$, like in the previous case we get $z^T\chi \leq z(S_i(L)) + z_{j_B} \leq z(S_i(L)) + z_{j'_B} \leq z(\sigma^{-1}(i))$. Otherwise, we have $z_{j'} = p_{j'}/T$ for all $j' \in R$. Thus,

   $$\begin{aligned} z^T\chi = z_{j_B} + \sum_{j \in \mathcal{J}\setminus\{j_B\}} z_j\chi_j \leq z_{j_B} + \sum_{j \in \mathcal{J}\setminus\{j_B\}} p_j\chi_j/T = z_{j_B} + (p^T\chi - p_{j_B})/T \\ < z_{j_B} + (p(R) - 5/6 \cdot T)/T \leq p(R)/T \leq z(\sigma^{-1}(i)). \end{aligned}$$

It remains to show that $\sum_{j \in \mathcal{J}} z_j > \sum_{i \in \mathcal{M}} y_i$. We prove that, with amortization, good machines satisfy $z(\sigma^{-1}(i)) \geq y_i$ and on bad machines strict inequality holds.

Let i be a bad machine. Then i repels all jobs (in particular those in $\sigma^{-1}(i)$). Hence,

$$z(\sigma^{-1}(i)) \geq 5/6 \cdot p(\sigma^{-1}(i))/T > 55/36 > y_i.$$

For good machines that do not repel all jobs, equality holds by definition. We will partition those good machines that do repel all jobs into those $i \in \mathcal{M}$ which have $(j, i) \in L$ for a small job j (case small-all) and those that do not (case big-all).

▶ **Fact 22.** *There are at least as many machines of case (small-all) as there are of case (big-all).*

The proof of Fact 22 is postponed until after the main proof. Let i be a machine of case (big-all). Then there is a big job j_B with $(j_B, i) \in L$ and this move is not valid. Either there is a job $j \in \sigma^{-1}(i)$ with $z_j = 5/6$ or $z_j = p_j/T$ for all $j \in \sigma^{-1}(i)$. Thus,

$$\begin{aligned} z(\sigma^{-1}(i)) &\geq \min\{5/6,\ p(\sigma^{-1}(i))/T\} \\ &\geq \min\{5/6,\ 11/6 - p_{j_B}/T\} \\ &\geq 5/6 = y_i - 1/6. \end{aligned}$$

Next, let i be a machine of case (small-all). Then there is a move $(j_S, i) \in L$ with j_S small. Of course, this move is not valid. In the following, we distinguish between the cases where $\sigma^{-1}(i)$ has no job j with $z_j = 5/6$, one such job, or at least two. Note that these jobs have $p_j \leq T$.

$$\begin{aligned} z(\sigma^{-1}(i)) &\geq \min\{p(\sigma^{-1}(i))/T,\ (p(\sigma^{-1}(i)) - T)/T + 5/6,\ 10/6\} \\ &\geq \min\{11/6 - p_{j_S}/T - 1/6,\ 10/6\} \\ &\geq 7/6 = y_i + 1/6. \end{aligned}$$

Because of Fact 22, we can amortize case (small-all) and case (big-all) and get

$$\sum_{j\in\mathcal{J}} z_j = \sum_{i\in\mathcal{M}} z(\sigma^{-1}(i)) > \sum_{i\in\mathcal{M}} y_i. \qquad ◀$$

Proof of Fact 22. Let $L_{b_1}, L_{b_2}, \dots, L_{b_h}$ be the moves that correspond to case (big-all) machines, i.e., for each $(j_k, i_k) = L_{b_k}$ $(k \le h)$, i_k repels all jobs and j_k is big.

We argue that there are $L_{s_1}, L_{s_2}, \dots, L_{s_h}$ such that $b_1 < s_1 < b_2 < s_2 < \dots b_h < s_h$, where for each $(j_k, i_k) = L_{s_k}$ $(k \le h)$, j_k is small and therefore i_k is a case (small-all) machine.

Let $k \le h$. A critical argument is that the algorithm prefers moves of small jobs over those of big jobs. In particular, when $L_{b_{k+1}}$ is added, there does not exist a small job move that it can add instead. Either L_{b_k} has already been subject to rule (big-all) when it was added or it turned to this after a repelled job was removed. Either way, at this time it was the last move in L and we had that

$$p(\{j \in \sigma^{-1}(i_k) : p_j > 1/2\}) + p(S_{i_k}(L_{\le b_k})) + p_{j_k} \le 11/6 \cdot T < p(\sigma^{-1}(i_k)) + p_{j_k},$$

where the first inequality comes from rule (big-all) and the second one from the fact that (j_k, i_k) is and was not valid. Hence, there must be a small job j in $\sigma^{-1}(i)$ which is not in $S_{i_k}(L_{\le b_k})$. By definition of $S_{i_k}(L_{\le b_k})$, j has a machine $i \in \Gamma(j)$ by which it was not repelled. Therefore there has been a small job move that could be added. This means $L_{b_{k+1}}$ was only added after a small job move has been. ◀

Just Take the Average! An Embarrassingly Simple 2^n-Time Algorithm for SVP (and CVP)

Divesh Aggarwal[*1] **and Noah Stephens-Davidowitz**[†2]

1 **CQT and Department of Computer Science, NUS, Singapore**
`dcsdiva@nus.edu.sg`

2 **New York University, New York, USA**
`noahsd@gmail.com`

Abstract

We show a $2^{n+o(n)}$-time (and space) algorithm for the Shortest Vector Problem on lattices (SVP) that works by repeatedly running an embarrassingly simple "pair and average" sieving-like procedure on a list of lattice vectors. This matches the running time (and space) of the current fastest known algorithm, due to Aggarwal, Dadush, Regev, and Stephens-Davidowitz (ADRS, in *STOC*, 2015), with a far simpler algorithm. Our algorithm is in fact a modification of the ADRS algorithm, with a certain careful rejection sampling step removed.

The correctness of our algorithm follows from a more general "meta-theorem," showing that such rejection sampling steps are unnecessary for a certain class of algorithms and use cases. In particular, this also applies to the related $2^{n+o(n)}$-time algorithm for the Closest Vector Problem (CVP), due to Aggarwal, Dadush, and Stephens-Davidowitz (ADS, in *FOCS*, 2015), yielding a similar embarrassingly simple algorithm for γ-approximate CVP for any $\gamma = 1 + 2^{-o(n/\log n)}$. (We can also remove the rejection sampling procedure from the $2^{n+o(n)}$-time ADS algorithm for *exact* CVP, but the resulting algorithm is still quite complicated.)

1998 ACM Subject Classification F.2.2 Nonnumerical algorithms and problems – Geometrical problems and computations

Keywords and phrases Lattices, SVP, CVP

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.12

1 Introduction

A lattice $\mathcal{L} \subset \mathbb{R}^n$ is the set of all integer linear combinations of some linearly independent basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^n$,

$$\mathcal{L} := \left\{ \sum_{i=1}^{n} z_i \boldsymbol{b}_i \ : \ z_i \in \mathbb{Z} \right\} .$$

The two most important computational problems on lattices are the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). Given a basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ for a lattice

[*] The first author was supported by the Singapore Ministry of Education and the National Research Foundation, also through the Tier 3 Grant "Random numbers from quantum processes" MOE2012-T3-1-009.

[†] The second author was supported by the National Science Foundation (NSF) under Grant No. CCF-1320188, and the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236.

$\mathcal{L} \subset \mathbb{R}^n$, SVP asks us to find a shortest non-zero vector in $\mathcal{L}$, and CVP asks us to find a closest lattice vector to some target vector $\boldsymbol{t} \in \mathbb{R}^n$. (Throughout this paper, we define distance in terms of the Euclidean, or ℓ_2, norm.) CVP seems to be the harder of the two problems, as there is an efficient reduction from CVP to SVP that preserves the dimension n [17], but both problems are known to be NP-hard [36, 3]. They are even known to be hard to approximate for certain approximation factors [24, 14, 21, 18].

Algorithms for solving these problems, both exactly and over a wide range of approximation factors, have found innumerable applications since the founding work by Lenstra, Lenstra, and Lovász in 1982 [23]. (E.g., [23, 19, 34, 20, 13].) More recently, following the celebrated work of Ajtai [4] and Regev [31], a long series of works has resulted in many cryptographic constructions whose security is based on the assumed *worst-case* hardness of approximating these (or closely related) problems. (See [29] for a survey of such constructions.) And, some of these constructions are now nearing widespread deployment. (See, e.g., [28, 7, 11].)

Nearly all of the fastest known algorithms for lattice problems – either approximate or exact – work via a reduction to either exact SVP or exact CVP (typically in a lower dimension). Even the fastest known polynomial-time algorithms (which solve lattice problems only up to large approximation factors) work by solving exact SVP on low-dimensional sublattices [33, 15, 27]. Therefore, algorithms for exact lattice problems are of particular importance, both theoretically and practically (and both for direct applications and to aid in the selection of parameters for cryptography). Indeed, much work has gone into improving the running time of these algorithms (e.g., [20, 5, 6, 30, 25, 26]), culminating in $2^{n+o(n)}$-time algorithms for both problems based on the technique of *discrete Gaussian sampling*, from our joint work with Dadush and Regev [1] and follow-up work with Dadush [2].

In order to explain our contribution, we first give a high-level description of the SVP algorithm from [1], which we refer to as the ADRS algorithm. (The presentation below does not represent the way that we typically view that algorithm.)

1.1 Sieving by averages

One can think of the ADRS algorithm as a rather strange variant of *randomized sieving.* Recall that the celebrated randomized sieving technique due to Ajtai, Kumar, and Sivakumar [5] starts out with a list of $2^{O(n)}$ not-too-long random vectors $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$ sampled from some efficiently samplable distribution. The sieving algorithm then repeatedly (1) searches for pairs of vectors $(\boldsymbol{X}_i, \boldsymbol{X}_j)$ that happen to be remarkably close together; and then (2) replaces the old list of vectors with the differences of these pairs $\boldsymbol{X}_i - \boldsymbol{X}_j$.

The ADRS algorithm similarly starts with a randomly chosen collection of $2^{n+o(n)}$ not-too-long vectors $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$ and repeatedly (1) selects pairs of vectors according to some rule; and (2) replaces the old list of vectors with some new vectors generated from these pairs. However, instead of taking the differences $\boldsymbol{X}_i - \boldsymbol{X}_j$ of pairs $(\boldsymbol{X}_i, \boldsymbol{X}_j)$, the ADRS algorithm takes *averages*, $(\boldsymbol{X}_i + \boldsymbol{X}_j)/2$.

Notice that the average $(\boldsymbol{X}_i + \boldsymbol{X}_j)/2$ of two lattice vectors $\boldsymbol{X}_i, \boldsymbol{X}_j \in \mathcal{L}$ will not generally be in the lattice. In fact, this average will be in the lattice if and only if the two vectors are equivalent mod $2\mathcal{L}$, i.e., $\boldsymbol{X}_i \equiv \boldsymbol{X}_j \bmod 2\mathcal{L}$. Therefore, at a minimum, the ADRS algorithm should select pairs that lie in the same *coset* mod $2\mathcal{L}$. (Notice that there are 2^n possible cosets.) I.e., the simplest possible version of "sieving by averages" just repeats Procedure 1 many times (starting with a list of $2^{n+o(n)}$ vectors, which is sufficient to guarantee that we can pair nearly every vector with a different unique vector in the same coset). The ADRS algorithm is more complicated than this, but it still only uses the cosets of the vectors mod $2\mathcal{L}$ when it decides which vectors to pair.

It might seem like a rather big sacrifice to only look at a vector's coset, essentially ignoring all geometric information. For example, the ADRS algorithm (and our variant) is likely to

Procedure 1: The basic "pair and average" procedure, which computes the averages $(\boldsymbol{X}_i + \boldsymbol{X}_j)/2$ of disjoint pairs $(\boldsymbol{X}_i, \boldsymbol{X}_j)$ satisfying $\boldsymbol{X}_i \equiv \boldsymbol{X}_j \bmod 2\mathcal{L}$.

```
Pair_and_Average ($\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$)
Input  : List of vectors $\boldsymbol{X}_i \in \mathcal{L} - \boldsymbol{t}$
Output : List of vectors $\boldsymbol{Y}_i \in \mathcal{L} - \boldsymbol{t}$
for each unpaired vector $\boldsymbol{X}_i$ do
    if there exists an unpaired vector $\boldsymbol{X}_j$ with $\boldsymbol{X}_j \equiv \boldsymbol{X}_i \bmod 2\mathcal{L}$ then
        add $(\boldsymbol{X}_i + \boldsymbol{X}_j)/2$ to the output
    end
end
```

Procedure 2: The "reject and average" sieving procedure, which repeatedly applies some rejection sampling procedure f according to the cosets of the $\boldsymbol{X}_i \bmod 2\mathcal{L}$ and then applies Procedure 1 (Pair_and_Average) to the "accepted" vectors. Here, f is some (possibly randomized) function that maps a list of cosets $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_M) \bmod 2\mathcal{L}$ to a set of "accepted" indices $\{j_1, \ldots, j_m\} \subseteq \{1, \ldots, M\}$.

```
Reject-and-Average Sieve ($\ell; \boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$)
Input  : Number of steps $\ell$, list of vectors $\boldsymbol{X}_i \in \mathcal{L} - \boldsymbol{t}$
Output : List of vectors $\boldsymbol{Y}_i \in \mathcal{L} - \boldsymbol{t}$
for $i = 1, \ldots, \ell$ do
    for $j = 1, \ldots, M$ do
        set $\boldsymbol{c}_j$ to be the coset of $\boldsymbol{X}_j \bmod 2\mathcal{L}$
    end
    $\{j_1, \ldots, j_m\} \leftarrow f(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_M)$
    $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{M'}) \leftarrow$ Pair_and_Average$(\boldsymbol{X}_{j_1}, \ldots, \boldsymbol{X}_{j_m})$
    $M \leftarrow M'$
end
output $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$.
```

miss many opportunities to pair two vectors whose average is very short. But, in exchange for this sacrifice, we get very strong control over the distribution of the vectors at each step. In particular, before applying Procedure 1, the ADRS algorithm uses a careful *rejection sampling* procedure over the cosets to guarantee that at each step of the algorithm, the vectors are distributed as independent samples from a distribution that we understand very well (the discrete Gaussian, which we describe in Section 1.3). I.e., at each step, the algorithm randomly throws away many of the vectors in each coset according to some rule that depends only on the list of cosets, and it only runs Procedure 1 on the remaining vectors, as shown in Procedure 2. This rejection sampling procedure is so selective that, though the algorithm starts out with $2^{n+o(n)}$ vectors, it typically finishes with only about $2^{n/2}$ vectors.

This does seem quite wasteful (since the algorithm typically throws away the vast majority of its input vectors) and a bit naive (since the algorithm ignores, e.g., the lengths of the vectors). But, because we get such good control over the output distribution, the result is still the fastest known algorithm for SVP.[1] (The algorithm for CVP in [2], which we refer

[1] There are various "heuristic" sieving algorithms for SVP that run significantly faster (e.g., in time $(3/2)^{n/2}$ [10]) but do not have formal proofs of correctness. One of the reasons that these algorithms lack proofs is because we do not understand their output distributions.

to as the ADS algorithm, relies on the same core idea, plus a rather complicated recursive procedure that converts an approximate CVP algorithm with certain special properties into an exact CVP algorithm.)

1.2 Our contribution

Our main contribution is to show that the rejection sampling procedure used in the ADRS algorithm is unnecessary! Indeed, informally, we show that "any collection of vectors that can be found via such a procedure (when the input vectors are sampled independently from an appropriate distribution) can also be found without it." (We make this precise in Theorem 9.) In particular, the SVP algorithm in [1] can be replaced by an extremely simple algorithm, which starts with a list of $2^{n+o(n)}$ vectors sampled from the right distribution and then just runs Procedure 1 repeatedly. (Equivalently, it runs Procedure 2 with f taken to be the trivial function that always outputs all indices, $\{1, \ldots, M\}$.)

▶ **Theorem 1** (SVP, informal). *There is a $2^{n+o(n)}$-time (and space) algorithm for SVP that starts with $2^{n+o(n)}$ vectors sampled from the same distribution as the ADRS algorithm and then simply applies Procedure 1 repeatedly, $\ell = O(\log n)$ times.*

The situation for CVP is, alas, more complicated because Procedure 2 is not the most difficult part of the *exact* CVP algorithm from [2]. Indeed, while this algorithm does run Procedure 2 and we *do* show that we can remove the rejection sampling procedure, the resulting algorithm retains the complicated recursive structure of the original algorithm. However, [2] also shows a much simpler non-recursive version of the algorithm that solves CVP up to an extremely good approximation factor. If we are willing to settle for such an algorithm, then we get the same result for CVP.

▶ **Theorem 2** (CVP, informal). *There is a $2^{n+o(n)}$-time (and space) algorithm that approximates CVP up to an approximation factor γ for any $\gamma = 1 + 2^{-o(n/\log n)}$ that starts with $2^{n+o(n)}$ vectors from the same distribution as the ADS algorithm and then simply applies Procedure 1 repeatedly, $\ell = o(n/\log n)$ times.*

In practice, such a tiny approximation factor is almost always good enough for applications.

1.3 Proof techniques

To describe the technical ideas behind our result, we now define the *discrete Gaussian distribution*, which plays a fundamental role in the algorithms in [1, 2] and a big part in our analysis. For any vector $\boldsymbol{x} \in \mathbb{R}^n$ and parameter $s > 0$, we define its Gaussian mass as

$$\rho_s(\boldsymbol{x}) := \exp(-\pi\|\boldsymbol{x}\|^2/s^2) ,$$

and we extend this definition to a shift of a lattice $\mathcal{L} \subset \mathbb{R}^n$ with shift vector $\boldsymbol{t} \in \mathbb{R}^n$ in the natural way,

$$\rho_s(\mathcal{L} - \boldsymbol{t}) := \sum_{\boldsymbol{y} \in \mathcal{L}} \rho_s(\boldsymbol{y} - \boldsymbol{t}) .$$

The *discrete Gaussian distribution* $D_{\mathcal{L}-\boldsymbol{t},s}$ is the probability distribution over $\mathcal{L} - \boldsymbol{t}$ induced by this measure, given by

$$\Pr_{\boldsymbol{X} \sim D_{\mathcal{L}-\boldsymbol{t},s}}[\boldsymbol{X} = \boldsymbol{y} - \boldsymbol{t}] := \frac{\rho_s(\boldsymbol{y} - \boldsymbol{t})}{\rho_s(\mathcal{L} - \boldsymbol{t})}$$

for any $\boldsymbol{y} \in \mathcal{L}$.

For very large parameters $s > 0$, we can sample from the discrete Gaussian $D_{\mathcal{L}-\boldsymbol{t},s}$ efficiently [16, 12]. (Notice that $D_{\mathcal{L}-\boldsymbol{t},s}$ tends to concentrate on shorter vectors as the parameter $s > 0$ gets smaller. In particular, [1] showed that about 1.38^n independent samples from the discrete Gaussian $D_{\mathcal{L},s}$ with an appropriately chosen parameter s will contain a shortest non-zero lattice vector with high probability. See Proposition 16.) So, in [1, 2], we use Procedure 2 with a carefully chosen rejection sampling procedure f in order to convert many independent samples from $D_{\mathcal{L}-\boldsymbol{t},s}$ with a relatively large parameter s to some smaller number of independent samples from $D_{\mathcal{L}-\boldsymbol{t},s/2^{\ell/2}}$.

This rejection sampling is certainly necessary if we wish to use Procedure 1 to sample from the discrete Gaussian distribution. Our new observation is that, even when we do not do this rejection sampling, the output of Procedure 1 still has a nice distribution. In particular, if we fix the coset mod $2\mathcal{L}$ of a pair of discrete Gaussian vectors $(\boldsymbol{X}_i, \boldsymbol{X}_j)$ with parameter $s > 0$, then their average will be distributed as a *mixture* of discrete Gaussians with parameter $s/\sqrt{2}$ over the cosets of $2\mathcal{L}$. I.e., while the probability of their average landing in any particular coset will not in general be proportional to the Gaussian mass of the coset, the distribution *inside each coset* will be exactly Gaussian. (See Lemma 6.)

This observation is sufficient to prove that no matter what rejection sampling procedure f we use in Procedure 2, if the input consists of independent samples from $D_{\mathcal{L}-\boldsymbol{t},s}$, the output will always be distributed as some *mixture* of samples from $D_{2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t},s/2^{\ell/2}}$ over the cosets $\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})$. I.e., while the output distribution might distribute weight amongst the cosets differently, if we condition on a fixed number of vectors landing in each coset, the output will always be distributed as independent discrete Gaussian vectors with parameter $s/2^{\ell/2}$. It follows immediately that "rejection sampling cannot help us." In particular, the probability that the output of Procedure 2 will contain a particular vector (say a shortest non-zero vector) with any rejection sampling procedure f will never be greater than the probability that we would see that vector without rejection sampling (i.e., when f is the trivial function that outputs $\{1, \ldots, M\}$).[2] See Corollary 8 and Theorem 9 for more detail.

