2nd Symposium on Simplicity in Algorithms

SOSA 2019, January 8–9, 2019, San Diego, CA, USA
Co-located with the *30th ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*

Edited by

Jeremy T. Fineman
Michael Mitzenmacher

Editors

Jeremy T. Fineman
Georgetown University
Washington, DC, USA
jfineman@cs.georgetown.edu

Michael Mitzenmacher
Harvard University
Cambridge, MA, USA
michaelm@eecs.harvard.edu

ACM Classification 2012
Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-099-6

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/978-3-95977-099-6.

Publication date
January, 2019

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.2019.0

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 2190-6807

http://www.dagstuhl.de/oasics

Contents

Preface
Jeremy T. Fineman and Michael Mitzenmacher 0:vii

Organisation
..... 0:ix

Regular Papers

Isotonic Regression by Dynamic Programming
Günter Rote 1:1–1:18

An Illuminating Algorithm for the Light Bulb Problem
Josh Alman 2:1–2:11

Simple Concurrent Labeling Algorithms for Connected Components
Sixue Liu and Robert E. Tarjan 3:1–3:20

A Framework for Searching in Graphs in the Presence of Errors
Dariusz Dereniowski, Stefan Tiegel, Przemysław Uznański, and Daniel Wolleb-Graf 4:1–4:17

Selection from Heaps, Row-Sorted Matrices, and $X + Y$ Using Soft Heaps
Haim Kaplan, László Kozma, Or Zamir, and Uri Zwick 5:1–5:21

Approximating Optimal Transport With Linear Programs
Kent Quanrud 6:1–6:9

LP Relaxation and Tree Packing for Minimum k-cuts
Chandra Chekuri, Kent Quanrud, and Chao Xu 7:1–7:18

On Primal-Dual Circle Representations
Stefan Felsner and Günter Rote 8:1–8:18

Asymmetric Convex Intersection Testing
Luis Barba and Wolfgang Mulzer 9:1–9:14

Relaxed Voronoi: A Simple Framework for Terminal-Clustering Problems
Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi 10:1–10:14

Towards a Unified Theory of Sparsification for Matching Problems
Sepehr Assadi and Aaron Bernstein 11:1–11:20

A New Application of Orthogonal Range Searching for Computing Giant Graph Diameters
Guillaume Ducoffe 12:1–12:7

Simplified and Space-Optimal Semi-Streaming $(2 + \varepsilon)$-Approximate Matching
Mohsen Ghaffari and David Wajc 13:1–13:8

Simple Greedy 2-Approximation Algorithm for the Maximum Genus of a Graph
Michal Kotrbčík and Martin Škoviera 14:1–14:9

2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy T. Fineman and Michael Mitzenmacher
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Note on Max k-Vertex Cover: Faster FPT-AS, Smaller Approximate Kernel and Improved Approximation
Pasin Manurangsi 15:1–15:21

Simple Contention Resolution via Multiplicative Weight Updates
Yi-Jun Chang, Wenyu Jin, and Seth Pettie 16:1–16:16

A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum
Ce Jin and Hongxun Wu 17:1–17:6

Submodular Optimization in the MapReduce Model
Paul Liu and Jan Vondrak 18:1–18:10

Compressed Sensing with Adversarial Sparse Noise via L1 Regression
Sushrut Karmalkar and Eric Price 19:1–19:19

Approximating Maximin Share Allocations
Jugal Garg, Peter McGlaughlin, and Setareh Taki 20:1–20:11

Preface

The first Symposium on Simplicity in Algorithms proved a remarkable success. Our goal in the second year was to see if this excitement was limited or lasting, and if it was lasting, to keep the momentum going.

We received 68 submissions, and accepted 20 papers. There was widespread feeling we could have easily accepted at least five more papers without compromising the quality; in the end, as is generally the case, we made some hard decisions. The high quality of the submissions reflects that there needs to be a home for papers that other conferences may dismiss for being "simple". As described in last year's preface, this workshop was established with the idea of making simplicity and elegance in the design and analysis of algorithms its main objectives. We believe this perspective continues to make this workshop a worthwhile endeavor.

We do continue to struggle with determining what constitutes simplicity and elegance, which leads to interesting discussions in the program committee! But generally with this focus we seek to advance understanding of an algorithmic problem in a way that will make it more accessible to a larger audience. In some cases, that may mean providing an argument you could teach in an undergraduate course. In some cases, that may mean taking something complex and of narrow interest, and making it less complex and of wider interest. We therefore have papers that cover quite a range of topics and difficulty, but all are remarkably interesting.