1.4 An open problem – towards a $2^{n/2}$-time algorithm

Our result shows that all known applications of the $2^{n+o(n)}$-time discrete Gaussian sampling algorithms in [1, 2] work just as well if we remove the rejection sampling procedure from these algorithms. This in particular includes the SVP application mentioned in Theorem 1 and the approximate CVP application mentioned in Theorem 2. (More generally, we can remove the rejection sampling procedure from any application that simply relies on finding a set of vectors with a certain property in the output distribution, such as a shortest non-zero vector, all shortest non-zero vectors, a vector that is close to a shortest lattice vector in $\mathcal{L} - \boldsymbol{t}$, etc.)

However, [1] also presents a $2^{n/2+o(n)}$-time algorithm that samples from $D_{\mathcal{L}-\boldsymbol{t},s}$ as long as the parameter $s > 0$ is not too small. (In particular, we need $s \geq \sqrt{2}\eta_{1/2}(\mathcal{L})$, where $\eta_{1/2}(\mathcal{L})$ is the *smoothing parameter* of the lattice. See [1] or [35] for the details.) This algorithm is similar to the $2^{n+o(n)}$-time algorithms in that it starts with independent discrete Gaussian vectors with some high parameter, and it gradually lowers the parameter using a rejection sampling procedure together with a procedure that takes the averages of pairs of vectors that lie in the same coset modulo some sublattice. But, it fails for smaller parameters specifically

[2] Notice that this property is far from obvious without the observation that the output distribution is always a mixture of Gaussians over the cosets. For example, if we modified Procedure 2 so that f acted on the $\boldsymbol{X}_i$ themselves, rather than just their cosets mod $2\mathcal{L}$, then this property would no longer hold.

because the rejection sampling procedure that it uses must throw out too many vectors in this case. (In [35], we use a different rejection sampling procedure that never throws away too many vectors, but we do not know how to implement it in $2^{n/2+o(n)}$ time for small parameters $s < \sqrt{2}\eta_{1/2}(\mathcal{L})$.) If we could find a suitable variant of this algorithm that works for small parameters, we would be able to solve SVP in $2^{n/2+o(n)}$ time.

So, we are naturally very interested in understanding what happens when we simply remove the rejection sampling procedure from this algorithm. And, the fact that this works out so nicely for the $2^{n+o(n)}$-time algorithm works seems quite auspicious! Unfortunately, we are unable to say very much at all about the resulting distribution in the $2^{n/2+o(n)}$-time case.[3] So, we leave the study of this distribution as an open problem.

Organization

In Section 2, we review a few basic facts necessary to prove our main "meta-theorem," Theorem 9, which shows that "rejection sampling is unnecessary." In Section 3, we finish this proof. In particular, this implies Theorem 1 and 2. For completeness, in the appendix, we prove these theorems more directly and show the resulting algorithms in full detail.

2 Preliminaries

We write $\mathbb{N} := \{0, 1, \ldots\}$ for the natural numbers (including zero). We make little to no distinction between a random variable and its distribution. For $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, we write $\|\boldsymbol{x}\| := (x_1^2 + \cdots + x_n^2)^{1/2}$ for the Euclidean norm of $\boldsymbol{x}$. For any set S, we write $S^* := \{(x_1, \ldots, x_M) \ : \ x_i \in S\}$ for the set lists over S of finite length. (The order of elements in a list $\mathcal{M} \in S^*$ will never concern us. We could therefore instead use multisets.)

2.1 Lattices

A lattice $\mathcal{L} \subset \mathbb{R}^n$ is the set of integer linear combinations

$$\mathcal{L} := \{a_1\boldsymbol{b}_1 + \cdots + a_n\boldsymbol{b}_n \ : \ a_i \in \mathbb{Z}\}$$

of some linearly independent basis vectors $\mathbf{B} := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$. We sometimes write $\mathcal{L}(\mathbf{B})$ for the lattice spanned by $\mathbf{B}$.

We write $\mathcal{L}/(2\mathcal{L})$ for the set of cosets of $\mathcal{L}$ over $2\mathcal{L}$. E.g., if $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ is a basis for $\mathcal{L}$, then each coset $\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})$ corresponds to a unique vector $a_1\boldsymbol{b}_1 + \cdots + a_n\boldsymbol{b}_n$ with $a_i \in \{0, 1\}$, and this correspondence is a bijection. Notice that the cosets in $\mathcal{L}/(2\mathcal{L})$ have a group structure under addition that is isomorphic to $\mathbb{Z}_2^n$.

2.2 The discrete Gaussian

For a parameter $s > 0$ and vector $\boldsymbol{x} \in \mathbb{R}^n$, we write

$$\rho_s(\boldsymbol{x}) := \exp(-\pi\|\boldsymbol{x}\|^2/s^2)$$

[3] After one step of "pairing and averaging," we know exactly the distribution that we get, and it *is* a weighted combination of Gaussians over the cosets of a certain sublattice! This seems quite auspicious. Unfortunately, the particular sublattice is not the same sublattice that we use to pair the vectors in the next step, and we therefore are unable to say much at all about what happens even after two steps.

for the *Gaussian mass of $\boldsymbol{x}$ with parameter $s > 0$*. Up to scaling, the Gaussian mass is the unique function on $\mathbb{R}^n$ that is invariant under rotations and a product function. In particular, it satisfies the following nice rotation identity,

$$\rho_s(\boldsymbol{x})\rho_s(\boldsymbol{y}) = \rho_{\sqrt{2}s}(\boldsymbol{x}+\boldsymbol{y})\rho_{\sqrt{2}s}(\boldsymbol{x}-\boldsymbol{y}) \tag{1}$$

for any parameter $s > 0$ and vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. This identity is fundamental to the results of [1, 2]. (See [32, 35] for a more detailed description of this connection and some additional results.)

We extend the Gaussian mass to a shift $\boldsymbol{t} \in \mathbb{R}^n$ of a lattice $\mathcal{L} \subset \mathbb{R}^n$ in the natural way,

$$\rho_s(\mathcal{L} - \boldsymbol{t}) := \sum_{\boldsymbol{y} \in \mathcal{L}} \rho_s(\boldsymbol{y} - \boldsymbol{t}) \ ,$$

and we call this the *Gaussian mass of $\mathcal{L} - \boldsymbol{t}$ with parameter s*.

We will need the following identity from [2]. (See [32, 35] for a much more general identity.)

▶ **Lemma 3.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t}$, and parameter $s > 0$, we have*

$$\sum_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})^2 = \rho_{s/\sqrt{2}}(\mathcal{L})\rho_{s/\sqrt{2}}(\mathcal{L} - \boldsymbol{t}) \ .$$

Proof. We have

$$\begin{aligned}
\sum_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})^2 &= \sum_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \sum_{\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathcal{L}} \rho_s(2\boldsymbol{y}_1 + \boldsymbol{c} - \boldsymbol{t})\rho_s(2\boldsymbol{y}_2 + \boldsymbol{c} - \boldsymbol{t}) \\
&= \sum_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \sum_{\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathcal{L}} \rho_{s/\sqrt{2}}(\boldsymbol{y}_1 + \boldsymbol{y}_2 + \boldsymbol{c} - \boldsymbol{t})\rho_{s/\sqrt{2}}(\boldsymbol{y}_1 - \boldsymbol{y}_2) \\
&= \sum_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \sum_{\boldsymbol{w}, \boldsymbol{y}_1 \in \mathcal{L}} \rho_{s/\sqrt{2}}(2\boldsymbol{y}_1 - \boldsymbol{w} + \boldsymbol{c} - \boldsymbol{t})\rho_{s/\sqrt{2}}(\boldsymbol{w}) \\
&= \rho_{s/\sqrt{2}}(\mathcal{L} - \boldsymbol{t}) \sum_{\boldsymbol{w} \in \mathcal{L}} \rho_{s/\sqrt{2}}(\boldsymbol{w}) \\
&= \rho_{s/\sqrt{2}}(\mathcal{L} - \boldsymbol{t})\rho_{s/\sqrt{2}}(\mathcal{L}) \ ,
\end{aligned}$$

as needed. ◀

2.3 Dominating distributions

Intuitively, we say that some random list $\mathcal{M} \in S^*$ dominates another random list $\mathcal{M}' \in S^*$ if for every fixed list $\mathcal{S} \in S^*$, "$\mathcal{M}$ is at least as likely to contain $\mathcal{S}$ as a subsequence as $\mathcal{M}'$ is."

▶ **Definition 4** (Dominating distribution). For some finite set S (which we identify with $\{1, \ldots, N\}$ without loss of generality) and two random lists $\mathcal{M} := (X_1, \ldots, X_M) \in S^*$ and $\mathcal{M}' := (X_1', \ldots, X_{M'}') \in S^*$ (where M and M' might themselves be random variables), we say that $\mathcal{M}$ *dominates* $\mathcal{M}'$ if for any $(k_1, \ldots, k_N) \in \mathbb{N}^N$,

$$\Pr[|\{j \ : \ X_j = i\}| \geq k_i, \ \forall i] \geq \Pr[|\{j \ : \ X_j' = i\}| \geq k_i, \ \forall i] \ .$$

We note the following basic facts about dominant distributions, which show that domination yields a partial order over random variables on S^*, and that this partial order behaves nicely under taking sublists.

▶ **Fact 5.** *For any finite set S and random variable $\mathcal{M} \in S^*$ that dominates some other random variable $\mathcal{M}' \in S^*$,*

1. *$\mathcal{M}$ dominates itself;*
2. *if $\mathcal{M}'$ dominates some random variable $\mathcal{M}'' \in S^*$, then $\mathcal{M}$ also dominates $\mathcal{M}''$; and*
3. *for any function $f : S^* \to S^*$ that maps a list of elements to a sublist, $\mathcal{M}$ dominates $f(\mathcal{M}')$.*

3 No need for rejection!

We now show our main observation: if $\boldsymbol{X}_1 \in \mathcal{L} - \boldsymbol{t}$ and $\boldsymbol{X}_2 \in \mathcal{L} - \boldsymbol{t}$ are sampled from the discrete Gaussian over a fixed coset $2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t}$ for some $\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})$, then their average $(\boldsymbol{X}_1+\boldsymbol{X}_2)/2$ is distributed as a mixture of Gaussians over the cosets $2\mathcal{L}+\boldsymbol{d}-\boldsymbol{t}$ for $\boldsymbol{d} \in \mathcal{L}/(2\mathcal{L})$ with parameter lowered by a factor of $\sqrt{2}$.

▶ **Lemma 6.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, parameter $s > 0$, coset $\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})$, $s > 0$, and $\boldsymbol{y} \in \mathcal{L}$, we have*

$$\Pr_{\boldsymbol{X}_1,\boldsymbol{X}_2 \sim D_{2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t},s}}[(\boldsymbol{X}_1 + \boldsymbol{X}_2)/2 = \boldsymbol{y} - \boldsymbol{t}] = \rho_{s/\sqrt{2}}(\boldsymbol{y} - \boldsymbol{t}) \cdot \frac{\rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{c} + \boldsymbol{y})}{\rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})^2} \ .$$

In particular, for any $\boldsymbol{d} \in \mathcal{L}/(2\mathcal{L})$ and $\boldsymbol{y} \in 2\mathcal{L} + \boldsymbol{d}$,

$$\Pr_{\boldsymbol{X}_1,\boldsymbol{X}_2 \sim D_{2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t},s}}[(\boldsymbol{X}_1+\boldsymbol{X}_2)/2 = \boldsymbol{y}-\boldsymbol{t} \mid (\boldsymbol{X}_1+\boldsymbol{X}_2)/2 \in 2\mathcal{L}+\boldsymbol{d}-\boldsymbol{t}] = \frac{\rho_{s/\sqrt{2}}(\boldsymbol{y} - \boldsymbol{t})}{\rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{d} - \boldsymbol{t})} \ .$$

Proof. We have

$$\begin{aligned}
&\rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})^2 \cdot \Pr_{\boldsymbol{X}_1,\boldsymbol{X}_2 \sim D_{2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t},s}}[(\boldsymbol{X}_1 + \boldsymbol{X}_2)/2 = \boldsymbol{y} - \boldsymbol{t}] \\
&\quad = \sum_{\boldsymbol{x} \in 2\mathcal{L}+\boldsymbol{c}} \rho_s(\boldsymbol{x} - \boldsymbol{t})\rho_s(2\boldsymbol{y} - \boldsymbol{x} - \boldsymbol{t}) \\
&\quad = \rho_{s/\sqrt{2}}(\boldsymbol{y} - \boldsymbol{t}) \sum_{\boldsymbol{x} \in 2\mathcal{L}+\boldsymbol{c}} \rho_{s/\sqrt{2}}(\boldsymbol{x} - \boldsymbol{y}) && \text{(Eq. (1))} \\
&\quad = \rho_{s/\sqrt{2}}(\boldsymbol{y} - \boldsymbol{t})\rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{c} + \boldsymbol{y}) \ ,
\end{aligned}$$

as needed. The "in particular" then follows from the fact that $\rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{c} + \boldsymbol{y}) = \rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{c} + \boldsymbol{d})$ is constant for $\boldsymbol{y} \in 2\mathcal{L} + \boldsymbol{d}$ for some fixed $\boldsymbol{d} \in \mathcal{L}/(2\mathcal{L})$. ◀

Lemma 6 motivates the following definition, which captures a key property of the distribution described in Lemma 6.

▶ **Definition 7.** For a lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, and parameter $s > 0$ we say that the random list $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M) \in (\mathcal{L} - \boldsymbol{t})^*$ is a *mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s* if the "distributions within the cosets of $2\mathcal{L}$" are independent Gaussians with parameter s. I.e., for any list of cosets $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_M) \in ((\mathcal{L} - \boldsymbol{t})/(2\mathcal{L}))^*$ mod $2\mathcal{L}$, if we *condition on* $\boldsymbol{X}_i \in 2\mathcal{L} + \boldsymbol{c}_i$ for all i, then the $\boldsymbol{X}_i$ are independent with $\boldsymbol{X}_i \sim D_{2\mathcal{L}+\boldsymbol{c}_i,s}$.

We call $(2\mathcal{L}+\boldsymbol{X}_1, \ldots, 2\mathcal{L}+\boldsymbol{X}_M)$ the *coset distribution* of the $\boldsymbol{X}_i$. We say that a mixture of independent Gaussians $\mathcal{M}$ over $\mathcal{L} - \boldsymbol{t}$ with parameter $s > 0$ dominates another, $\mathcal{M}'$, if the coset distribution of $\mathcal{M}$ dominates the coset distribution of $\mathcal{M}'$ (as in Definition 4).

In other words, mixtures of independent Gaussians are exactly the distributions obtained by first sampling $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_M) \in ((\mathcal{L} - \boldsymbol{t})/(2\mathcal{L}))^*$ from some arbitrary coset distributions and then sampling $\boldsymbol{X}_i \sim D_{\mathcal{L}+\boldsymbol{c}_i,s}$ independently for each i. We now list some basic facts that follow from what we have done so far.

▶ **Corollary 8** (Properties of mixtures of Gaussians and Procedure 1). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, and parameter $s > 0$,*

1. *a mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s is uniquely characterized by its coset distribution;*
2. *if we apply Procedure 1 to a mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s, the result will be a mixture of Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter $s/\sqrt{2}$;*
3. *Procedure 1 preserves domination – i.e., if we apply Procedure 1 to two mixtures $\mathcal{M}, \mathcal{M}'$ of Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s and $\mathcal{M}$ dominates $\mathcal{M}'$, then the output of Procedure 1 on input $\mathcal{M}$ will dominate that of $\mathcal{M}'$; and*
4. *if $\boldsymbol{X}_1, \boldsymbol{X}_2$ are a mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s with coset distribution given by $\boldsymbol{X}_1 \equiv \boldsymbol{X}_2 \bmod 2\mathcal{L}$ and*

$$\Pr[2\mathcal{L} + \boldsymbol{X}_1 + \boldsymbol{t} = \boldsymbol{c}] = \frac{\rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})^2}{\sum_{\boldsymbol{d} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{d} - \boldsymbol{t})^2}$$

for any $\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})$, then their average $(\boldsymbol{X}_1 + \boldsymbol{X}_2)/2$ is distributed exactly as $D_{\mathcal{L}-\boldsymbol{t}, s/\sqrt{2}}$.

Proof. Item 1 follows immediately from the definition of a mixture of Gaussians. Items 2 and 3 are immediate consequences of Lemma 6.

For Item 4, we apply Lemma 6 to see that for any $\boldsymbol{y} \in \mathcal{L}$,

$$\begin{aligned}\Pr[(\boldsymbol{X}_1 + \boldsymbol{X}_2)/2 = \boldsymbol{y} - \boldsymbol{t}] &= \frac{\rho_{s/\sqrt{2}}(\boldsymbol{y} - \boldsymbol{t})}{\sum_{\boldsymbol{d} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{d} - \boldsymbol{t})^2} \sum_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{c} + \boldsymbol{y}) \\ &= \frac{\rho_{s/\sqrt{2}}(\boldsymbol{y} - \boldsymbol{t})}{\sum_{\boldsymbol{d} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{d} - \boldsymbol{t})^2} \cdot \rho_{s/\sqrt{2}}(\mathcal{L}) \; .\end{aligned}$$

The result then follows from Lemma 3. (Indeed, summing the left-hand side and the right-hand side over all $\boldsymbol{y} \in \mathcal{L}$ gives a proof of Lemma 3.) ◀

In [1, 2], we performed a careful rejection sampling procedure f in Procedure 2 so that, at each step of the algorithm, the output was distributed exactly as $D_{\mathcal{L}-\boldsymbol{t}, s/2^{i/2}}$ (up to some small statistical distance). In particular, we applied the rejection sampling procedure guaranteed by Theorem 12 to obtain independent vectors distributed as in Item 4, which yield independent Gaussians with a lower parameter when combined as in Procedure 1. But, Corollary 8 makes this unnecessary. Indeed, Corollary 8 shows that "any collection of vectors that can be found with any rejection sampling procedure can be found without it." The following meta-theorem makes this formal.

▶ **Theorem 9.** *For any (possibly randomized) rejection function f mapping lists of cosets modulo $2\mathcal{L}$ to a subset of indices (as in Procedure 2), let $\mathcal{A}$ be the algorithm defined in Procedure 2. Let $\mathcal{A}'$ be the same algorithm with f replaced by the trivial function that just outputs all indices (i.e., $\mathcal{A}'$ just repeatedly runs Procedure 1 with no rejection).*

Then, for any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift vector $\boldsymbol{t} \in \mathbb{R}^n$, parameter $s > 0$, if $\mathcal{A}$ and $\mathcal{A}'$ are each called on input $\ell \geq 1$ and a list of $M \geq 2$ independent samples from $D_{\mathcal{L}-\boldsymbol{t},s}$, the resulting output distributions will be mixtures of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter $s/2^{\ell/2}$. Furthermore, the distribution corresponding to $\mathcal{A}'$ will dominate the distribution corresponding to $\mathcal{A}$. In particular, for any finite set $S \subset \mathcal{L} - \boldsymbol{t}$,

$$\Pr_{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M \sim D_{\mathcal{L}-\boldsymbol{t},s}}[S \subseteq \mathcal{A}(\ell, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)] \leq \Pr_{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M \sim D_{\mathcal{L}-\boldsymbol{t},s}}[S \subseteq \mathcal{A}'(\ell, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)] \; .$$

Proof. Notice that, since f only acts on the cosets of the $\boldsymbol{X}_i$, f "preserves mixtures of independent Gaussians." I.e., if $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{M'})$ is some mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter $s' > 0$ and $(j_1, \ldots, j_m) \leftarrow f(2\mathcal{L} + \boldsymbol{X}_1, \ldots, 2\mathcal{L} + \boldsymbol{X}_{M'})$, then $(\boldsymbol{X}_{j_1}, \ldots, \boldsymbol{X}_{j_m})$ is also a mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s'. (Notice that this would *not* be true if f acted on vectors, rather than cosets.) Similarly, by Item 2, Procedure 1 maps mixtures of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter s' to mixtures with parameter $s'/\sqrt{2}$. It follows that for both $\mathcal{A}$ and $\mathcal{A}'$, after the ith step of the algorithm, the list of vectors is a mixture of Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter $s/2^{i/2}$. And, the same holds after the application of f in algorithm $\mathcal{A}$. Therefore, the only question is the coset distributions.

By Fact 5, we see that $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$ dominates $(\boldsymbol{X}_{j_1}, \ldots, \boldsymbol{X}_{j_m})$. Therefore, by Item 3, the distribution of vectors corresponding to $\mathcal{A}'$ dominates the distribution of $\mathcal{A}$ after the first step. If we assume for induction that, after the $(i-1)$st step, the distribution of vectors corresponding to $\mathcal{A}'$ dominates the distribution corresponding to $\mathcal{A}$, then the exact same argument together with another application of Fact 5 shows that the same holds after step i. The result follows. ◀

Theorem 9, together with the corresponding algorithms in [1, 2], immediately implies Theorems 1 and 2. For completeness, we give more direct proofs of these theorems in the appendix, more-or-less recreating the corresponding proofs in [1, 2].

Acknowledgments. We thank Oded Regev and Daniel Dadush for many helpful discussions.

References

1 Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the Shortest Vector Problem in 2^n time via discrete Gaussian sampling. In *STOC*, 2015.

2 Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. Solving the Closest Vector Problem in 2^n time— The discrete Gaussian strikes again! In *FOCS*, 2015.

3 Miklós Ajtai. The shortest vector problem in L_2 is *NP*-hard for randomized reductions (extended abstract). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 10–19. ACM, 1998. doi:10.1145/276698.276705.

4 Miklós Ajtai. Generating hard instances of lattice problems. In *Complexity of computations and proofs*, volume 13 of *Quad. Mat.*, pages 1–32. Dept. Math., Seconda Univ. Napoli, Caserta, 2004. Preliminary version in STOC'96.

5 Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610. ACM, 2001. doi:10.1145/380752.380857.

6 Miklós Ajtai, Ravi Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *CCC*, pages 41–45, 2002.

7 Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange — A new hope. In *USENIX Security Symposium*, 2016.

8 László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. doi:10.1007/BF02579403.

9 Wojciech Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(4):625–635, 1993. doi:10.1007/BF01445125.

10 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, 2016.

11 Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In *CCS*, 2016.

12 Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584. ACM, 2013. `doi:10.1145/2488608.2488680`.

13 R. de Buda. Some optimal codes have structure. *Selected Areas in Communications, IEEE Journal on*, 7(6):893–899, Aug 1989.

14 Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003.

15 Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *STOC*, 2008.

16 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

17 Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Pierre Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, 1999. `doi:10.1016/S0020-0190(99)00083-6`.

18 Ishay Haviv and Oded Regev. Tensor-based hardness of the Shortest Vector Problem to within almost polynomial factors. *Theory of Computing*, 8(23):513–531, 2012. Preliminary version in STOC'07.

19 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. `doi:10.1287/moor.8.4.538`.

20 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. `doi:10.1287/moor.12.3.415`.

21 Subhash Khot. Hardness of approximating the Shortest Vector Problem in lattices. *Journal of the ACM*, 52(5):789–808, 2005. Preliminary version in FOCS'04.

22 Philip Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, 2000.

23 A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982. `doi:10.1007/BF01457454`.

24 Daniele Micciancio. The Shortest Vector Problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001. Preliminary version in FOCS 1998.

25 Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the Shortest Vector Problem. In *SODA*, 2010.

26 Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. *SIAM Journal on Computing*, 42(3):1364–1391, 2013.

27 Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Eurocrypt*, 2016.

28 NIST post-quantum standardization call for proposals. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/cfp-announce-dec2016.html`, 2016. Accessed: 2017-04-02.

29 Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

30 Xavier Pujol and Damien Stehlé. Solving the Shortest Lattice Vector Problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009.

31 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. `doi:10.1145/1568318.1568324`.

32 Oded Regev and Noah Stephens-Davidowitz. An inequality for Gaussians on lattices. *SIDMA*, 2017.

33 Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987. doi:10.1016/0304-3975(87)90064-8.

34 Adi Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Trans. Information Theory*, 30(5):699–704, 1984. doi:10.1109/TIT.1984.1056964.

35 Noah Stephens-Davidowitz. *On the Gaussian Measure Over Lattices.* PhD thesis, New York University, 2017.

36 Peter van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical report, University of Amsterdam, Department of Mathematics, Netherlands, 1981. Technical Report 8104.

A Additional preliminaries

We will need some additional preliminaries. We write

$$\lambda_1(\mathcal{L}) := \min_{\boldsymbol{y} \in \mathcal{L} \setminus \{\boldsymbol{0}\}} \|\boldsymbol{y}\|$$

for the length of the shortest non-zero vector in the lattice. And, for a target vector $\boldsymbol{t} \in \mathbb{R}^n$, we write

$$\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) := \min_{\boldsymbol{y} \in \mathcal{L}} \|\boldsymbol{y} - \boldsymbol{t}\|$$

for the distance from $\boldsymbol{t}$ to the lattice. Notice that this is the same as the length of the shortest vector in $\mathcal{L} - \boldsymbol{t}$.

A.1 Some known algorithms

We will need the famous result of Lenstra, Lenstra, and Lovász [23].

▶ **Theorem 10** ([23]). *There is an efficient algorithm that take as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and outputs $\tilde{\lambda} > 0$ with*

$$\lambda_1(\mathcal{L}) \le \tilde{d} \le 2^{n/2} \lambda_1(\mathcal{L}) \; .$$

We will also need the following celebrated result due to Babai [8].

▶ **Theorem 11** ([8]). *There is an efficient algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a target $\boldsymbol{t} \in \mathbb{R}^n$ and outputs $\tilde{d} > 0$ with*

$$\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) \le \tilde{d} \le 2^{n/2} \operatorname{dist}(\boldsymbol{t}, \mathcal{L}) \; .$$

A.2 The distribution of disjoint pairs

Recall that Procedure 1 takes the T_i elements from the ith coset and converts them into $\lfloor T_i/2 \rfloor$ disjoint pairs. Therefore, for a list $\mathcal{M} := (X_1, \ldots, X_M) \in S^*$ over some finite set S, we write $\lfloor \mathcal{M}/2 \rfloor$ for the random variable obtained as in Procedure 1. I.e., up to ordering (which does not concern us), $\lfloor \mathcal{M}/2 \rfloor := (X'_1, \ldots, X'_{M'}) \in (S \times S)^*$ is defined by

$$|\{j \; : \; X'_j = (s,s)\}| = \lfloor |\{j \; : \; X_j = s\}|/2 \rfloor$$

for each $s \in S$.

▶ **Theorem 12** ([1, Theorem 3.3]). *For any probabilities, $p_1, \ldots, p_N \in [0,1]$ with $\sum p_i = 1$, integer M, and $\kappa \geq \Omega(\log M)$ (the confidence parameter) with $M \geq 10\kappa^2/p_{\mathsf{max}}$, let $\mathcal{M} = (X_1, \ldots, X_M) \in \{1, \ldots, N\}^M$ be the distribution obtained by sampling each X_j independently from the distribution that assigns to element i probability p_i. Then, there exists a rejection sampling procedure that, up to statistical distance $\exp(-\Omega(\kappa))$, maps $\mathcal{M}$ to the distribution $\mathcal{M}' := (X_1', \ldots, X_M') \in \{(1,1), \ldots, (N,N)\}^{M'}$ obtained by sampling each pair X_j independently from the distribution that assigns to the pair (i,i) probability p_i^2/p_{col}, where*

$$M' := \left\lceil M \cdot \frac{p_{\mathsf{col}}}{32\kappa p_{\mathsf{max}}} \right\rceil ,$$

$p_{\mathsf{max}} := \max p_i$, and $p_{\mathsf{col}} := \sum p_i^2$.

▶ **Corollary 13.** *For any probabilities, $p_1, \ldots, p_N \in [0,1]$ with $\sum p_i = 1$, integer M, and $\kappa \geq \Omega(\log M)$ (the confidence parameter) with $M \geq 10\kappa^2/p_{\mathsf{max}}$, let $\mathcal{M} := (X_1, \ldots, X_M) \in \{1, \ldots, N\}^M$ be the distribution obtained by sampling each X_j independently from the distribution that assigns to element i probability p_i. Let $\mathcal{M}' := (X_1', \ldots, X_{M'}') \in \{(1,1), \ldots, (N,N)\}^{M'}$ be the distribution obtained by sampling each pair X_j' independently from the distribution that assigns to the pair (i,i) probability p_i^2/p_{col}, where*

$$M' := \left\lceil M \cdot \frac{p_{\mathsf{col}}}{32\kappa p_{\mathsf{max}}} \right\rceil ,$$

$p_{\mathsf{max}} := \max p_i$, and $p_{\mathsf{col}} := \sum p_i^2$. Then, $\mathcal{M}$ dominates $\mathcal{M}'$.

A.3 Additional facts about the discrete Gaussian

We will also need some additional facts about the discrete Gaussian.

▶ **Lemma 14** ([9]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, parameter $s \geq 1$, and shift $\boldsymbol{t} \in \mathbb{R}^n$, $\rho_s(\mathcal{L} - \boldsymbol{t}) \leq s^n \rho(\mathcal{L})$.*

The following theorem shows that, if the parameter s is appropriately small, then $D_{\mathcal{L}-\boldsymbol{t},s} + \boldsymbol{t}$ will be an approximate closest vector to $\boldsymbol{t}$, with approximation factor roughly $1 + \sqrt{n}s/\operatorname{dist}(\boldsymbol{t}, \mathcal{L})$. (This is a basic consequence of Banaszczyk's celebrated theorem [9].)

▶ **Proposition 15** ([35, Corollary 1.3.11], see also [2, Corollary 2.8]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, parameter $s > 0$, shift $\boldsymbol{t} \in \mathbb{R}^n$, and radius $r > \sqrt{n/(2\pi)} \cdot s$, with $r > \operatorname{dist}(\boldsymbol{t}, \mathcal{L})$ and*

$$r^2 > \operatorname{dist}(\boldsymbol{t}, \mathcal{L})^2 + \frac{ns^2}{\pi} \cdot \log(2\pi \operatorname{dist}(\boldsymbol{t}, \mathcal{L})^2/(ns^2)) ,$$

we have

$$\Pr_{\boldsymbol{X} \sim D_{\mathcal{L}-\boldsymbol{t},s}}[\|\boldsymbol{X}\| > r] < (2e)^{n/2+1} \exp(-\pi y^2/2) ,$$

where $y := \sqrt{r^2 - \operatorname{dist}(\boldsymbol{t}, \mathcal{L})^2}/s$.

The next theorem shows that exponentially many samples from $D_{\mathcal{L},s}$ with $s \approx \lambda_1(\mathcal{L})/\sqrt{n}$ is sufficient to find a shortest non-zero lattice vector.

▶ **Proposition 16** ([1, Proposition 4.3]). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, and parameter*

$$s := \sqrt{2^{0.198}\pi e/n} \cdot \lambda_1(\mathcal{L}) ,$$

we have

$$\Pr_{\boldsymbol{X} \sim D_{\mathcal{L},s}}[\|\boldsymbol{X}\| = \lambda_1(\mathcal{L})] \geq 1.38^{-n-o(n)} .$$

The next corollary follows immediately from Proposition 16 and Lemma 14.

▶ **Corollary 17.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, and parameter*

$$\sqrt{2^{0.198}\pi e/n} \cdot \lambda_1(\mathcal{L}) \le s \le 1.01 \cdot \sqrt{2^{0.198}\pi e/n} \cdot \lambda_1(\mathcal{L}) \ ,$$

we have

$$\Pr_{\boldsymbol{X} \sim D_{\mathcal{L},s}}[\|\boldsymbol{X}\| = \lambda_1(\mathcal{L})] \ge 1.4^{-n-o(n)} \ .$$

We will also need the following result from [2], which is an immediate consequence of the main identity in [32]. (See also [35].)

▶ **Lemma 18** ([2, Corollary 3.3])**.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, and parameter $s > 0$, we have*

$$\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})^2 \le \rho_{s/\sqrt{2}}(\mathcal{L}) \max_{\boldsymbol{c} \in \mathcal{L}} \rho_{s/\sqrt{2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t}) \ .$$

From this, we derive the following rather technical-looking inequality, which is implicit in [2]. (This inequality comes up naturally in the proof of Corollary 21. We separate it out here to make that proof cleaner.)

▶ **Corollary 19.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, parameter $s > 0$, and integer $\ell \ge 0$, we have*

$$\prod_{i=0}^{\ell-1} \frac{\rho_{s/2^{(i+1)/2}}(\mathcal{L} - \boldsymbol{t})\rho_{s/2^{(i+1)/2}}(\mathcal{L})}{\rho_{s/2^{i/2}}(\mathcal{L} - \boldsymbol{t}) \cdot \max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}$$
$$\ge \frac{\rho_{s/2^{\ell/2}}(\mathcal{L} - \boldsymbol{t})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{\ell/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \cdot \frac{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}{\rho_s(\mathcal{L} - \boldsymbol{t})} \ .$$

Proof. From Lemma 18, we see that for all i,

$$\frac{\rho_{s/2^{(i+1)/2}}(\mathcal{L})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \ge \frac{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{(i+1)/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \ .$$

Therefore, the product in the statement of the corollary is at least

$$\prod_{i=0}^{\ell-1} \frac{\rho_{s/2^{(i+1)/2}}(\mathcal{L} - \boldsymbol{t}) \cdot \max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}{\rho_{s/2^{i/2}}(\mathcal{L} - \boldsymbol{t}) \cdot \max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{(i+1)/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}$$
$$= \frac{\rho_{s/2^{\ell/2}}(\mathcal{L} - \boldsymbol{t})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{\ell/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \cdot \frac{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}{\rho_s(\mathcal{L} - \boldsymbol{t})} \ ,$$

where we have used the fact that this is a telescoping product. ◀

B Running Procedure 1 on Gaussian input

▶ **Theorem 20.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, parameter $s > 0$, integer M, and confidence parameter $\kappa \ge \Omega(\log M)$, if $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$ are sampled independently from $D_{\mathcal{L}-\boldsymbol{t},s}$ with*

$$M \ge 10\kappa^2 \cdot \frac{\rho_s(\mathcal{L} - \boldsymbol{t})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \ ,$$

then the output of Procedure 1 applied to the $\boldsymbol{X}_i$ will be a mixture of independent Gaussians with parameter $s/\sqrt{2}$ that dominates the distribution of

$$M' := \left\lceil \frac{M}{32\kappa} \cdot \frac{\rho_{s/\sqrt{2}}(\mathcal{L}-\boldsymbol{t}) \cdot \rho_{s/\sqrt{2}}(\mathcal{L})}{\rho_s(\mathcal{L}-\boldsymbol{t}) \cdot \max_{\boldsymbol{d}\in\mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L}+\boldsymbol{d}-\boldsymbol{t})} \right\rceil$$

independent samples from $D_{\mathcal{L}-\boldsymbol{t},s/\sqrt{2}}$, up to statistical distance $\exp(-\Omega(\kappa))$.

Proof. By Item 2 of Corollary 8, the resulting distribution will in fact be a mixture of independent Gaussians over $\mathcal{L}-\boldsymbol{t}$ with parameter $s/\sqrt{2}$. Notice that, if $\mathcal{M}$ is the coset distribution of $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$, then Procedure 1 first maps the $\boldsymbol{X}_i$ into the mixture of independent Gaussians over $\mathcal{L}-\boldsymbol{t}$ with parameter s and coset distribution $\lfloor \mathcal{M}/2 \rfloor$ and then takes the averages of the corresponding pairs of these vectors.

We wish to apply Corollary 13 over the coset distribution, with the probabilities $p_i := p_{2\mathcal{L}+\boldsymbol{c}}$ taken to be the weights of the cosets in the original distribution discrete Gaussian,

$$p_{2\mathcal{L}+\boldsymbol{c}} := \frac{\rho_s(2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t})}{\rho_s(\mathcal{L}-\boldsymbol{t})} \ .$$

Notice that, by Lemma 3,

$$M' = \left\lceil M \cdot \frac{p_{\mathsf{col}}}{32\kappa p_{\max}} \right\rceil \ ,$$

which is exactly what is needed to apply Corollary 13. By the corollary, up to statistical distance $\exp(-\Omega(\kappa))$ this distribution dominates the mixture of independent Gaussians over $\mathcal{L}-\boldsymbol{t}$ with parameter s whose coset distribution is given by $\boldsymbol{c}_{2k-1} = \boldsymbol{c}_{2k}$ for $1 \le k \le M'$, with the odd-indexed cosets $\boldsymbol{c}_{2k-1}$ sampled independently from the distribution that assigns to coset $\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})$ probability

$$\frac{p_i}{p_{\mathsf{col}}} = \frac{\rho_s(2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t})^2}{\sum_{\boldsymbol{d}\in\mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L}+\boldsymbol{d}-\boldsymbol{t})^2} \ .$$

Notice that this "squared" distribution" (so-called because the cosets are given weight proportional to their square) is simply M' independent copies of the distribution from Item 4 of Corollary 8. So, if we run Procedure 1 on this "squared" distribution, the output will be exactly M' independent samples from $D_{\mathcal{L}-\boldsymbol{t},s/\sqrt{2}}$.

Finally, by Fact 5, we see that, since the actual pairs dominate these "squared" pairs (up to statistical distance $\exp(-\Omega(\kappa))$), the output must dominate M' independent samples from $D_{\mathcal{L}-\boldsymbol{t},s/\sqrt{2}}$. ◀

▶ **Corollary 21.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, parameter $s > 0$, integer $M \ge 2$, and confidence parameter $\kappa \ge \Omega(\log M)$, if $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$ are sampled independently from $D_{\mathcal{L}-\boldsymbol{t},s}$ with*

$$M \ge (10\kappa)^{2\ell} \cdot \frac{\rho_s(\mathcal{L}-\boldsymbol{t})}{\max_{\boldsymbol{c}\in\mathcal{L}/(2\mathcal{L})} \rho_s(2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t})} \ ,$$

and we apply Procedure 1 repeatedly to the $\boldsymbol{X}_i$ a total of $\ell \ge 1$ times, the result will be a mixture of independent Gaussians with parameter $s/2^{\ell/2}$ that dominates the distribution of

$$M' := \left\lceil \frac{M}{(32\kappa)^\ell} \cdot \prod_{i=0}^{\ell-1} \frac{\rho_{s/2^{(i+1)/2}}(\mathcal{L}-\boldsymbol{t})\rho_{s/2^{(i+1)/2}}(\mathcal{L})}{\rho_{s/2^{i/2}}(\mathcal{L}-\boldsymbol{t}) \cdot \max_{\boldsymbol{c}\in\mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L}+\boldsymbol{c}-\boldsymbol{t})} \right\rceil$$

independent samples from $D_{\mathcal{L}-\boldsymbol{t},s/2^{\ell/2}}$, up to statistical distance $\ell\exp(-\Omega(\kappa))$.

Proof. By Item 2 of Corollary 8, the output will in fact be a mixture of independent Gaussians over $\mathcal{L} - \boldsymbol{t}$ with parameter $s/2^{\ell/2}$. The only question is what the coset distribution is.

To show that the coset distribution is as claimed, the idea is to simply apply Theorem 20 ℓ times. In particular, we prove the result via induction on ℓ. When $\ell = 1$, this is exactly Theorem 20. For $\ell > 1$, we assume the statement is true for $\ell - 1$. In particular, before applying Procedure 1 the ℓth time, we have a mixture of independent Gaussians with parameter $s/2^{\ell/2}$ that dominates

$$\widehat{M} := \left\lceil \frac{M}{(32\kappa)^\ell} \cdot \prod_{i=0}^{\ell-2} \frac{\rho_{s/2^{(i+1)/2}}(\mathcal{L} - \boldsymbol{t})\rho_{s/2^{(i+1)/2}}(\mathcal{L})}{\rho_{s/2^{i/2}}(\mathcal{L} - \boldsymbol{t}) \cdot \max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \right\rceil$$

$$\geq 10\kappa^2 \cdot \frac{\rho_s(\mathcal{L} - \boldsymbol{t})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_s(\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \cdot \prod_{i=0}^{\ell-2} \frac{\rho_{s/2^{(i+1)/2}}(\mathcal{L} - \boldsymbol{t})\rho_{s/2^{(i+1)/2}}(\mathcal{L})}{\rho_{s/2^{i/2}}(\mathcal{L} - \boldsymbol{t}) \cdot \max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{i/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})}$$

independent Gaussians up to statistical distance $(\ell - 1)\exp(-\Omega(\kappa))$.

By Fact 5, it suffices to prove that the output of Procedure 1 on these $\widehat{M}$ samples dominates M' independent samples from $D_{\mathcal{L}-\boldsymbol{t}, s/2^{\ell/2}}$ up to statistical distance $\exp(-\Omega(\kappa))$. Indeed, this is exactly what Theorem 20 says, provided that

$$\widehat{M} \geq 10\kappa^2 \cdot \frac{\rho_{s/2^{(\ell-1)/2}}(\mathcal{L} - \boldsymbol{t})}{\max_{\boldsymbol{c} \in \mathcal{L}/(2\mathcal{L})} \rho_{s/2^{(\ell-1)/2}}(2\mathcal{L} + \boldsymbol{c} - \boldsymbol{t})} \ .$$

And, this inequality follows immediately from Corollary 19 together with the assumed lower bound on $\widehat{M}$. ◀

C The initial distribution

The following theorem was proven by Ajtai, Kumar, and Sivakumar [5], building on work of Schnorr [33].