We would like to thank members of the program committee for putting in the hard work, made somewhat harder by the process of this workshop continuing to find its identity. We also thank the members of the steering committee for establishing this workshop; it has provided something the community needed, but had somehow not managed to create.

We hope SOSA will continue for many years to come.

Michael Mitzenmacher
Jeremy T. Fineman
Program Committee Chairs

2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy T. Fineman and Michael Mitzenmacher

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organisation

Program Committee

Susanne Albers, *Technische Universität München*
Chandra Chekuri, *University of Illinois at Urbana-Champaign*
Michael Dinitz, *Johns Hopkins University*
Jeff Erickson, *University of Illinois at Urbana-Champaign*
Martin Farach-Colton, *Rutgers University*
Jeremy T. Fineman (co-chair), *Georgetown University*
Michael Goodrich, *University of California, Irvine*
Philip Klein, *Brown University*
Andrew McGregor, *University of Massachusetts, Amherst*
Michael Mitzenmacher (co-chair), *Harvard University*
Jelani Nelson, *Harvard University*
Rasmus Pagh, *IT University of Copenhagen*
Ely Porat, *Bar-Ilan University*
Eric Price, *University of Texas at Austin*
Tim Roughgarden, *Stanford University*
Robert Sedgewick, *Princeton University*
Yaron Singer, *Harvard University*
Eva Tardos, *Cornell University*
Justin Thaler, *Georgetown University*

Steering Committee

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

2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy T. Fineman and Michael Mitzenmacher

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Isotonic Regression by Dynamic Programming

Günter Rote
Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany
rote@inf.fu-berlin.de
https://orcid.org/0000-0002-0351-5945

Abstract

For a given sequence of numbers, we want to find a monotonically increasing sequence of the same length that best approximates it in the sense of minimizing the weighted sum of absolute values of the differences. A conceptually easy dynamic programming approach leads to an algorithm with running time $O(n \log n)$. While other algorithms with the same running time are known, our algorithm is very simple. The only auxiliary data structure that it requires is a priority queue. The approach extends to other error measures.

2012 ACM Subject Classification Theory of computation → Dynamic programming, Theory of computation → Computational geometry, Mathematics of computing → Regression analysis

Keywords and phrases Convex functions, dynamic programming, convex hull, isotonic regression

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.1

1 Problem Statement: Weighted Isotonic L_1 Regression

Weighted isotonic L_1 regression (or weighted isotonic median regression) is the following problem:

> Approximate a given sequence of numbers $a = (a_1, \ldots, a_n)$ with weights $w_i > 0$ by an increasing sequence
>
> $$z_1 \le z_2 \le \cdots \le z_n, \tag{1}$$
>
> minimizing the weighted L_1-error
>
> $$\sum_{i=1}^{n} w_i \cdot |z_i - a_i|. \tag{2}$$

If the input sequence $(a_1, \ldots, a_n)$ has decreasing sections, the optimum solution values z_i tend to cluster together in *runs* or *level sets* of equal values $z_j = z_{j+1} = \cdots = z_k$, see Figure 1. This common value z is the weighted median of the corresponding elements $a_j, a_{j+1}, \ldots, a_k$, because this is the value that minimizes $\sum_{i=j}^{k} w_i|z - a_i|$.

Isotonic regression has applications in many fields, including statistics and production planning. The problem has been studied for a long time, see [2] for an early monograph, and there is a large literature that treats many variations of the problem. In particular, there are algorithms that solve the weighted L_1 regression problem in $O(n \log n)$ time [1, 8]. These algorithms will be reviewed in Section 11.

2 Our Algorithm

We present a new algorithm that is based on the dynamic programming method. Our approach differs somewhat from standard use of this technique, as the intermediate objects of our dynamic programming recursion are real functions, and thus infinite objects.

2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy Fineman and Michael Mitzenmacher; Article No. 1; pp. 1:1–1:18

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

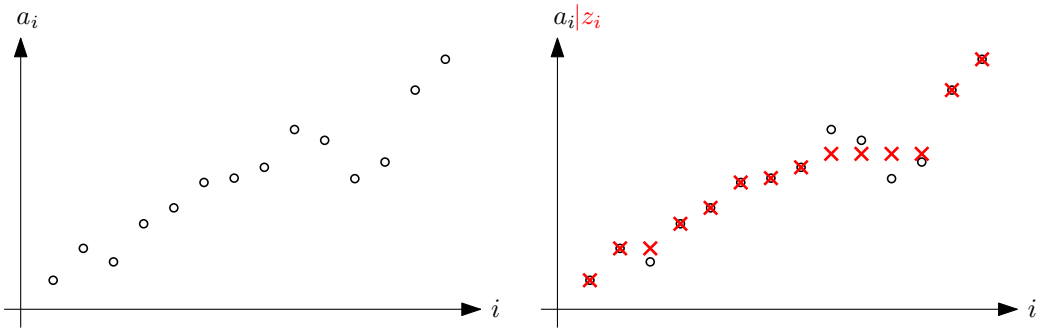


Figure 1 The original data sequence $(a_1, \ldots, a_n)$ is shown on the left. The crosses on the right form a monotone approximation $(z_1, \ldots, z_n)$. It contains two runs that are longer than a single element.