▶ **Theorem 22** ([33, 5]). *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and $u \geq 2$ and outputs an $u^{n/y}$-reduced basis of $\mathcal{L}$ in time $\exp(O(u)) \cdot \mathrm{poly}(n)$, where we say that a basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of a lattice $\mathcal{L}$ is γ-reduced for some $\gamma \geq 1$ if*

1. *$\|\boldsymbol{b}_1\| \leq \gamma \cdot \lambda_1(\mathcal{L})$; and*
2. *$\pi_{\{\boldsymbol{b}_1\}^\perp}(\boldsymbol{b}_2), \ldots, \pi_{\{\boldsymbol{b}_1\}^\perp}(\boldsymbol{b}_n)$ is a γ-reduced basis of $\pi_{\{\boldsymbol{b}_1\}^\perp}(\mathcal{L})$.*

This next theorem is originally due to [16], based on analysis of an algorithm originally studied by Klein [22]. We present a slightly stronger version due to [12] for convenience.

▶ **Theorem 23** ([12, Lemma 2.3]). *There is a probabilistic polynomial-time algorithm that takes as input a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ with $n \geq 2$, a shift $\boldsymbol{t} \in \mathbb{R}^n$, and $\hat{s} > C\sqrt{\log n} \cdot \|\widetilde{\mathbf{B}}\|$ and outputs a vector that is distributed exactly as $D_{\mathcal{L}-\boldsymbol{t}, \hat{s}}$, where $\|\widetilde{\mathbf{B}}\| := \max \|\widetilde{\boldsymbol{b}}_i\|$.*

▶ **Proposition 24** ([2, Proposition 4.5]). *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$, shift $\boldsymbol{t} \in \mathbb{R}^n$, $r > 0$, and parameter $u \geq 2$, such that if*

$$r \geq u^{n/u}(1 + \sqrt{n} u^{n/u}) \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \ ,$$

then the output of the algorithm is $\boldsymbol{y} \in \mathcal{L}$ and a basis $\mathbf{B}'$ of a (possibly trivial) sublattice $\mathcal{L}' \subseteq \mathcal{L}$ such that all vectors from $\mathcal{L} - \boldsymbol{t}$ of length at most $r/u^{n/u} - \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$ are also contained in $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$, and $\|\widetilde{\mathbf{B}}'\| \leq r$. The algorithm runs in time $\mathrm{poly}(n) \cdot 2^{O(u)}$.

Proof. On input a lattice $\mathcal{L} \subset \mathbb{R}^n$, $\boldsymbol{t} \in \mathbb{R}^n$, and $r > 0$, the algorithm behaves as follows. First, it calls the procedure from Theorem 22 to compute a $u^{n/u}$-HKZ basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of $\mathcal{L}$. Let $(\widetilde{\boldsymbol{b}}_1, \ldots, \widetilde{\boldsymbol{b}}_n)$ be the corresponding Gram-Schmidt vectors. Let $k \geq 0$ be maximal such that $\|\widetilde{\boldsymbol{b}}_i\| \leq r$ for $1 \leq i \leq k$, and let $\mathbf{B}' = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k)$. Let $\pi_k = \pi_{\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}^\perp}$ and $\mathcal{M} = \pi_k(\mathcal{L})$. The algorithm then calls the procedure from Theorem 22 again with the same s and input $\pi_k(\boldsymbol{t})$ and $\mathcal{M}$, receiving as output $\boldsymbol{x} = \sum_{i=k+1}^n a_i \pi_k(\boldsymbol{b}_i)$ where $a_i \in \mathbb{Z}$, a $\sqrt{n}u^{n/u}$-approximate closest vector to $\pi_k(\boldsymbol{t})$ in $\mathcal{M}$. Finally, the algorithm returns $\boldsymbol{y} = -\sum_{i=k+1}^n a_i \boldsymbol{b}_i$ and $\mathbf{B}' = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k)$.

The running time is clear, as is the fact that $\|\widetilde{\mathbf{B}'}\| \leq r$. It remains to prove that $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$ contains all sufficiently short vectors in $\mathcal{L} - \boldsymbol{t}$. If $k = n$, then $\mathcal{L}' = \mathcal{L}$ and $\boldsymbol{y}$ is irrelevant, so we may assume that $k < n$. Note that, since $\mathbf{B}$ is a $u^{n/u}$-HKZ basis, $\lambda_1(\mathcal{M}) \geq \|\widetilde{\boldsymbol{b}}_{k+1}\|/u^{n/u} > r/u^{n/u}$. In particular, $\lambda_1(\mathcal{M}) > (1 + \sqrt{n} \cdot u^{n/u}) \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \geq (1 + \sqrt{n} \cdot u^{n/u}) \cdot \mathrm{dist}(\pi_k(\boldsymbol{t}), \mathcal{M})$. So, there is a unique closest vector to $\pi_k(\boldsymbol{t})$ in $\mathcal{M}$, and by triangle inequality, the next closest vector is at distance greater than $\sqrt{n} \cdot u^{n/u}\, \mathrm{dist}(\pi_k(\boldsymbol{t}), \mathcal{M})$. Therefore, the call to the subprocedure from Theorem 22 will output the exact closest vector $\boldsymbol{x} \in \mathcal{M}$ to $\pi_k(\boldsymbol{t})$.

Let $\boldsymbol{w} \in \mathcal{L} \setminus (\mathcal{L}' - \boldsymbol{y})$ so that $\pi_k(\boldsymbol{w}) \neq \pi_k(-\boldsymbol{y}) = \boldsymbol{x}$. We need to show that $\boldsymbol{w} - \boldsymbol{t}$ is relatively long. Since $\mathbf{B}$ is a $s^{n/s}$-HKZ basis, it follows that

$$\|\pi_k(\boldsymbol{w}) - \boldsymbol{x}\| \geq \lambda_1(\mathcal{M}) > r/u^{n/u} \; .$$

Applying triangle inequality, we have

$$\|\boldsymbol{w} - \boldsymbol{t}\| \geq \|\pi_k(\boldsymbol{w}) - \pi_k(\boldsymbol{t})\| \geq \|\pi_k(\boldsymbol{w}) - \boldsymbol{x}\| - \|\boldsymbol{x} - \pi_k(\boldsymbol{t})\| > r/u^{n/u} - \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \; ,$$

as needed. ◀

▶ **Corollary 25** ([2, Corollary 4.6])**.** *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ with $n \geq 2$, shift $\boldsymbol{t} \in \mathbb{R}^n$, $M \in \mathbb{N}$ (the desired number of output vectors), and parameters $u \geq 2$ and $\hat{s} > 0$ and outputs $\boldsymbol{y} \in \mathcal{L}$, a (possibly trivial) sublattice $\mathcal{L}' \subseteq \mathcal{L}$, and M vectors from $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$ such that if*

$$\hat{s} \geq 10\sqrt{n \log n} \cdot u^{2n/u} \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \; ,$$

then the output vectors are distributed as M independent samples from $D_{\mathcal{L}'-\boldsymbol{y}-\boldsymbol{t},\hat{s}}$, and $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$ contains all vectors in $\mathcal{L} - \boldsymbol{t}$ of length at most $\hat{s}/(10u^{n/u}\sqrt{\log n})$. The algorithm runs in time $\mathrm{poly}(n) \cdot 2^{O(u)} + \mathrm{poly}(n) \cdot M$. (And, if $\boldsymbol{t} = \boldsymbol{0}$, then $\boldsymbol{y} = \boldsymbol{0}$.)

Proof. The algorithm first calls the procedure from Proposition 24 with input $\mathcal{L}$, $\boldsymbol{t}$, and

$$r := \frac{10\hat{s}}{\sqrt{\log n}} \geq u^{n/u}(1 + \sqrt{n}u^{n/u}) \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \; ,$$

receiving as output $\boldsymbol{y} \in \mathcal{L}$ and a basis $\mathbf{B}'$ of a sublattice $\mathcal{L}' \subset \mathcal{L}$. It then runs the algorithm from Theorem 23 M times with input $\mathcal{L}'$, $\boldsymbol{y} + \boldsymbol{t}$, and $\hat{s}$ and outputs the resulting vectors, $\boldsymbol{y}$, and $\mathcal{L}'$.

The running time is clear. By Proposition 24, $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$ contains all vectors of length at most $r/u^{n/u} - \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \geq \hat{s}/(10u^{n/u}\sqrt{\log n})$ in $\mathcal{L} - \boldsymbol{t}$, and $\|\widetilde{\mathbf{B}}'\| \leq r \leq C\hat{s}/\sqrt{\log n}$. So, it follows from Theorem 23 that the output has the correct distribution. ◀

D Finishing the proof

▶ **Theorem 26** (SVP algorithm)**.** *For any lattice $\mathcal{L} \subset \mathbb{R}^n$, the output of Procedure 3 on input $\mathcal{L}$ will be a shortest non-zero vector in $\mathcal{L}$ except with probability at most $\exp(-\Omega(n))$.*

Procedure 3: The final $2^{n+o(n)}$-time SVP algorithm. Here $M = 2^{n+\Theta(\log^2 n)}$, $u = \Theta(n)$, and $\ell = \Theta(\log n)$.

```
SVP ($\mathcal{L}$)
Input  : A lattice $\mathcal{L} \subset \mathbb{R}^n$
Output : A vector $\boldsymbol{y} \in \mathcal{L}$ with $\|\boldsymbol{y}\| = \lambda_1(\mathcal{L})$
Use the procedure from Thereom 10 to compute $\widehat{\lambda}$ with $\lambda_1(\mathcal{L}) \le \widehat{\lambda} \le 2^{n/2}\lambda_1(\mathcal{L})$.
for $i = 1, \ldots, 200n$ do
    Set $\mathcal{L}' \subseteq \mathcal{L}$ and $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M \in \mathcal{L}$ to be the output of Corollary 25 on input $\mathcal{L}$,
     $\boldsymbol{t} := \boldsymbol{0}$, $u$, and $s_i := 1.01^{-i} \cdot \widehat{\lambda}$.
    for $j = 1, \ldots, \ell$ do
        $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{M'}) \leftarrow$ Pair_and_Average$(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$
        $M \leftarrow M'$
    end
    $\boldsymbol{Y}_i \leftarrow \arg\min_{\boldsymbol{X}_j \neq \boldsymbol{0}} \|\boldsymbol{X}_j\|$.
end
Output $\arg\min \|\boldsymbol{Y}_i\|$.
```

Proof. The running time is clear. Let $\kappa = \Theta(n)$. Let i such that $s_i/2^{\ell/2}$ satisfies the inequality in Corollary 17. By Corollary 25, the $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$ corresponding to this i will be distributed exactly as $D_{\mathcal{L}',s_i}$ where $\mathcal{L}' \subseteq \mathcal{L}$ contains all vectors of length at most $\lambda_1(\mathcal{L})$. So, $\lambda_1(\mathcal{L}') = \lambda_1(\mathcal{L})$, and it suffices to argue that we will find a shortest vector in $\mathcal{L}'$. By Corollary 21, the output distribution $(X_1, \ldots, X_M)$ will be a mixture of independent Gaussians over $\mathcal{L}'$ with parameter $s_i/2^{\ell/2}$ that dominates the distribution of

$$M' = \left\lceil \frac{M}{(32\kappa)^\ell} \cdot \prod_{j=0}^{\ell-1} \frac{\rho_{s_i/2^{(j+1)/2}}(\mathcal{L}')^2}{\rho_{s_i/2^{j/2}}(\mathcal{L}') \cdot \rho_{s_i/2^{(j+2)/2}}(\mathcal{L}')} \right\rceil$$

independent samples from $D_{\mathcal{L}',s_i/2^{\ell/2}}$ up to statistical distance $\exp(-\Omega(\kappa))$, where we have applied Lemma 14 to show that the coset with maximal mass is the central coset. Noting that this product is telescoping, we have

$$M' = \left\lceil \frac{M}{(32\kappa)^\ell} \cdot \frac{\rho_{s_i/\sqrt{2}}(\mathcal{L}')}{\rho_{s_i}(\mathcal{L}')} \cdot \frac{\rho_{s_i/2^{\ell/2}}(\mathcal{L}')}{\rho_{s_i/2^{(\ell+1)/2}}(\mathcal{L}')} \right\rceil \ge 2^{n/2} \ ,$$

where we have applied Lemma 14. The result then follows from Corollary 17, together with the fact that $\sqrt{2} > 1.4$. ◀

Procedure 4: The final $2^{n+o(n)}$-time SVP algorithm. Here $M = 2^{n+\Theta(n/\log n)}$, $u = \Theta(n)$, and $\ell = \Theta(n/\log^2 n)$.

```
CVP $(\mathcal{L}, \boldsymbol{t})$
Input  : A lattice $\mathcal{L} \subset \mathbb{R}^n$ and target $\boldsymbol{t} \in \mathbb{R}^n$
Output : A vector $\boldsymbol{y} \in \mathcal{L}$ with $\|\boldsymbol{y} - \boldsymbol{t}\| \le (1 + 2^{-n/\log^2 n}) \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$
Use the procedure from Thereom 11 to compute $\hat{d}$ with
 $\mathrm{dist}(\mathcal{L}, \boldsymbol{t}) \le \hat{d} \le 2^{n/2}\,\mathrm{dist}(\boldsymbol{t}, \mathcal{L})$.
for $i = 1, \ldots, n$ do
    Set $\mathcal{L}' \subseteq \mathcal{L}$, $\boldsymbol{y} \in \mathcal{L}$, and $X_1, \ldots, X_M \in \mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$ to be the output of Corollary 25
     on input $\mathcal{L}$, $\boldsymbol{t}$, $u$, and $s_i := 20n^2 \cdot 2^{-i} \cdot \hat{d}$.
    for $j = 1, \ldots, \ell$ do
        $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{M'}) \leftarrow$ Pair_and_Average$(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$
        $M \leftarrow M'$
    end
    $\boldsymbol{Y}_i \leftarrow \arg\min_{\boldsymbol{X}_j} \|\boldsymbol{X}_j\|$.
end
Output $\boldsymbol{t} + \arg\min \|\boldsymbol{Y}_i\|$.
```

▶ **Theorem 27** (CVP algorithm). *For any lattice $\mathcal{L} \subset \mathbb{R}^n$ and $\boldsymbol{t} \in \mathbb{R}^n$, the output of Procedure 4 on input $\mathcal{L}$ and $\boldsymbol{t}$ will a vector $\boldsymbol{y} \in \mathcal{L}$ with $\|\boldsymbol{y} - \boldsymbol{t}\| \le (1 + \exp(-\Omega(n/\log^2 n))) \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$, except with probability at most* $\exp(-\Omega(n))$.[4]

Proof. The running time is clear. Let $\kappa = \Theta(n)$. Let i such that

$$10\sqrt{n \log n} \cdot u^{2n/u} \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \le s_i \le 20\sqrt{n \log n} \cdot u^{2n/u} \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \; .$$

By Corollary 25, the $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M)$ corresponding to this i will be distributed exactly as $D_{\mathcal{L}'-\boldsymbol{y}-\boldsymbol{t}, s_i}$ where $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t} \subseteq \mathcal{L} - \boldsymbol{t}$ contains all vectors of length at most $\mathrm{dist}(\boldsymbol{t}, \mathcal{L})$. So, it suffices to argue that we will find a $(1 + 2^{-n/\log^2 n})$-approximate shortest vector in $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$. By Corollary 21, the output distribution $(X_1, \ldots, X_M)$ will be a mixture of independent Gaussians over $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$ with parameter $s_i/2^{\ell/2}$ that dominates the distribution of

$$M' = \left\lceil \frac{M}{(32\kappa)^\ell} \cdot \prod_{j=0}^{\ell-1} \frac{\rho_{s_i/2^{(j+1)/2}}(\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t})\rho_{s_i/2^{(j+1)/2}}(\mathcal{L})}{\rho_{s_i/2^{j/2}}(\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}) \cdot \max_{\boldsymbol{c} \in \mathcal{L}'/(2\mathcal{L}')} \rho_{s_i/2^{j/2}}(2\mathcal{L}' + \boldsymbol{c} - \boldsymbol{y} - \boldsymbol{t})} \right\rceil \ge 1$$

independent samples from $D_{\mathcal{L}'-\boldsymbol{y}-\boldsymbol{t}, s_i/2^{\ell/2}}$ up to statistical distance $\exp(-\Omega(\kappa))$, where we have applied Corollary 19.

Notice that $s_i/2^{\ell/2} < \exp(-\Omega(n/\log^2 n))\,\mathrm{dist}(\boldsymbol{t}, \mathcal{L})$. The result then follows from Proposition 15, which says that, except with probability $\exp(-\Omega(n))$ a sample from $D_{\mathcal{L}'-\boldsymbol{y}-\boldsymbol{t}, s_i/2^{\ell/2}}$ will be a $(1 + \exp(-\Omega(n/\log^2 n)))$-approximate shortest vector in $\mathcal{L}' - \boldsymbol{y} - \boldsymbol{t}$. ◀

[4] It is immediate from the proof that this result can be extended to work for any approximation factor γ with $\gamma > 1 + \exp(-o(n/\log n))$, by taking $\ell = o(n/\log n)$ and $M = 2^{n+o(n)}$ to be slightly larger.

Complex Semidefinite Programming and Max-k-Cut[*]

Alantha Newman

CNRS–Université Grenoble Alpes, F-38000, Grenoble, France
alantha.newman@grenoble-inp.fr

Abstract

In a second seminal paper on the application of semidefinite programming to graph partitioning problems, Goemans and Williamson showed how to formulate and round a *complex semidefinite program* to give what is to date still the best-known approximation guarantee of .836008 for Max-3-Cut [5]. (This approximation ratio was also achieved independently by De Klerk et al. [2].) Goemans and Williamson left open the problem of how to apply their techniques to Max-k-Cut for general k. They point out that it does not seem straightforward or even possible to formulate a good quality complex semidefinite program for the general Max-k-Cut problem, which presents a barrier for the further application of their techniques.

We present a simple rounding algorithm for the standard semidefinite programmming relaxation of Max-k-Cut and show that it is equivalent to the rounding of Goemans and Williamson in the case of Max-3-Cut. This allows us to transfer the elegant analysis of Goemans and Williamson for Max-3-Cut to Max-k-Cut. For $k \geq 4$, the resulting approximation ratios are about .01 worse than the best known guarantees. Finally, we present a generalization of our rounding algorithm and conjecture (based on computational observations) that it matches the best-known guarantees of De Klerk et al.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Graph Partitioning, Max-k-Cut, Semidefinite Programming

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.13

1 Introduction

In the Max-k-Cut problem, we are given an undirected graph, $G = (V, E)$, with non-negative edge weights. Our objective is to divide the vertices into at most k disjoint sets, for some given positive integer k, so as to maximize the weight of the edges whose endpoints lie in different sets. When $k = 2$, this problem is known simply as the Max-Cut problem. The approximation guarantee of $1 - 1/k$ can be achieved for all k by placing each vertex uniformly at random in one of k sets. For all values of $k \geq 2$, this simple algorithm yielded the best-known approximation ratio until 1994. In that year, Goemans and Williamson gave a .87856-approximation algorithm for the Max-Cut problem based on semidefinite programming (SDP), thereby introducing this method as a successful new technique for designing approximation algorithms [4].

Frieze and Jerrum subsequently developed an algorithm for the Max-k-Cut problem that can be viewed as a generalization of Goemans and Williamson's algorithm for Max-Cut in the sense that it is same algorithm when $k = 2$ [3]. Although the rounding algorithm of Frieze and Jerrum is arguably simple and natural, the analysis is quite involved. Their

[*] Supported in part by LabEx PERSYVAL-Lab (ANR-11-LABX-0025).

approximation ratios improved upon the previously best-known guarantees of $1 - 1/k$ for $k \geq 3$ and are shown in Table 1. A few years later, Andersson, Engebretsen and Håstad also used semidefinite programming to design an algorithm for the more general problem of Max-E2-Lin mod k, in which the input is a set of 2-variable equations or inequations mod k (e.g. $x - y \equiv c \mod k$) and the objective is to assign an integer from the range $[0, k-1]$ to each variable so that the maximum number of equations are satisfied [1]. They proved that the approximation guarantee of their algorithm is at least $f(k)$ more than that of the simple randomized algorithm, where $f(k)$ is a (small) linear function of k. In the special case of Max-k-Cut, they showed that the performance ratio of their algorithm is no better than that of Frieze and Jerrum. Although they did not show the equivalence of these two algorithms, they stated that numerical evidence suggested that the two algorithms have the same approximation ratio. Shortly thereafter, De Klerk, Pasechnik and Warners presented an algorithm for Max-k-Cut with improved approximation guarantees for all $k \geq 3$, shown in Table 1. Additionally, they showed that their algorithm has the same worst-case performance guarantee as that of Frieze and Jerrum [2].

Around the same time, Goemans and Williamson independently presented another algorithm for Max-3-Cut based on *complex semidefinite programming* (CSDP) [5]. For this problem, they improved the best-known approximation guarantee of .832718 due to Frieze and Jerrum to .836008, the same approximation ratio proven by De Klerk, Pasechnik and Warners. Goemans and Williamson showed that their algorithm is equivalent to that of Andersson, Engebretsen and Håstad and to that of Frieze and Jerrum (and therefore to that of De Klerk, Pasechnik and Warners) in the case of Max-3-Cut [5]. However, they argued that their decision to use complex semidefinite programming and, specifically, their choice to represent each vertex by a single complex vector resulted in "cleaner models, algorithms, and analysis than the equivalent models using standard semidefinite programming."

One issue noted by Goemans and Williamson with respect to their elegant new model was that it is not clear how to apply their techniques to Max-k-Cut for $k \geq 4$. Their approach seemed to be tailored specifically to the Max-3-Cut problem. This is because one cannot model, say, the Max-4-Cut problem directly using a complex semidefinite program. This limitation is discussed in Section 8 of [5]. In fact, as they point out, a direct attempt to model Max-k-Cut with a complex semidefinite program would only result in a $(1 - 1/k)$-approximation for $k \geq 4$. De Klerk et al. also state that there is no obvious way to extend the approach based on CSDP to Max-k-Cut for $k > 3$. (See page 269 in [2].)

1.1 Our Contribution

In this paper, we make the following contributions.

1. We present a simple rounding algorithm based on the standard semidefinite programming relaxation of Max-k-Cut and show that it can be analyzed using the tools from [5].
   - For $k = 3$, this results in an implementation of the Goemans-Williamson algorithm that avoids complex semidefinite programming.
   - For $k \geq 4$, the resulting approximation ratios are slightly worse than the best-known guarantees.
2. We present a simple generalization of this rounding algorithm and conjecture that it yields the best-known approximation ratios.

Thus, the main contribution of this paper is to show that, despite its limited modeling power, we can still apply the tools from complex semidefinite programming developed by Goemans and Williamson to Max-k-Cut. In fact, we obtain the following worst-case

Table 1 Approximation guarantees for MAX-k-CUT.

k	[4]	[3]	[5]	[2]	This paper
$k = 2$	.878956	-	-	-	-
$k = 3$	-	.832718	.836008	.836008	-
$k = 4$	-	.850304	-	.857487	.846478
$k = 5$	-	.874243	-	.876610	.862440
$k = 10$	-	.926642	-	.926788	.915885

approximation guarantee for the MAX-k-CUT problem for all k, which is the same bound they achieve for $k = 3$:

$$\phi_k = \frac{k-1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(\left(\frac{1}{k-1}\right)\cos\left(\frac{2\pi}{k}\right)\right) - \arccos^2\left(\frac{1}{k-1}\right)\right]. \tag{1}$$

We note that for $k \geq 4$, the approximation ratio ϕ_k is about .01 worse than the approximation ratio proved by Frieze and Jerrum. See Table 1 for a comparison. However, given the technical difficulty of Frieze and Jerrum's analysis, we believe that it is beneficial to present an alternative algorithm and analysis that yields a similiar approximation guarantee. Moreover, we wish to take a closer look at the techniques used by Goemans and Williamson for MAX-3-CUT since these tools have not been widely applied in the area of approximation algorithms, in sharp contrast to the tools used to solve the MAX-CUT problem. In fact, we are aware of only two papers that use the main tools of [5]: The first is for a generalization of the MAX-3-CUT problem [6] and the second is for an optimization problem in which the variables are to be assigned complex vectors [8].

While Goemans and Williamsons' framework of complex semidefinite programming does result in an elegant formulation and analysis for MAX-3-CUT, it also to some extent obscures the geometric structure that is apparent when one views the same algorithm from the viewpoint of standard semidefinite programming. Specifically, in the latter framework, their complex semidefinite program is equivalent to modeling each vertex with a 2-dimensional circle or disc of vectors. In our opinion, their main technical contribution is a formula for the exact distribution of the difference of the angles resulting when a normal vector is projected onto two of these discs that are correlated in a particular way. (See Lemma 8 in [5].) Thus, while the limitation in modeling MAX-k-CUT with complex semidefinite programming comes from the fact that we cannot model the general problem with these 2-dimensional discs, we can circumvent this barrier in the following way. We construct 2-dimensional discs using the vectors obtained from a solution to the standard semidefinite program. We then show that a pair of these 2-dimensional discs (i.e. one disc for each vertex) are correlated in the same way as those produced in the case of MAX-3-CUT. Then we can apply and analyze the same algorithm used for MAX-3-CUT.

In some cases, e.g. MAX-3-CUT, using the distribution of the angle between two elements is stronger than using the expected angle, which is what is used for MAX-CUT. It therefore seems that this tool has unexplored potential applications for other optimization problems, for which it may also be possible to overcome the modeling limitations of complex semidefinite programming in a similiar manner as we do here. On a high level, the idea of constructing the "complex" vectors from a solution to a standard semidefinite program was used for a circular arrangement problem [7].

Finally, we remark that the approach used in Section 4 to create a disc from a vector is reminiscent of Zwick's method of outward rotations in which he combines hyperplane rounding and independent random assignment [9]. For each unit vector v_i from an SDP solution, he computes a disc in the plane spanned by v_i and u_i, where the u_i's form a set of pairwise orthogonal vectors that are also orthogonal to the v_i's, and chooses a new vector from this disc based on a predetermined angle. Thus, the goal is to rotate each vector v_i to obtain a new set of unit vectors, which are then given as input to a now standard rounding algorithm, such as random-hyperplane rounding. In contrast, our goal is to use the actual disc in the rounding, as done originally by Goemans and Williamson in the case of Max-3-Cut.

1.2 Organization

We give some background on the (standard) semidefinite programming relaxation used by Frieze and Jerrum and discuss their algorithm for Max-k-Cut in Section 2. In Section 3, we present Goemans and Williamson's algorithm for Max-3-Cut from the viewpoint of standard semidefinite programming. In Section 4, we show how to create a 2-dimensional disc for each vertex given a solution to the standard semidefinite program for Max-k-Cut. We do not wish to formally prove the relationship between these discs and the complex vectors. Thus, in Section 5, we simply prove that if two discs are correlated in a specified way, then the distribution of the angle is equivalent to a distribution already computed exactly by Goemans and Williamson in [5]. We can then easily prove that the 2-dimensional discs we create for the vertices have the required pairwise correlation. This results in a closed form approximation ratio for general k, Theorem 6.

2 Frieze and Jerrum's Algorithm

Consider the following integer program for Max-k-Cut:

$$\begin{aligned}
&\max \sum_{ij\in E}(1 - v_i\cdot v_j)\frac{k-1}{k}\\
&v_i\cdot v_i = 1, \quad \forall i\in V,\\
&\quad v_i \in \Sigma_k, \quad \forall i \in V.
\end{aligned}\tag{P}$$

Here, Σ_k are the vertices of the equilateral simplex, where each vertex is represented by a k-dimensional vector, and each pair of vectors corresponding to a pair of vertices has dot product $-1/(k-1)$. If we relax the dimension of the vectors, we obtain the following semidefinite relaxation, where $n = |V|$:

$$\begin{aligned}
&\max \sum_{ij\in E}(1 - v_i\cdot v_j)\frac{k-1}{k}\\
&v_i\cdot v_i = 1, \quad \forall i\in V,\\
&v_i\cdot v_j \geq -\frac{1}{k-1}, \quad \forall i,j\in V,\\
&\quad v_i \in \mathbb{R}^n, \quad \forall i \in V.
\end{aligned}\tag{Q}$$

Frieze and Jerrum used this semidefinite relaxation to obtain an algorithm for the Max-k-Cut problem [3]. Specifically, they proposed the following rounding algorithm: Choose k random vectors, $g_1, g_2, \ldots, g_k \in \mathbb{R}^n$, with each entry of each vector chosen from the normal distribution $\mathcal{N}(0,1)$. For each vertex $i \in V$, consider the k dot products of vector v_i with each of the k random vectors, $v_i\cdot g_1, v_i\cdot g_2, \ldots, v_i\cdot g_k$. One of these dot products is maximum. Assign the vertex the label of the random vector with which it has the maximum dot product.

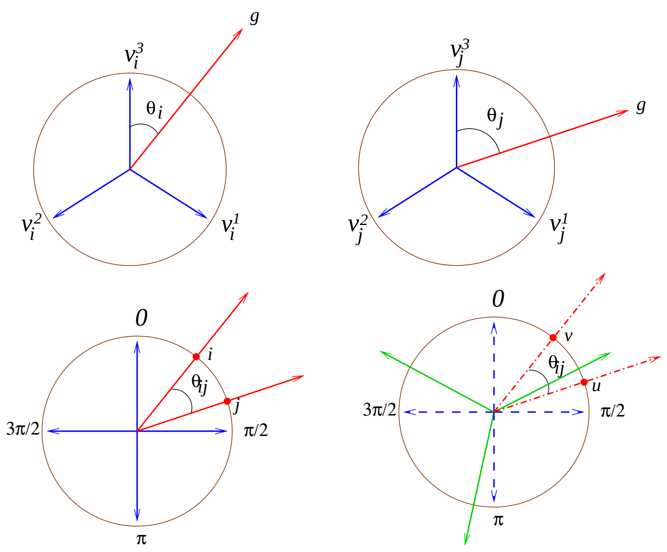

Figure 1 Three vectors v_i^1, v_i^2 and v_i^3 lie on a 2-dimensional plane corresponding to vertex i. The vector g is projected onto the disc for element i to obtain θ_i. Angle θ_{ij} is the difference between angles θ_i and θ_j.

In other words, if $v_i \cdot g_h = \max_{\ell=1}^{k}\{v_i \cdot g_\ell\}$, then vertex i is assigned to to cluster h. Frieze and Jerrum were able to prove a lower bound on the approximation guarantee of this algorithm for every k. See Table 1 for some of these ratios.

3 Goemans-Williamson Algorithm for Max-3-Cut

Goemans and Williamson gave an algorithm for MAX-3-CUT in which they first modeled the problem as a complex semidefinite program, i.e. each element is represented by a complex vector. It is not too difficult to see that these complex vectors are equivalent to 2-dimensional discs or sets of unit vectors. For example, here is an equivalent semidefinite program for MAX-3-CUT. The input is an undirected graph $G = (V, E)$ with non-negative edge weights $\{w_{ij}\}$.

$$\max \sum_{ij \in E} w_{ij}(1 - v_i^1 \cdot v_j^1)\frac{2}{3} \tag{2}$$

$$v_i^a \cdot v_i^b = -1/2, \qquad \forall i \in V,\ a \neq b \in [3], \tag{3}$$

$$v_i^a \cdot v_j^b = v_i^{a+c} \cdot v_j^{b+c}, \qquad \forall i, j \in V,\ a, b, c \in [3], \tag{4}$$

$$v_i^a \cdot v_j^b \geq -1/2, \qquad \forall i, j \in V,\ a, b \in [3], \tag{5}$$

$$v_i^a \cdot v_i^a = 1, \qquad \forall i \in V,\ a \in [3], \tag{6}$$

$$v_i^a \in \mathbb{R}^{3n}, \qquad \forall i \in V,\ a \in [3]. \tag{7}$$

Consider a set of $3n$ unit vectors forming a solution to this semidefinite program. Note that for a fixed vertex $i \in V$, the vectors v_i^1, v_i^2 and v_i^3 are in the same two dimensional plane, since they are constrained to be pairwise 120° apart. In an "integer" solution for this semidefinite program, all these discs would be constrained to be in the same 2-dimensional space and each angle of rotation of the discs would be constrained to be $0, 2\pi/3$ or $4\pi/3$, where each angle would correspond to a partition. In a solution to the above relaxation, these discs are no longer constrained to be in 2 dimensions.

In the rounding algorithm of Goemans and Williamson, we first pick a vector $g \in \mathbb{R}^{3n}$ such that each entry is chosen according to the normal distribution $\mathcal{N}(0,1)$. Then for each vertex $i \in V$, we project this vector g onto its corresponding disc. This gives an angle θ_i in the range $[0, 2\pi)$ for each element i. (Note that without loss of generality, we can assume that θ_i is the angle in the clockwise direction between the projection of g and the vector v_i^3.) We can envision the angles $\{\theta_i\}$ for each $i \in V$ embedded onto the same disc. Then we randomly partition this disc into three equal pieces, each of length $2\pi/3$, i.e. we choose an angle $\psi \in [0, 2\pi)$ and let the three angles of partition be $\psi, \psi + 2\pi/3$ and $\psi + 4\pi/3$. These three pieces correspond to the three sets in the partition.

The angle θ_{ij} is the angle $\theta_j - \theta_j$ modulo 2π. The probability that an edge ij is cut in this partitioning scheme is equal to $3\theta_{ij}/2\pi$ if $\theta_{ij} < 2\pi/3$ and 1 otherwise. In expectation, the angle θ_{ij} is equal to $\arccos(v_i^1 \cdot v_j^1)$. (This can be shown using the techniques in [4]. See Lemma 3 in [7].) But using the expected angle is not sufficient to obtain an approximation guarantee better than 2/3; If angle θ_{ij} is $2\pi/3$ in expectation, then one third of the time it could be zero (not cut) and two thirds of the time it could be π (cut). However, it contributes one to the objective function. The exact probability Pr[edge ij is cut] that edge ij is cut is:

$$\sum_{\gamma=0}^{2\pi/3} \Pr[\theta_{ij} = \theta] \times \frac{\theta}{2\pi/3} + \sum_{\gamma=2\pi/3}^{4\pi/3} \Pr[\theta_{ij} = \theta] + \sum_{\theta=4\pi/3}^{2\pi} \Pr[\theta_{ij} = \theta] \times \frac{2\pi - \theta}{2\pi/3}.$$

Therefore, we must compute $\Pr[\theta_{ij} = \theta]$ for all $\theta \in [0, 2\pi)$. One of the main technical contributions of Goemans and Williamson [5] is that they compute the exact probability that $\theta_{ij} < \delta$ for all $\delta \in [0, 2\pi)$. This can be found in Lemma 8 [5]. This enables them to compute the probability that an edge is cut, resulting in their approximation guarantee.

4 Algorithm for Max-k-Cut

As previously mentioned, we cannot model Max-k-Cut as an integer program directly using 2-dimensional discs as we do for Max-3-Cut, because any rotation corresponding to an angle of at least $2\pi/k$ should contribute one to the objective function. Note that in the case of Max-3-Cut, there are two possible non-zero rotations in an integer solution: $2\pi/3$ and $4\pi/3$ and both of the contribute the same amount (one) to the objective function. Since it seems impossible to penalize all angles greater than $2\pi/k$ at the same cost, it seems similiarly impossible to model the problem directly with a complex semidefinite program.

We now present our approach for rounding the semidefinite programming relaxation (Q) for Max-k-Cut. After solving the semidefinite program, we obtain a set of vectors $\{v_i\}$ corresponding to each vertex $i \in V$. We can assume these vectors to be in dimension n. Let $\mathbf{0}$ represent the vector with n zeros. For each vertex $i \in V$, we construct the following two orthogonal vectors:

$$v_i \ := \ (v_i, \mathbf{0}), \qquad v_i^{\perp} \ := \ (\mathbf{0}, v_i). \tag{8}$$

Each vertex $i \in V$ now corresponds to a 2-dimensional disc spanned by vectors v_i and $v_i^{\perp}$. Specifically, this 2-dimensional disc consists of the (continuous) set of vectors defined for $\phi \in [0, 2\pi)$:

$$v_i(\phi) = v_i \cos\phi + v_i^{\perp} \sin\phi. \tag{9}$$

Now that we have constructed a 2-dimensional disc for each element, we can use the same rounding scheme due to Goemans and Williamson described in the previous section: First,

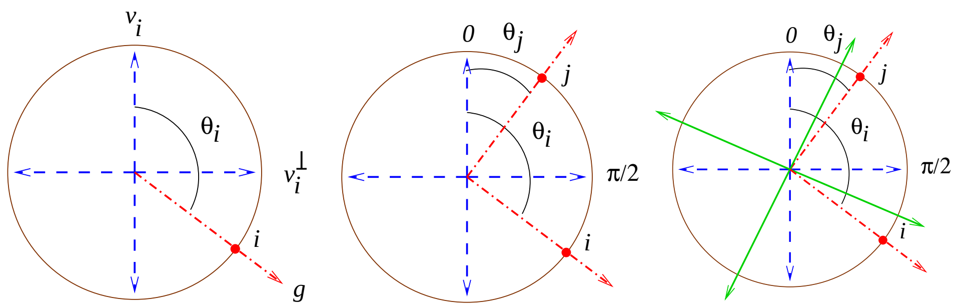

Figure 2 A 2-dimensional plane for vertex i spanning v_i and $v_i^\perp$. After projecting θ_i and θ_j onto the same disc, we partion the disc into $k = 4$ equal sized pieces.

we choose a vector $g \in \mathbb{R}^{2n}$ in which each coordinate is randomly chosen according to the normal distribution $\mathcal{N}(0, 1)$. For each $i \in V$, we project this vector g onto the disc $\{v_i(\phi)\}$, which results in an angle θ_i, where:

$$g \cdot v_i(\theta_i) = \max_{0 \le \phi < 2\pi} g \cdot v_i(\phi).$$

Note that we do not have to compute infinitely many dot products, since, for example, if $g \cdot v_i,\ g \cdot v_i^\perp \ge 0$, then:

$$\theta_i = \arctan\left(\frac{g \cdot v_i^\perp}{g \cdot v_i}\right),$$

and the three other cases depending on the sign of $g \cdot v_i$ and $g \cdot v_i^\perp$ can be handled accordingly.

After we find an angle θ_i for each $i \in V$, we can assign each element to a position corresponding to its angle θ_i on a single disc and divide this disc (randomly) into k equal sections of size $2\pi/k$. Specifically, choose a random angle ψ and use the partition $\psi + \frac{c \cdot 2\pi}{k}$ for all integers $c \in [0, k)$, where angles are taken modulo 2π. These are the k partitions of the vertices in the k-cut.

5 Analysis

We prove that the distribution of the angle θ_{ij} is the same as Lemma 8 of [5]. This implies that we can use the analysis that Goemans and Williamson use for Max-3-Cut to obtain an analogous approximation ratio for Max-k-Cut.

▶ **Lemma 1.** *Given two sets of vectors $x_i = \{x_i(\phi)\}$ and $x_j = \{x_j(\phi)\}$ defined on $\phi \in [0, 2\pi)$, where*

$$x_i(\phi) = (\cos\phi,\ \sin\phi,\ 0,\ 0),$$
$$x_j(\phi) = (\cos\theta\cos\phi,\ \cos\theta\sin\phi,\ \sin\theta\cos\phi,\ \sin\theta\sin\phi).$$

Let $\gamma \in [0, 2\pi)$ denote the angle $\theta_j - \theta_i$ after the vector $g \in \mathcal{N}(0, 1)^{2n}$ is projected onto x_i and x_j. Then for $\delta \in [0, 2\pi)$,

$$\Pr[0 \le \gamma < \delta] = \frac{1}{2\pi}\left(\delta + \frac{r\sin\delta}{\sqrt{1 - r^2\cos^2\delta}}\arccos\left(-r\cos\delta\right)\right). \tag{10}$$

Proof. Note that the set of vectors x_j is 2-dimensional, since the angle between $x_j(\phi_1)$ and $x_j(\phi_2)$ for $\phi_2 > \phi_1$ is $\phi_2 - \phi_1$. Thus, the rounding algorithm in Section 4 is well defined.

Recall that each coordinate of the vector g is chosen according to the normal distribution $\mathcal{N}(0,1)$. Even though the vector g has $2n$ dimensions, we only need to consider the first four, $g = (g_1, g_2, g_3, g_4)$. This vector is chosen equivalently to choosing α, β uniformly in $[0, 2\pi)$ and p_1, p_2 according to the distribution:

$$f(y) = ye^{-y^2/2}.$$

In other words, the vector g is equivalent to:

$$g = (p_1 \cos\beta,\ p_1 \sin\beta,\ p_2 \cos\alpha,\ p_2 \sin\alpha).$$

Let $r = \cos\theta$ and let $s = \sin\theta$. We will show that the probability that $\gamma \in [0, \delta)$ for $\delta \le \pi$ is:

$$\Pr[0 \le \gamma < \delta] = \frac{1}{2\pi}\left[\delta + \int_{\delta}^{\pi} \Pr\left[\frac{p_2 \cdot s}{\sin\delta} \le \frac{p_1 \cdot r}{\sin(\alpha - \delta)}\right] d\alpha\right]. \tag{11}$$

Lemma 8 in [5] shows this is equivalent to probability in (10).

First, let us consider the case when $\theta \in [0, \pi/2]$, or $\cos\theta \ge 0$. Without loss of generality, assume that the projection of g onto the 2-dimensional disc x_i occurs at $\phi = 0$. Then we can see that

$$x_i(0) \cdot g = p_1.$$

In other words, we can assume that $\theta_i = 0$. As previously mentioned, α is chosen uniformly in the range $[0, 2\pi)$. However, if $\gamma < \delta$, then $\alpha < \pi$. If $\alpha < \delta$, then the projection of g onto x_j, namely θ_j (which equals θ_{ij} in this case, because we have assumed that $\theta_i = 0$), is less than δ. The probability that $\gamma \le \delta$ if $\alpha \in [\delta, \pi)$ is equal to the probability that:

$$\frac{p_2 \cdot s}{\sin\delta} \le \frac{p_1 \cdot r}{\sin(\alpha - \delta)} \iff$$
$$p_2 \cdot s \le \frac{p_1 \cdot r}{\sin(\alpha - \delta)} \cdot \sin\delta.$$

(See Figure 3 in [5].) If $\theta \in (\pi/2, pi)$ and $r = \cos\theta < 0$, then the probability that γ is in $[0, \delta)$ is the probability that γ is in $[\pi, \pi + \delta)$, which is $\delta/(2\pi)$. And the probability that γ is in $[\delta, \pi)$ is the probability that γ is in $[\pi + \delta, 2\pi)$ for $-r$. This is:

$$p_2 \cdot s \le \frac{p_1 \cdot (-r)}{\sin(\alpha - \delta)} \cdot \sin(\pi + \delta). \tag{12}$$

However, since $\sin(\pi + \delta) = -\sin\delta$, we have:

$$p_2 \cdot s \le \frac{p_1 \cdot r}{\sin(\alpha - \delta)} \cdot \sin\delta. \tag{13}$$

Thus for all $\delta < \pi$, we have proved the expression in (11). In Lemma 8 of [5], they show that Equation (11) is equivalent to Equation (10) when $\delta < \pi$. Then they argue by symmetry that Equation (10) also holds when $\pi \le \delta < 2\pi$. ◀

▶ **Lemma 2.** *Suppose $v_i \cdot v_j = \cos\theta$ for two unit vectors v_i and v_j. Let $v_i(\phi)$ and $v_j(\phi)$ be defined as in equation (9). Then, we can assume that:*

$$v_i(\phi) = (\cos\phi,\ \sin\phi,\ 0,\ 0),$$
$$v_j(\phi) = (\cos\theta\cos\phi,\ \cos\theta\sin\phi,\ \sin\theta\cos\phi,\ \sin\theta\sin\phi).$$

Proof. From the definition (in Equation (9)) of $v_i(\phi)$, we can see that:

$$\begin{aligned} v_i(\phi_1) \cdot v_j(\phi_2) &= (v_i \cos\phi_1 + v_i^{\perp} \sin\phi_1) \cdot (v_j \cos\phi_2 + v_j^{\perp} \sin\phi_2) \\ &= v_i \cdot v_j \cos\phi_1 \cos\phi_2 + v_i^{\perp} \cdot v_j^{\perp} \sin\phi_1 \sin\phi_2 \\ &\quad + \ v_i \cdot v_j^{\perp} \cos\phi_1 \sin\phi_2 + v_i^{\perp} \cdot v_j \sin\phi_1 \cos\phi_2 \\ &= \cos\theta(\cos\phi_1 \cos\phi_2 + \sin\phi_1 \sin\phi_2). \end{aligned}$$

Note that $v_i \cdot v_j^{\perp} = v_i^{\perp} \cdot v_j = 0$ since each v_i vector has n zeros in the second half of the entries and each $v_i^{\perp}$ vector has n zeros in the first half of the entries. If we compute $v_i(\phi_1) \cdot v_j(\phi_2)$ using the assumption in the lemma, then we get the same dot product. Thus, the two sets are equivalent. ◀

Since the distribution of the angle is the same, we can use the same analysis of [5] (generalized from 3 to k) to prove the following Lemma. Although it is essentially the exact same proof, we include it here for completeness. As in Corollary 9 of [5], we define:

$$g(r,\delta) = \frac{1}{2\pi}\left(\delta + \frac{r\sin\delta}{\sqrt{1-r^2\cos^2\delta}}\arccos\left(-r\cos\delta\right)\right).$$

In other words, $g(r,\delta)$ is the probability that angle θ_{ij} obtained by projecting g onto the two discs $\{v_i(\phi)\}$ and $\{v_j(\phi)\}$, correlated by $r = v_i \cdot v_j$, is less than δ.

▶ **Lemma 3.** *Let $r = v_i \cdot v_j$ and let $y_i \in [0,1,2,\ldots k)$ be the integer assignment of vertex i to its partition. Then the probability that the equation $y_i - y_j \equiv c \pmod{k}$ is satisfied is*

$$\frac{1}{k} + \frac{k}{8\pi^2}\left[2\arccos^2\left(-r\cos\left(\frac{2\pi c}{k}\right)\right) - \arccos^2\left(-r\cos\left(\frac{2\pi(c+1)}{k}\right)\right) - \arccos^2\left(-r\cos\left(\frac{2\pi(c-1)}{k}\right)\right)\right].$$

Proof. $\Pr[y_i - y_j \equiv c \pmod{k}$ satisfied$]$

$$\begin{aligned} &= \frac{k}{2\pi}\int_0^{\frac{k}{2\pi}} \Pr_{\gamma}\left[\frac{2\pi c}{k} - \tau \le \gamma < \frac{2\pi(c+1)}{k} - \tau\right] d\tau \\ &= \frac{k}{2\pi}\int_0^{\frac{2\pi}{k}} \left(g\left(\tau, \frac{2\pi(c+1)}{k} - \tau\right) - g(\tau, \frac{2\pi c}{k} - \tau)\right) d\tau \\ &= \frac{k}{2\pi}\int_{\frac{2\pi c}{k}}^{\frac{2\pi(c+1)}{k}} g(r,\nu)d\nu - \frac{k}{2\pi}\int_{\frac{2\pi(c-1)}{k}}^{\frac{2\pi c}{k}} g(r,\nu)d\nu \end{aligned}$$

$$= \frac{k}{2\pi}\frac{1}{2\pi}\left(\int_{\frac{2\pi c}{k}}^{\frac{2\pi(c+1)}{k}} \nu d\nu - \left[\frac{1}{2}\arccos^2\left(-r\cos\nu\right)\right]_{\frac{2\pi c}{k}}^{\frac{2\pi(c+1)}{k}}\right.$$
$$\left. - \int_{\frac{2\pi(c-1)}{k}}^{\frac{2\pi c}{k}} \nu d\nu + \left[\frac{1}{2}\arccos^2\left(-r\cos\nu\right)\right]_{\frac{2\pi(c-1)}{k}}^{\frac{2\pi c}{k}}\right)$$
$$= \frac{k}{8\pi^2}\left[\left(\frac{2\pi(c+1)}{k}\right)^2 + \left(\frac{2\pi(c-1)}{k}\right)^2 - 2\left(\frac{2\pi c}{k}\right)^2\right]$$
$$+ \frac{k}{8\pi^2}\left[2\arccos^2\left(-r\cos\left(\frac{2\pi c}{k}\right)\right)\right.$$
$$\left. - \arccos^2\left(-r\cos\left(\frac{2\pi(c+1)}{k}\right)\right) - \arccos^2\left(-r\cos\left(\frac{2\pi(c-1)}{k}\right)\right)\right]$$
$$= \frac{1}{k} + \frac{k}{8\pi^2}\left[2\arccos^2\left(-r\cos\left(\frac{2\pi c}{k}\right)\right)\right.$$
$$\left. - \arccos^2\left(-r\cos\left(\frac{2\pi(c+1)}{k}\right)\right) - \arccos^2\left(-r\cos\left(\frac{2\pi(c-1)}{k}\right)\right)\right]. \qquad ◀$$

▶ **Lemma 4.** *Let $r = v_i \cdot v_j$. The probability that edge ij is* not *cut by our algorithm is:*

$$\frac{1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(-r\right) - \arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right)\right].$$

Proof. In the case of Max-k-Cut, we set $c = 0$. By Lemma 3, we have the probability that edge ij is not cut is:

$$\frac{1}{k} + \frac{k}{8\pi^2}\left[2\arccos^2\left(-r\right) - \arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right) - \arccos^2\left(-r\cos\left(-\frac{2\pi}{k}\right)\right)\right]$$
$$= \frac{1}{k} + \frac{k}{8\pi^2}\left[2\arccos^2\left(-r\right) - 2\arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right)\right]$$
$$= \frac{1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(-r\right) - \arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right)\right]. \qquad ◀$$

▶ **Lemma 5.** *Let $r = v_i \cdot v_j$. The probability that edge ij is cut by our algorithm is:*

$$\frac{k-1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(-r\cdot\cos\left(\frac{2\pi}{k}\right)\right) - \arccos^2\left(-r\right)\right]. \qquad (14)$$

Proof. By Lemma 4 and the previously stated assumption that $r = v_i \cdot v_j = \cos\left(\theta_{ij}\right)$, we have:

$$1 - \left[\frac{1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(-r\right) - \arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right)\right]\right]$$
$$= \frac{k-1}{k} - \frac{k}{4\pi^2}\left[\arccos^2\left(-r\right) - \arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right)\right]$$
$$= \frac{k-1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(-r\cos\left(\frac{2\pi}{k}\right)\right) - \arccos^2\left(-r\right)\right]. \qquad ◀$$

▶ **Theorem 6.** *The worst case approximation ratio of our algorithm for* Max-k-Cut *is:*

$$\phi_k = \frac{k-1}{k} + \frac{k}{4\pi^2}\left[\arccos^2\left(\left(\frac{1}{k-1}\right)\cos\left(\frac{2\pi}{k}\right)\right) - \arccos^2\left(\frac{1}{k-1}\right)\right].$$

Proof. As a function of r in the range $[1, -1/(k-1)]$, the expression in Equation 14 is minimized when $r = -1/(k-1)$. Thus, if we do an edge-by-edge analysis, the worst case approximation ratio is obtained when $v_i \cdot v_j = -1/(k-1)$ for all edges $ij \in E$. ◀

6 Another Rounding Algorithm

The algorithm presented in Section 4 can be restated as the following rounding scheme. Let w_1, w_2 and w_3 denote vectors in $\mathbb{R}^2$ with pairwise dot product $-1/2$. In other words, w_1, w_2 and w_3 are the vertices of the simplex Σ_3. Now take two random gaussians $g_1, g_2 \in \mathbb{R}^n$ and set $x_i = g_1 \cdot v_i$, $y_i = g_2 \cdot v_i$. To assign the vertex i to one of the three partitions, we simply assign it to j such that $w_j \cdot (x_i, y_i)$ is maximized.

We can generalize this approach by choosing $k-1$ random gaussians, $g_1, \dots, g_{k-1}$. For each vertex i, we obtain the vector $(g_1 \cdot v_i, g_2 \cdot v_i, \dots, g_{k-1} \cdot v_i)$ in $\mathbb{R}^{k-1}$. This vector is assigned to the closest vertex of Σ_k. Computationally, this rounding scheme seems to yield approximation ratios that match those of De Klerk et al.

Acknowledgements. Thanks to Moses Charikar, Anupam Gupta, R. Ravi and Madhur Tulsiani for helpful discussions and comments on the presentation.

References

1 Gunnar Andersson, Lars Engebretsen, and Johan Håstad. A new way of using semidefinite programming with applications to linear equations mod p. *J. Algorithms*, 39(2):162–204, 2001. `doi:10.1006/jagm.2000.1154`.

2 Etienne de Klerk, Dmitrii V. Pasechnik, and Joost P. Warners. On approximate graph colouring and max-k-cut algorithms based on the theta-function. *J. Comb. Optim.*, 8(3):267–294, 2004. `doi:10.1023/B:JOCO.0000038911.67280.3f`.

3 Alan M. Frieze and Mark Jerrum. Improved approximation algorithms for MAX k-cut and MAX BISECTION. *Algorithmica*, 18(1):67–81, 1997. `doi:10.1007/BF02523688`.

4 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. `doi:10.1145/227683.227684`.

5 Michel X. Goemans and David P. Williamson. Approximation algorithms for m_{ax}-3-c_{ut} and other problems via complex semidefinite programming. *J. Comput. Syst. Sci.*, 68(2):442–470, 2004. `doi:10.1016/j.jcss.2003.07.012`.

6 Ai-fan Ling. Approximation algorithms for max 3-section using complex semidefinite programming relaxation. In Ding-Zhu Du, Xiaodong Hu, and Panos M. Pardalos, editors, *Combinatorial Optimization and Applications, Third International Conference, COCOA 2009, Huangshan, China, June 10-12, 2009. Proceedings*, volume 5573 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2009. `doi:10.1007/978-3-642-02026-1_20`.

7 Konstantin Makarychev and Alantha Newman. Complex semidefinite programming revisited and the assembly of circular genomes. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 444–459. Tsinghua University Press, 2011. URL: `http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/27.html`.

8 Shuzhong Zhang and Yongwei Huang. Complex quadratic optimization and semidefinite programming. *SIAM Journal on Optimization*, 16(3):871–890, 2006. `doi:10.1137/04061341X`.

9 Uri Zwick. Outward rotations: A tool for rounding solutions of semidefinite programming relaxations, with applications to MAX CUT and other problems. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 679–687. ACM, 1999. `doi:10.1145/301250.301431`.

A Simple, Space-Efficient, Streaming Algorithm for Matchings in Low Arboricity Graphs

Andrew McGregor[1] and Sofya Vorotnikova[2]

1 **College of Computer and Information Sciences, University of Massachusetts, Amherst, MA, USA**
`mcgregor@cs.umass.edu`

2 **College of Computer and Information Sciences, University of Massachusetts, Amherst, MA, USA**
`svorotni@cs.umass.edu`

Abstract

We present a simple single-pass data stream algorithm using $O(\epsilon^{-2} \log n)$ space that returns a $(\alpha + 2)(1 + \epsilon)$ approximation to the size of the maximum matching in a graph of arboricity α.

1998 ACM Subject Classification F.2 Analysis of Algorithms & Problem Complexity

Keywords and phrases data streams, matching, planar graphs, arboricity

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.14

1 Introduction

We present a data stream algorithm for estimating the size of the maximum matching of a low arboricity graph. Recall that a graph has arboricity α if its edges can be partitioned into at most α forests and that a planar graph has arboricity $\alpha = 3$. Estimating the size of the maximum matching in such graphs has been a focus of recent data stream research [1–4, 6, 8]. See also [7] for a survey of the general area of graph algorithms in the stream model.

A surprising result on this problem was recently proved by Cormode et al. [4]. They designed an ingenious algorithm that returned a $(22.5\alpha + 6)(1 + \epsilon)$ approximation using a single pass over the edges of the graph (ordered arbitrarily) and $O(\epsilon^{-3} \cdot \alpha \cdot \log^2 n)$ space[1]. We improve the approximation factor to $(\alpha + 2)(1 + \epsilon)$ via a simpler and tighter analysis and show that, with a modification and simplification of their algorithm, the space required can be reduced to $O(\epsilon^{-2} \log n)$.

2 Results

Let $\mathsf{match}(G)$ be the maximum size of a matching in a graph G and let E_α be the set of edges uv where the number of edges incident to u or v that appear in the stream after uv are both at most α.

2.1 A Better Approximation Factor

We first show a bound for $\mathsf{match}(G)$ in terms of $|E_\alpha|$. Cormode et al. proved a similar but looser bound via results on the size of matchings in bounded degree graphs.

[1] Here, and throughout, space is specified in words and we assume that an edge or a counter (between 0 and α) can be stored in one word of space.

▶ **Theorem 1.** $\mathsf{match}(G) \le |E_\alpha| \le (\alpha+2)\,\mathsf{match}(G)$.

Proof. We first prove the right inequality. To do this define $y_e = 1/(\alpha+1)$ if e is in E_α and 0 otherwise. Note that $\{y_e\}_{e\in E}$ is a fractional matching with maximum weight $1/(\alpha+1)$. A corollary of Edmonds' Matching Polytope Theorem [5] implies that its total weight is at most $(\alpha+2)/(\alpha+1)$ larger than the maximum integral matching. This corollary is likely well known but, for completeness, we include a proof of the corollary in the appendix. Hence,

$$\frac{|E_\alpha|}{\alpha+1} = \sum_e y_e \le \frac{\alpha+2}{\alpha+1}\cdot \mathsf{match}(G)\ .$$

It remains to prove the left inequality. Define H to be the set of vertices with degree $\alpha+1$ or greater. We refer to these as the *heavy* vertices. For $u \in V$, let B_u be the set of the last $\alpha+1$ edges incident to u that arrive in the stream.

Say an edge uv is *good* if $uv \in B_u \cap B_v$ and *wasted* if $uv \in B_u \oplus B_v$, i.e., the symmetric difference. Then $|E_\alpha|$ is exactly the number of good edges. Define

$$\begin{aligned} w &= \text{number of good edges with no end points in } H\ ,\\ x &= \text{number of good edges with exactly one end point in } H\ ,\\ y &= \text{number of good edges with two end points in } H\ ,\\ z &= \text{number of wasted edges with two end points in } H\ , \end{aligned}$$

and note that $|E_\alpha| = w + x + y$.

We know $x + 2y + z = (\alpha+1)|H|$ because B_u contains exactly $\alpha+1$ edges if $u \in H$. Furthermore, $z + y \le \alpha|H|$ because the graph has arboricity α. Therefore

$$x + y \ge (\alpha+1)|H| - \alpha|H| = |H|\ .$$

Let E_L be the set of edges with no endpoints in H. Since every edge in E_L is good, $w = |E_L|$. Hence, $|E_\alpha| \ge |H| + |E_L| \ge \mathsf{match}(G)$ where the last inequality follows because at most one edge incident to each heavy vertex can appear in a matching. ◀

Let G_t be the graph defined by the stream prefix of length t and let E_α^t be the set of good edges with respect to this prefix, i.e., all edges uv from G_t where the number of edges incident to u or v that appear after uv in the prefix are both at most α. By applying the theorem to G_t, and noting that $\max_t |E_\alpha^t| \ge |E_\alpha|$ and $\mathsf{match}(G_t) \le \mathsf{match}(G)$, we deduce the following corollary:

▶ **Corollary 2.** *Let* $E^* = \max_t |E_\alpha^t|$. *Then* $\mathsf{match}(G) \le E^* \le (\alpha+2)\,\mathsf{match}(G)$.

2.2 A Simpler Algorithm using Smaller Space

See Figure 1 for an algorithm that approximates E^* to a $(1+\epsilon)$-factor in the insert-only graph stream model. The algorithm is a modification of the algorithm for estimating $|E_\alpha|$ designed by Cormode et al. [4]. The basic idea is to independently sample edges from E_α^t with probability that is high enough to obtain an accurate approximation of $|E_\alpha^t|$ and yet low enough to use a small amount of space. For every sampled edge $e = uv$, the algorithm stores the edge itself and two counters c_e^u and c_e^v for degrees of its endpoints in the rest of the stream. If we detect that a sampled edge is not in E_α^t, i.e., either of the associated counters exceed α, it is deleted.

Cormode et al. ran multiple instances of this basic algorithm corresponding to sampling probabilities $1, (1+\epsilon)^{-1}, (1+\epsilon)^{-2}, \ldots$ in parallel; terminated any instance that used too

Algorithm 1 APPROXIMATING E^* Algorithm.

```
1. Initialize $S \leftarrow \emptyset$, $p = 1$, estimate $= 0$
2. For each edge $e = uv$ in the stream:
   a. With probability $p$ add $e$ to $S$ and initialize counters $c_e^u \leftarrow 0$ and $c_e^v \leftarrow 0$
   b. For each edge $e' \in S$, if $e'$ shares endpoint $w$ with $e$:
      - Increment $c_{e'}^w$
      - If $c_{e'}^w > \alpha$, remove $e'$ and corresponding counters from $S$
   c. If $|S| > 40\epsilon^{-2}\log n$:
      - $p \leftarrow p/2$
      - Remove each edge in $S$ and corresponding counters with probability $1/2$
   d. estimate $\leftarrow \max(\text{estimate}, |S|/p)$
3. Return estimate
```

much space; and returned an estimate based on one of the remaining instantiations. Instead, we start sampling with probability 1 and put a cap on the number of edges stored by the algorithm. Whenever the capacity is reached, the algorithm halves the sampling probability and deletes every edge currently stored with probability 1/2. This modification saves a factor of $O(\epsilon^{-1}\log n)$ in the space use and update time of the algorithm. We save a further $O(\alpha)$ factor in the analysis by using the algorithm to estimate E^* rather than $|E_\alpha|$.

▶ **Theorem 3.** *With high probability, Algorithm 1 outputs a $(1+\epsilon)$ approximation of E^*.*

Proof. Let k be such that $2^{k-1}\tau \le E^* < 2^k\tau$ where $\tau = 20\epsilon^{-2}\log n$. First suppose we toss $O(\log n)$ coins for each edge in E_α^t and say that an edge e is sampled at level i if at least the first $i-1$ coin tosses at heads. Hence, the probability that an edge is sampled at level i is $p_i = 1/2^i$ and that the probability an edge is sampled at level i conditioned on being sampled at level $i-1$ is $1/2$. Let s_i^t be the number of edges sampled. It follows from the Chernoff bound that for $i \le k$,

$$\mathbb{P}\left[|s_i^t - p_i|E_\alpha^t|| \ge \epsilon p_i E^*\right] \le \exp\left(-\frac{\epsilon^2 E^* p_i}{4}\right) \le \exp\left(-\frac{\epsilon^2 E^* p_k}{4}\right) \le$$
$$\le \exp\left(-\frac{\epsilon^2\tau}{8}\right) = \frac{1}{\text{poly}(n)} .$$

By the union bound, with high probability, $s_i^t/p_i = |E_\alpha^t| \pm \epsilon E^*$ for all $0 \le i \le k$, $1 \le t \le \alpha n$.

The algorithm initially maintains the edges in E_α^t sampled at level $i = 0$. If the number of these edges exceeds the threshold, we subsample these to construct the set of edges sampled at level $i = 1$. If this set of edges also exceeds the threshold, we again subsample these to construct the set of edges at level $i = 2$ and so on. If i never exceeds k, then the above calculation implies that the output is $(1 \pm \epsilon)E^*$. But if s_k^t is bounded above by $(1+\epsilon)E^*/2^k < (1+\epsilon)\tau$ for all t with high probability, then i never exceeds k. ◀

It is immediate that the algorithm uses $O(\epsilon^{-2}\log n)$ space since this is the maximum number of edges stored at any one time. By Corollary 2, E^* is an $(\alpha+2)$ approximation of $\mathsf{match}(G)$ and hence we have proved the following theorem.

▶ **Theorem 4.** *The size of the maximum matching of a graph with arboricity α can be $(\alpha+2)(1+\epsilon)$-approximated with high probability using a single pass over the edges of G given $O(\epsilon^{-2}\log n)$ space.*

Acknowledgement. In an earlier version of the proof of Theorem 3, we erroneously claimed that, conditioned on the current sampling rate being $1/2^j$, edges in E^t_α had been sampled at that rate. Thanks to Sepehr Assadi, Vladimir Braverman, Michael Dinitz, Lin Yang, and Zeyu Zhang for catching this mistake.

References

1 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On Estimating Maximum Matching Size in Graph Streams. In *Proceedings of the Twenty-Eigth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, January 16-19, 2017*, pages 1723-1742, 2017.

2 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, September 14-16, 2015, Proceedings*, pages 263–274, 2015.

3 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, January 10-12, 2016*, pages 1326–1344, 2016.

4 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In *Algorithms - ESA 2017 - 25th Annual European Symposium, September 4-6, 2017, Proceedings*, 2017.

5 Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards, 69:125-130*, 1965.

6 Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, January 4-6, 2015*, pages 1217–1233, 2015.

7 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

8 Andrew McGregor and Sofya Vorotnikova. *Planar Matching in Streams Revisited.* APPROX, 2016.

A Corollary of Edmonds' Theorem

For completeness, we include a simple corollary of Edmonds' Theorem used to prove Theorem 1. Recall that Edmonds' Theorem implies that if the weight of a fractional matching on any induced subgraph $G(U)$ is at most $(|U|-1)/2$, then the weight on the entire graph is at most $\mathsf{match}(G)$.

▶ **Lemma 5.** *Let $\{y_e\}_{e\in E}$ be a fractional matching where the maximum weight is ϵ. Then,*

$$\sum_e y_e \leq (1+\epsilon)\,\mathsf{match}(G)\ .$$

Proof. Let U be an arbitrary subset of vertices and let $E(U)$ be the edges in the induced subgraph on U. Let $t=|U|$. Then since $|E(U)|\leq t(t-1)/2$,

$$\sum_{e\in E(U)} y_e \leq \min\left(\frac{t}{2},\epsilon|E(U)|\right) \leq \frac{t-1}{2}\cdot\min\left(\frac{t}{t-1},\epsilon t\right) \leq \frac{t-1}{2}\cdot(1+\epsilon)\ .$$

Hence, the fractional matching defined by $z_e = y_e/(1+\epsilon)$ satisfies $\sum_e z_e \leq \mathsf{match}(G)$. Therefore, $\sum_e y_e \leq (1+\epsilon)\sum_e z_e \leq (1+\epsilon)\,\mathsf{match}(G)$. ◀

Simple Analyses of the Sparse Johnson-Lindenstrauss Transform[*]

Michael B. Cohen[†1], T. S. Jayram[2], and Jelani Nelson[‡3]

1 **MIT, 32 Vassar Street, Cambridge, MA 02139, USA**
micohen@mit.edu

2 **IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA**
jayram@us.ibm.com

3 **Harvard John A. Paulson School of Engineering and Applied Sciences, 29 Oxford Street, Cambridge, MA 02138, USA**
minilek@seas.harvard.edu

Abstract

For every n-point subset X of Euclidean space and target distortion $1+\varepsilon$ for $0 < \varepsilon < 1$, the *Sparse Johnson Lindenstrauss Transform* (SJLT) of [19] provides a linear dimensionality-reducing map $f : X \to \ell_2^m$ where $f(x) = \Pi x$ for Π a matrix with m rows where (1) $m = O(\varepsilon^{-2} \log n)$, and (2) each column of Π is sparse, having only $O(\varepsilon m)$ non-zero entries. Though the constructions given for such Π in [19] are simple, the analyses are not, employing intricate combinatorial arguments. We here give two simple alternative proofs of the main result of [19], involving no delicate combinatorics. One of these proofs has already been tested pedagogically, requiring slightly under forty minutes by the third author at a casual pace to cover all details in a blackboard course lecture.

1998 ACM Subject Classification F.2.0 General

Keywords and phrases dimensionality reduction, Johnson-Lindenstrauss, Sparse Johnson-Lindenstrauss Transform

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.15

1 Introduction

A widely applied technique to gain speedup and reduce memory footprint when processing high-dimensional data is to first apply a dimensionality-reducing map which approximately preserves the geometry of the input in a pre-processing step. One cornerstone result along these lines is the following Johnson-Lindenstrauss (JL) lemma [16].

▶ **Lemma 1** (JL lemma). *For all $0 < \varepsilon < 1$, integers $n, d > 1$, and $X \subset \mathbb{R}^d$ with $|X| = n$, there exists $f : X \to \mathbb{R}^m$ with $m = O(\varepsilon^{-2} \log n)$ such that*

$$\forall y, z \in X,\ (1-\varepsilon)\|y-z\|_2 \le \|f(y)-f(z)\|_2 \le (1+\varepsilon)\|y-z\|_2. \tag{1}$$

[*] The last two authors dedicate this work to the first author, who in this work and other interactions was a constant source of energy and intellectual enthusiasm.

[†] M. B. Cohen is supported by NSF grants CCF-1111109 and CCF-1553428, and by a National Defense Science and Engineering Graduate Fellowship.

[‡] J. Nelson is supported by NSF grant IIS-1447471 and CAREER award CCF-1350670, ONR Young Investigator award N00014-15-1-2388 and DORECG award N00014-17-1-2127, an Alfred P. Sloan Research Fellowship, and a Google Faculty Research Award.

The target dimension m given by the JL lemma is optimal for nearly the full range of n, d, ε; in particular, for any n, d, ε, there exists a point set $X \subset \mathbb{R}^d$ with $|X| = n$ such that any $(1+\varepsilon)$-distortion embedding of X into $\mathbb{R}^m$ under the Euclidean norm must have $m = \Omega(\min\{n, d, \varepsilon^{-2}\log(\varepsilon^2 n)\})$ [21, 5]. Note that an isometric embedding (i.e. $\varepsilon = 0$) is always achievable into dimension $m = \min\{n-1, d\}$, and thus the lower bound is optimal except potentially for ε close to $1/\sqrt{n}$.

All known proofs of the JL lemma instantiate f as a linear map. The original proof in [16] picked $f(x) = \Pi x$ where $\Pi \in \mathbb{R}^{m \times d}$ was an appropriately scaled orthogonal projection onto a uniformly random m-dimensional subspace. It was then shown that as long as $m = \Omega(\varepsilon^{-2}\log(1/\delta))$,

$$\forall x \in \mathbb{R}^d \text{ such that } \|x\|_2 = 1, \quad \mathop{\mathbb{P}}_{\Pi}(|\|\Pi x\|_2^2 - 1| > \varepsilon) < \delta. \tag{2}$$

The JL lemma then followed by setting $\delta < 1/\binom{n}{2}$ and considering $x = (y-z)/\|y-z\|_2$ for each pair $y, z \in X$, and adjusting ε by a constant factor. It is known that this bound of m for attaining (2) is tight; that is, m must be $\Omega(\min\{d, \varepsilon^{-2}\log(1/\delta)\})$ [15, 17].

One should typically think of applying dimensionality reduction techniques for applications as being a two-step process: first (1) one applies the dimension-reducing map f to the data, then (2) one runs some algorithm on the lower dimensional data $f(X)$. While reducing m typically speeds up the second phase, in order to speed up the first phase it is necessary to give an f which can be both found and applied to data quickly. To this end, Achlioptas showed Π can be chosen with i.i.d. entries where $\Pi_{i,j} = 0$ with probability $2/3$, and otherwise $\Pi_{i,j}$ is uniform in $\pm 1/\sqrt{m/3}$ [1]. This was accomplished without increasing m by even a constant factor over previous best analyses of the JL lemma. Thus essentially a 3x speedup in step (2) is obtained without any loss in the quality of dimensionality reduction. Later, Ailon and Chazelle developed the FJLT [2] which uses the Fast Fourier Transform to implement a JL map Π with $m = O(\varepsilon^{-2}\log n)$ supporting matrix-vector multiplication in time $O(d\log d + m^3)$. Later work of [3] gave a different construction which, for the same m, improved the multiplication time to $O(d\log d + m^{2+\gamma})$ for arbitrarily small $\gamma > 0$. More recently, a sequence of works give embedding time $O(d\log d)$ but with a suboptimal embedding dimension $m = O(\varepsilon^{-2}\log n \cdot \mathrm{poly}(\log\log n))$ [4, 20, 22, 6, 12].

Note that the line of work beginning with the FJLT requires $\Omega(d\log d)$ embedding time per point, which is worse than the $O(m \cdot \|x\|_0)$ time to embed x using a dense Π if x is sufficiently sparse. Here $\|x\|_0$ denotes the number of non-zero entries in x. Motivated by speeding up dimensionality reduction further for sparse inputs, Kane and Nelson in [19], following [10, 18, 7], introduced the SJLT with $m = O(\varepsilon^{-2}\log n)$, and with $s = O(\varepsilon m)$ non-zero entries per column. This reduced the embedding time to compute Πx from $O(m \cdot \|x\|_0)$ to $O(s \cdot \|x\|_0) = O(\varepsilon m \cdot \|x\|_0)$. The original analysis of the SJLT in [19] showed Equation (2) for $m = O(\varepsilon^{-2}\log(1/\delta))$, $s = O(\varepsilon^{-1}\log(1/\delta))$ via the moment method. Specifically, the analysis there for $\|x\|_2 = 1$ defined

$$Z = \|\Pi x\|_2^2 - 1 \tag{3}$$

then used Markov's inequality to yield $\mathbb{P}(|Z| > \varepsilon) < \varepsilon^{-q} \cdot \mathbb{E}\, Z^q$ for some large even integer q (specifically $q = \Theta(\log(1/\delta))$). The bulk of the work was in bounding $\mathbb{E}\, Z^q$, which was accomplished by expanding Z^q as a polynomial with exponentially many terms, grouping terms with similar combinatorial structure, then employing intricate combinatorics to achieve a sufficiently good bound.

Our Main Contribution. We give two new analyses of the SJLT of [19], both of which avoid expanding Z^q into many terms and employing intricate combinatorics. As mentioned in the abstract, one of these proofs has already been tested pedagogically, requiring slightly under forty minutes by the third author at a casual pace to cover all details in a blackboard lecture.

2 Preliminaries

We say $f(x) \lesssim g(x)$ if $f(x) = O(g(x))$, and $f(x) \simeq g(x)$ denotes $f(x) = \Theta(g(x))$. For random variable X and $q \in \mathbb{R}$, $\|X\|_q$ denotes $(\mathbb{E}\,|X|^q)^{1/q}$. Minkowski's inequality, which we repeatedly use, states that $\|\cdot\|_q$ is a norm for $q \geq 1$. If X depends on many random sources, e.g. $X = X(a,b)$, we use $\|X\|_{L_q(a)}$, say, to denote the q-norm over the randomness in a (and thus the result will be a random variable depending only on b). A *Bernoulli-Rademacher* random variable $X = \eta\sigma$ with parameter p is such that η is a Bernoulli random variable (on $\{0,1\}$) with $\mathbb{E}\,\eta = p$ and σ is a Rademacher random variable, i.e. uniform in $\{-1,1\}$. Overloading notation, a random vector X whose coordinates are i.i.d. Bernoulli-Rademacher with parameter p will also be called by the same name. For a square real matrix A, let A° be obtained by zeroing out the diagonal of A. Throughout this paper we use $\|\cdot\|_F$ to denote Frobenius norm, and $\|\cdot\|$ to denote $\ell_2 \to \ell_2$ operator norm.

Both our SJLT analyses in this work show Eq. (2) by analyzing tail bounds for the random variable Z defined in Eq. (3). We continue to use the same notation, where $x \in \mathbb{R}^d$ of unit norm is as in Eq. (3). Our first SJLT analysis uses the following moment bounds for the binomial distribution and for quadratic forms with Rademacher random variables.

▶ **Lemma 2** ([14]). *For Y distributed as* Binomial(N,α) *for integer $N \geq 1$ and $\alpha \in (0,1)$, let $1 \leq p \leq N$ and define $B := p/(\alpha N)$. Then*

$$\|Y\|_p \lesssim \begin{cases} \frac{p}{\log B} & \text{if } B \geq e \\ \frac{p}{B} & \text{if } B < e \end{cases}$$

A more modern, general proof of the below Hanson-Wright inequality can be found in [23].

▶ **Theorem 3** (Hanson-Wright inequality [11]). *For $\sigma_1, \ldots, \sigma_n$ independent Rademachers and $A \in \mathbb{R}^{n\times n}$, for all $q \geq 1$*

$$\|\sigma^T A\sigma - \mathbb{E}\,\sigma^T A\sigma\|_q \lesssim \sqrt{q}\cdot\|A\|_F + q\cdot\|A\|.$$

Our second analysis uses a standard decoupling inequality; a proof is in [25, Remark 6.1.3]

▶ **Theorem 4** (Decoupling). *Let $A \in \mathbb{R}^{n\times n}$ be arbitrary, and $X_1, \ldots, X_n$ be independent and mean zero. Then, for every convex function $F : \mathbb{R} \to \mathbb{R}$*

$$\mathbb{E}\,F(\sum_{i\neq j j} A_{i,j}X_iX_j) \leq \mathbb{E}\,F(4\cdot\sum_{i,j} A_{i,j}X_iX'_j)$$

where the X'_i are independent copies of the X_i.

Before describing the SJLT, we describe the related CountSketch of [8], which was shown to satisfy Eq. (3) in [24]. In this construction for Π, one picks a hash function $h : [d] \to [m]$ from a pairwise independent family, and a function $\sigma : [d] \to \{-1,1\}$ from a 4-wise independent family. Then for each $i \in [d]$, $\Pi_{h(i),i} = \sigma(i)$, and the rest of the ith column is 0. It was shown

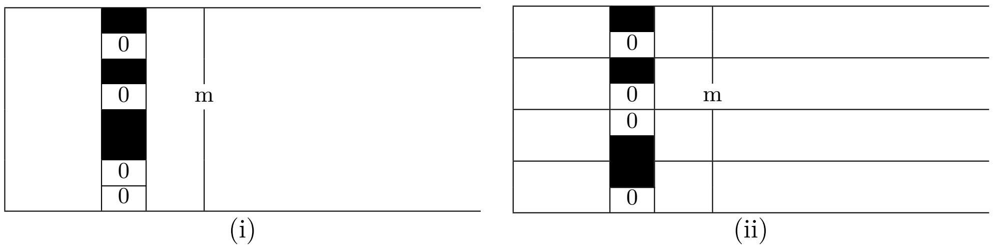

Figure 1 Both distributions have s non-zeroes per column, with each non-zero being independent in $\pm 1/\sqrt{s}$. In (i), they are in random locations, without replacement. (ii) is the CountSketch (with $s > 1$), whose rows are grouped into s blocks of size m/s each, with one non-zero per block per column in a uniformly random location, independent of other blocks; in this example, $m = 8, s = 4$.

in [24] that this distribution satisfies Eq. (3) for $m = \Omega(1/(\varepsilon^2\delta))$. Note that the column sparsity s equals 1. The analysis is simply via Chebyshev's inequality, i.e. bounding the second moment of Z.

The reason for the poor dependence in m on the failure probability δ is that we use Chebyshev's inequality. This is avoided by bounding a higher moment (as in [19], or our first analysis in this work), or by analyzing the moment generating function (MGF) (as in our second analysis in this work). To improve the dependence of m on $1/\delta$, we allow ourselves to increase s.

Now we describe the SJLT. This is a JL distribution over Π having exactly s non-zero entries per column where each entry is a scaled Bernoulli-Rademacher. Specifically, in the SJLT, the random $\Pi \in \mathbb{R}^{m\times d}$ satisfies $\Pi_{r,i} = \eta_{r,i}\sigma_{r,i}/\sqrt{s}$ for some integer $1 \le s \le m$. The $\sigma_{r,i}$ are independent Rademachers and jointly independent of the Bernoulli random variables $\eta_{r,i}$ satisfying:

(a) For any $i \in [d]$, $\sum_{r=1}^m \eta_{r,i} = s$. That is, each column of Π has *exactly* s non-zero entries.

(b) For all $r \in [m], i \in [d]$, $\mathbb{E}\,\eta_{r,i} = s/m$.

(c) The $\eta_{r,i}$ are negatively correlated: $\forall\ S \subset [d]\times[n]$, $\mathbb{E}\prod_{(r,i)\in S}\eta_{r,i} \le \prod_{(r,i)\in S}\mathbb{E}\,\eta_{r,i} = (s/m)^{|S|}$.

See Figure 1 for at least two natural distributions satisfying the above requirements. Thus

$$\|\Pi x\|_2^2 = \frac{1}{s}\sum_{r=1}^m\sum_{i,j=1}^d \eta_{r,i}\eta_{r,j}\sigma_{r,i}\sigma_{r,j}x_ix_j.$$

Using (a) above we have $(1/s)\cdot\sum_r\sum_i \eta_{r,i}x_i^2 = \|x\|_2^2 = 1$, so that

$$Z = \|\Pi x\|_2^2 - 1 = \frac{1}{s}\sum_{r=1}^m\sum_{i\neq j}\eta_{r,i}\eta_{r,j}\sigma_{r,i}\sigma_{r,j}x_ix_j. \tag{4}$$

▶ Remark. In both our analyses, item (a) above is only used to remove the diagonal $i = j$ terms from eq. (4). Thenceforth, it turns out in both analyses of SJLT that (b) and (c) imply we can assume the $\eta_{r,i}$ are fully independent, i.e., the entries of Π are fully independent. This is not the same as saying we can replace the sketch matrix Π with fully independent entries because then part (a) would be violated and it is important for only the "cross" terms in the quadratic form representing Z to be present. In the analysis we justify this assumption by considering the integer moments of Z which we show here cannot decrease by replacement with fully independent entries. For each integer q, each monomial in the expansion of Z^q has expectation equal to $s^{-q}x_{\alpha_1}^{d_1}\cdots x_{\alpha_t}^{d_t}\cdot(\mathbb{E}\prod_{(r,i)\in S}\eta_{r,i})$ whenever all the d_j are even, and S contains all the distinct (r, i) such that $\eta_{r,i}$ appears in the monomial; otherwise the expectation equals 0. Now, $s^{-q}x_{\alpha_1}^{d_1} \cdots x_{\alpha_t}^{d_t}$ is nonnegative, and $\mathbb{E}\prod_{(r,i)\in S}\eta_{r,i} \le (s/m)^{|S|}$. Thus monomials' expectations are term-by-term dominated by the case that all $\eta_{r,i}$ are i.i.d. Bernoulli with expectation s/m.

3 Proof Overview

Hanson-Wright analysis. Note Z can be written as the quadratic form $\sigma^T A_{x,\eta}\sigma$, where $A_{x,\eta}$ is block diagonal with m blocks, where the rth block is $(1/s)x^{(r)}(x^{(r)})^T$ but with the diagonal zeroed out. Here $x^{(r)}$ is the vector with $(x^{(r)})_i = \eta_{r,i}x_i$. To apply Hanson-Wright, we must then bound $\|\|A_{x,\eta}\|_F\|_p$ and $\|\|A_{x,\eta}\|\|_p$, over the randomness of η. This was done in [19], but suboptimally, leading to a simple proof there but of a weaker result (namely, the bound on s proven there was suboptimal by a $\sqrt{\log(1/\varepsilon)}$ factor). As already observed in [19], a simple calculation shows $\|A_{x,\eta}\| \le 1/s$ with probability 1. In this work we improve the analysis of $\|\|A_{x,\eta}\|_F\|_p$ by a simple combination of the triangle and Bernstein inequalities to yield a tight analysis.

MGF analysis. We apply the Chernoff-Rubin bound $\mathbb{P}(|Z| > \varepsilon) \le 2e^{-t\varepsilon}\,\mathbb{E}\cosh(tZ)$, so that we must bound $\mathbb{E}\cosh(tZ)$ (for t in some bounded range) then optimize the choice of t. We accomplish our analysis by writing $Z = X^{\mathsf{T}}A^{\circ}X$ for an appropriate matrix A where X *is* a Bernoulli-Rademacher vector, by Taylor expansion of cosh and considerations similar to Remark 2. We then bound $\mathbb{E}\cosh(tX^{\mathsf{T}}A^{\circ}X)$ using decoupling followed by arguments similar to [13, 23]. We note one can also recover an MGF-based analysis by specializing the analysis of [9] for analyzing sparse oblivious subspace embeddings to the case of "1-dimensional subspaces", though the resulting proof would be quite different from the one presented here. We believe the MGF-based analysis we give in this work appeals to more standard arguments, although the analysis in [9] does provide the advantage that it yields tradeoff bounds for s, m.

4 Our SJLT analyses

4.1 A first analysis: via the Hanson-Wright inequality

▶ **Theorem 5.** *For Π coming from an SJLT distribution, as long as $m \simeq \varepsilon^{-2}\log(1/\delta)$ and $s \simeq \varepsilon m$,*

$$\forall x : \|x\|_2 = 1,\ \mathbb{P}_{\Pi}(|\|\Pi x\|_2^2 - 1| > \varepsilon) < \delta.$$

Proof. As noted, we can write Z as a quadratic form

$$Z = \|\Pi x\|_2^2 - 1 = \frac{1}{s}\sum_{r=1}^{m}\sum_{i\ne j}\eta_{r,i}\eta_{r,j}\sigma_{r,i}\sigma_{r,j}x_i x_j \overset{\text{def}}{=} \sigma^T A_{x,\eta}\sigma,$$

Set $q = \Theta(\log(1/\delta)) = \Theta(s^2/m)$. By Hanson-Wright and the triangle inequality,

$$\|Z\|_q \le \|\sqrt{q}\cdot\|A_{x,\eta}\|_F + q\cdot\|A_{x,\eta}\|\|_q \le \sqrt{q}\cdot\|\|A_{x,\eta}\|_F\|_q + q\cdot\|\|A_{x,\eta}\|\|_q,$$

where $A_{x,\eta}$ is defined in Section 3. Since $A_{x,\eta}$ is block-diagonal, its operator norm is the largest operator norm of any block. The eigenvalue of the rth block is at most $(1/s)\cdot$

$\max\{\|x^{(r)}\|_2^2, \|x^{(r)}\|_\infty^2\} \le 1/s$, and thus $\|A_{x,\eta}\| \le 1/s$ with probability 1. Next, define $Q_{i,j} = \sum_{r=1}^m \eta_{r,i}\eta_{r,j}$ so that

$$\|A_{x,\eta}\|_F^2 = \frac{1}{s^2}\sum_{i\neq j} x_i^2 x_j^2 \cdot Q_{i,j}.$$

Suppose $\eta_{r_1,i}, \ldots, \eta_{r_s,i} = 1$ for distinct r_t and write $Q_{i,j} = \sum_{t=1}^s Y_t$, where Y_t is an indicator random variable for the event $\eta_{r_t,j} = 1$. By Remark 2 we may assume the Y_t are independent, in which case $Q_{i,j}$ is distributed as $\mathsf{Binomial}(s, s/m)$. Then by Lemma 2, $\|Q_{i,j}\|_q \lesssim q$. Thus,

$$\begin{aligned}\|\|A_{x,\eta}\|_F\|_q &= \|\|A_{x,\eta}\|_F^2\|_{q/2}^{1/2}\\ &\le \|\frac{1}{s^2}\sum_{i\neq j} x_i^2 x_j^2 \cdot Q_{i,j}\|_q^{1/2}\\ &\le \frac{1}{s}\left(\sum_{i\neq j} x_i^2 x_j^2 \cdot \|Q_{i,j}\|_q\right)^{1/2} \quad \text{(triangle inequality)}\\ &\le \frac{1}{\sqrt{m}}\end{aligned}$$

Then by Markov's inequality and the settings of q, s, m,

$$\mathbb{P}(|\|\Pi x\|_2^2 - 1| > \varepsilon) = \mathbb{P}(|\sigma^T A_{x,\eta}\sigma| > \varepsilon) < \varepsilon^{-q} \cdot C^q(m^{-q/2} + s^{-q}) < \delta. \qquad ◀$$

▶ Remark. Less general bounds than Lemma 2 would have still sufficed for our purposes. For example, Bernstein's inequality and the triangle inequality together imply $\|Y\|_p \lesssim \alpha N + p$ for any $p \ge 1$, which suffices for our application since we were interested in the case $p = \alpha N$.

4.2 A second analysis: bounding the MGF

In this analysis we show the following bound on the symmetrized MGF of the error:

$$\mathbb{E}\cosh(tZ) \le \exp\left(\tfrac{K^2t^2}{m}\right), \quad \text{for } |t| \le \tfrac{s}{K}, \text{ where } K = 4\sqrt{2} \tag{5}$$

Using the above, we obtain tail estimates in a standard manner. By the generic Chernoff-Rubin bound:

$$\mathbb{P}(|Z| > \varepsilon) \le 2e^{-t\varepsilon}\,\mathbb{E}\cosh(tZ) \le 2\exp\left(\tfrac{K^2t^2}{m} - t\varepsilon\right), \quad \text{for all } 0 \le t \le \tfrac{s}{K}$$

Optimizing over the choice of t, we obtain the tail bound:

$$\mathbb{P}(|Z| > \varepsilon) \le 2\max\{\exp(-C^2\varepsilon^2 m), \exp(-C\varepsilon s)\}, \qquad \text{where } C = \tfrac{1}{8\sqrt{2}}$$

▶ Remark. The cross-over point for the two bounds is when $\frac{s}{m} = \Theta(\varepsilon)$. To obtain a failure probability of δ, this yields the desired $s = O\big(\frac{1}{\varepsilon}\log\big(\frac{1}{\delta}\big)\big)$ and $m = O\big(\frac{1}{\varepsilon^2}\log\big(\frac{1}{\delta}\big)\big)$.

Our goal now is to prove eq. (5) for t satisfying $|t| \le \frac{s}{K}$. Now by Taylor expansion, we have $\mathbb{E}\cosh(tZ) = \sum_{\text{even } q} \frac{|t|^q}{q!} \cdot \mathbb{E}\,Z^q$. Therefore, by section 2, we may assume that the $\eta_{r,i}$ are fully independent to bound $\mathbb{E}\cosh(tZ)$ from above. Now $\mathbb{E}\cosh(tZ) = \frac{1}{2}(\mathbb{E}\exp(tZ) + \mathbb{E}\exp(-tZ)) \le \max\{\mathbb{E}\exp(tZ), \mathbb{E}\exp(-tZ)\}$, for all $t \in \mathbb{R}$. Let $B \overset{\text{def}}{=} \frac{1}{s}xx^\mathsf{T}$. Let $\Pi = \frac{1}{s}H$ and let $Y_1, Y_2, \ldots, Y_m$ denote the rows of H. Then $Z = \sum_{r=1}^m Y_r^\mathsf{T} B^\circ Y_r$. By the independence assumption, Y_i are i.i.d. Bernoulli-Rademacher vectors. Letting Y denote an identical copy of a single row of H,

$$\mathbb{E}\exp(\pm tZ) = \prod_r \mathbb{E}\exp(\pm tY_r^{\mathsf{T}} B^\circ Y_r) = \big(\mathbb{E}\exp(\pm tY^{\mathsf{T}} B^\circ Y)\big)^m, \quad \text{for all } t \in \mathbb{R} \tag{6}$$

Let Y' be an independent copy of Y. By decoupling (Theorem 4),

$$\mathbb{E}\exp(tY^{\mathsf{T}} B^\circ Y) \le \mathbb{E}\exp(4tY^{\mathsf{T}} BY') = \mathbb{E}\exp(Y^{\mathsf{T}} \tilde{B} Y'), \quad \text{for all } t \in \mathbb{R}, \text{ where } \tilde{B} \overset{\text{def}}{=} 4tB \tag{7}$$

We show below that

$$\mathbb{E}\exp(Y^{\mathsf{T}} \tilde{B} Y') \le 1 + \tfrac{K^2t^2}{m^2}, \quad \text{provided } |t| \le \tfrac{s}{K}, \text{ where } K = 4\sqrt{2} \tag{8}$$

Substituting this bound in eq. (7) and combining with eq. (6), we obtain:

$$\mathbb{E}\exp(\pm tZ) \le \big(1 + \tfrac{K^2t^2}{m^2}\big)^m \le \exp\big(\tfrac{K^2t^2}{m}\big), \quad \text{provided } |t| \le \tfrac{s}{K}, \text{ where } K = 4\sqrt{2},$$

which completes the proof of (5) as desired. It remains to prove eq. (8).

Bilinear forms of Bernoulli-Rademacher random variables.

The MGF of a Bernoulli-Rademacher random variable $X = \eta\sigma$ with parameter p equals $\mathbb{E}\exp(tX) = 1 - p + p\,\mathbb{E}\exp(t\sigma) \le 1 - p + p\exp(t^2/2)$, for all $t \in \mathbb{R}$.

Let $\lambda(z) \overset{\text{def}}{=} \exp(z) - 1$. Rewriting the above, we have $\mathbb{E}\,\lambda(tX) \le p\,\lambda(t^2/2) = p\,\mathbb{E}\,\lambda(tg)$, where $g \sim \mathcal{N}(0,1)$. We show an analogous replacement inequality for Bernoulli-Rademacher vectors.

▶ **Lemma 6.** *Let Y be a Bernoulli-Rademacher vector with parameter p. Then:*

$$\mathbb{E}\,\lambda(b^{\mathsf{T}} Y) \le p\,\lambda(\|b\|^2/2) = p\,\mathbb{E}\,\lambda(b^{\mathsf{T}} g) \quad \textit{for all vectors } b, \textit{ where } g \sim \mathcal{N}(0, I_n)$$

Proof. By stability of Gaussians, $\mathbb{E}\exp(b^{\mathsf{T}} g) = \exp(\|b\|_2^2/2)$, demonstrating the last equality above. Let $g(t) \overset{\text{def}}{=} \sum_{S \ne \emptyset} t^{|S|-1} \prod_{i \in S} \lambda(b_i^2/2)$ for $t \ge 0$. We have $\prod_i \big(1 + t\,\lambda(b_i^2/2)\big) = 1 + tg(t)$. Now:

$$\mathbb{E}\exp(b^{\mathsf{T}} Y) = \prod_i \mathbb{E}\exp(b_iY_i) = \prod_i \big(1 + \mathbb{E}\,\lambda(b_iY_i)\big) \le \prod_i \big(1 + p\,\lambda(b_i^2/2)\big) = 1 + pg(p)$$

Thus, $\mathbb{E}\,\lambda(b^{\mathsf{T}} Y) \le pg(p) \le pg(1)$, since $g(t)\!\uparrow$. To conclude, we claim that $g(1) = \lambda(\|b\|_2^2/2)$. Indeed:

$$1 + g(1) = \prod_i (1 + \lambda(b_i^2/2)) = \prod_i \exp(b_i^2/2) = \exp\big(\sum_i b_i^2/2\big) = 1 + \lambda\big(\|b\|_2^2/2\big)$$ ◀

Let $p \overset{\text{def}}{=} \frac{s}{m}$. In the left side of eq. (8), we have $\mathbb{E}\exp(Y^{\mathsf{T}} \tilde{B} Y') = 1 + \mathbb{E}\,\lambda(Y^{\mathsf{T}} \tilde{B} Y')$. By the law of total expectation:

$$\mathop{\mathbb{E}}_{Y,Y'} \lambda(Y^{\mathsf{T}} \tilde{B} Y') = \mathop{\mathbb{E}}_{Y}\mathop{\mathbb{E}}_{Y'}[\lambda((Y^{\mathsf{T}} \tilde{B})Y') \mid Y] \le \; p \cdot \mathop{\mathbb{E}}_{Y}\mathop{\mathbb{E}}_{g'}[\lambda((Y^{\mathsf{T}} \tilde{B})g') \mid Y]$$

(by lemma 6, applied to Y')

Exchange the order of expectations of Y and g' via Fubini-Tonelli's theorem. Now apply lemma 6, this time to Y. Finish using the law of total expectation which yields an upper bound of $p^2 \cdot \mathbb{E}\,\lambda(g^{\mathsf{T}} \tilde{B} g')$. Thus:

$$\mathbb{E}\exp(Y^{\mathsf{T}} \tilde{B} Y') \le 1 + p^2 \cdot \mathbb{E}\,\lambda(g^{\mathsf{T}} \tilde{B} g') \tag{9}$$

In order to be self-contained we include a standard proof of the following lemma, though note that the lemma itself is equivalent to the Hanson-Wright inequality for gaussian random variables since it gives a bound on the MGF of decoupled quadratic forms in gaussian random variables.

▶ **Lemma 7.** $\mathbb{E}\exp(g^\mathsf{T} Q g') \le \exp\big(\|Q\|_F^2\big)$ *for independent* $g, g' \sim \mathcal{N}(0, I_n)$*, provided* $\|Q\| \le \frac{1}{\sqrt{2}}$.

Proof. Let $Q = U\Sigma V^\mathsf{T}$, where $\Sigma = \mathsf{diag}(s_1, \dots, s_n)$. So $\mathbb{E}\exp(g^\mathsf{T} Q g') = \mathbb{E}\exp(g^\mathsf{T} U\Sigma V^\mathsf{T} g')$. Since U is orthonormal, by rotational invariance, $U^\mathsf{T} g \sim \mathcal{N}(0, I_n)$ and is independent of $V^\mathsf{T} g' \sim \mathcal{N}(0, I_n)$. Therefore, $\mathbb{E}\exp(g^\mathsf{T} Q g') = \mathbb{E}\exp(g^\mathsf{T} \Sigma g')$. Now $g^\mathsf{T}\Sigma g' = \sum_i s_i g_i g_i'$, therefore:

$$\mathbb{E}\exp(g^\mathsf{T}\Sigma g') = \prod_i \mathbb{E}\,\mathbb{E}[\exp(s_i g_i g_i') \mid g_i] = \prod_i \mathbb{E}\exp(s_i^2 g_i^2/2) = \prod_i \tfrac{1}{\sqrt{1-s_i^2}}$$

Now $s_i^2 \le \|Q\|^2 \le \frac{1}{2}$ for each i. Use the bound $e^{-x} \le \sqrt{1-x}$ for $x \le \frac{1}{2}$ so that:

$$\mathbb{E}\exp(g^\mathsf{T} Q g') \le \prod_i \exp(s_i^2) = \exp(\sum_i s_i^2) = \exp\big(\|Q\|_F^2\big) \qquad ◀$$

Note that $\|\tilde{B}\|_F = 4t\|B\|_F$ and $\|\tilde{B}\| = 4t\|B\|$. Now $B = \frac{1}{s} x x^\mathsf{T}$, so that $\|B\|_F = \|B\| = \frac{1}{s}$. Using the above proposition in the right side of eq. (9) with $Q = \tilde{B}$, we obtain:

$$\mathbb{E}\exp(Y^\mathsf{T}\tilde{B}Y') \le 1 + p^2 \cdot \lambda\big(\tfrac{K^2 t^2}{2s^2}\big), \quad \text{provided } |t| \le \tfrac{s}{K}, \text{ where } K = 4\sqrt{2}$$

In the right side above, use the bound $\lambda(x) \le 2x$, which holds for $x \le \frac{1}{2}$, and substitute $p = \frac{s}{m}$ so that

$$\mathbb{E}\exp(Y^\mathsf{T}\tilde{B}Y') \le 1 + \tfrac{K^2 t^2}{m^2}, \quad \text{provided } |t| \le \tfrac{s}{K}, \text{ where } K = 4\sqrt{2}$$

This yields the desired bound stated in eq. (8).

References

1 Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

2 Nir Ailon and Bernard Chazelle. The fast Johnson-Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.

3 Nir Ailon and Edo Liberty. Fast dimension reduction using Rademacher series on dual BCH codes. *Discrete & Computational Geometry*, 42(4):615–630, 2009.

4 Nir Ailon and Edo Liberty. An almost optimal unrestricted fast Johnson-Lindenstrauss transform. *ACM Trans. Algorithms*, 9(3):21:1–21:12, 2013.

5 Noga Alon and Bo'az Klartag. Optimal compression of approximate inner products and dimension reduction. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2017.

6 Jean Bourgain. An improved estimate in the restricted isometry problem. *Geometric Aspects of Functional Analysis*, Lecture Notes in Mathematics Volume 2116:65–70, 2014.

7 Vladimir Braverman, Rafail Ostrovsky, and Yuval Rabani. Rademacher chaos, random Eulerian graphs and the sparse Johnson-Lindenstrauss transform. *CoRR*, abs/1011.2590, 2010.

8 Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

9 Michael B. Cohen. Nearly tight oblivious subspace embeddings by trace inequalities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 278–287, 2016.

10 Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A sparse Johnson-Lindenstrauss transform. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 341–350, 2010.

11 David Lee Hanson and Farroll Tim Wright. A bound on tail probabilities for quadratic forms in independent random variables. *Ann. Math. Statist.*, 42:1079–1083, 1971.

12 Ishay Haviv and Oded Regev. The restricted isometry property of subsampled Fourier matrices. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 288–297, 2016.

13 Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Trans. Algorithms*, 3(3):31, 2007.

14 Meena Jagadeesan. Simple analysis of sparse, sign-consistent JL. *CoRR*, abs/1708.02966, 2017.

15 T. S. Jayram and David P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Trans. Algorithms*, 9(3):26:1–26:17, 2013.

16 William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

17 Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit Johnson-Lindenstrauss families. In *Proceedings of the 15th International Workshop on Randomization and Computation (RANDOM)*, pages 628–639, August 2011.

18 Daniel M. Kane and Jelani Nelson. A derandomized sparse Johnson-Lindenstrauss transform. *CoRR*, abs/1006.3585, 2010.

19 Daniel M. Kane and Jelani Nelson. Sparser Johnson-Lindenstrauss transforms. *J. ACM*, 61(1):4, January 2014. Preliminary version in SODA 2012.

20 Felix Krahmer and Rachel Ward. New and improved Johnson-Lindenstrauss embeddings via the Restricted Isometry Property. *SIAM J. Math. Anal.*, 43(3):1269–1281, 2011.

21 Kasper Green Larsen and Jelani Nelson. Optimality of the Johnson-Lindenstrauss lemma. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2017.

22 Jelani Nelson, Eric Price, and Mary Wootters. New constructions of RIP matrices with fast multiplication and fewer rows. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1515–1528, 2014.

23 Mark Rudelson and Roman Vershynin. Hanson-Wright inequality and sub-gaussian concentration. *Electronic Communications in Probability*, 18:1–9, 2013.

24 Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

25 Roman Vershynin. *High-Dimensional Probability.* May 2017. Last accessed at `http://www-personal.umich.edu/~romanv/papers/HDP-book/HDP-book.pdf` on August 22, 2017.