This is not a revolutionary idea. Historically, the concept of dynamic programming and Bellman's optimality principle applies also to continuous processes such as rocket flight. And while the notion of "dynamic programming" has made an independent career as an algorithmic design principle in computer science, it continues to be used in optimal control and in discrete as well as continuous optimization.

Computer scientists, on the other hand, tend to shun continuous and infinite structures. They need not be afraid: The functions that arise in our problem turn out to be piecewise linear functions, and they can be treated as peaceful discrete objects. In the programming contest literature, this approach is known under the name "convexity dynamic programming", see Section 12.

With the proper choice of representation, our approach leads to a simple algorithm with running time $O(n \log n)$. The most sophisticated data structure that is needed for an efficient implementation is a standard priority queue. Other merits of our algorithm are discussed in Section 11.1. Despite its simplicity, the algorithm would be mysterious without the conceptual background of its design (see Algorithm 3).

3 The Dynamic Programming Setup

We consider the subproblems

$$f_k(x) := \min\Big\{ \sum_{i=1}^{k} w_i \cdot |z_i - a_i| : z_1 \le z_2 \le \cdots \le z_k = x \Big\} \tag{3}$$

for $k = 1, \ldots, n$ and a real parameter x. We get the following straightforward dynamic programming recursion, including $k = 0$ as the base case:

$$\begin{aligned} f_k(x) &:= \min\{ f_{k-1}(z) : z \le x \} + w_k \cdot |x - a_k| \quad (k = 1, \ldots, n;\ x \in \mathbb{R}) \\ f_0(x) &:= 0 \qquad (x \in \mathbb{R}) \end{aligned} \tag{4}$$

The following sections develop the details of how to implement this recursion.

4 The Functions f_k

▶ **Lemma 1.**

(a) *Each function f_k is a piecewise linear convex function, for $0 \le k \le n$.*

(b) *The breakpoints are located at a subset of the points $a_1, a_2, \ldots, a_k$.*

(c) *The leftmost piece has slope $-\sum_{i=1}^{k} w_i$. The rightmost piece has slope w_k.*

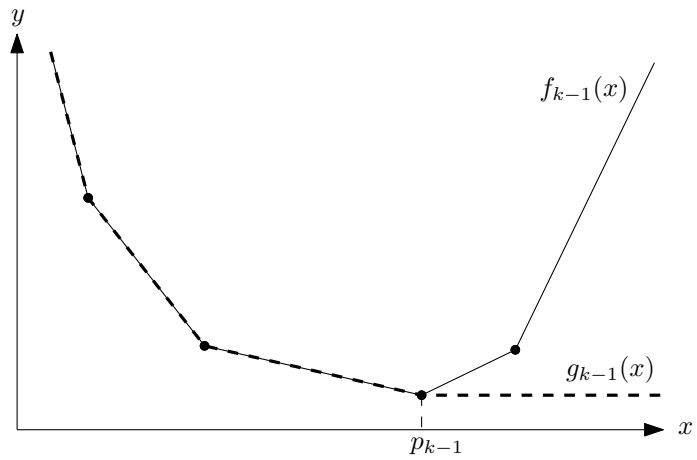


Figure 2 Constructing g_{k-1} from f_{k-1}.

Proof. These properties are easily established by induction. The base cases ($k = 0$ and $k = 1$) are obvious. We denote by

$$g_{k-1}(x) := \min\{\, f_{k-1}(z) : z \le x \,\}$$

the intermediate function in the transition from f_{k-1} to f_k.

Let $k \ge 2$, and let us assume by induction that all properties of the lemma hold for f_{k-1}. The function f_{k-1} is first monotonically decreasing to a minimum and then monotonically increasing. We denote by p_{k-1} the (not necessarily unique) position where the minimum occurs. The optimum of $f_{k-1}(z)$ under the constraint $z \le x$ depends on the position of x relative to p_{k-1}: If $x \le p_{k-1}$ then $z = x$ is the optimum choice, and $g_{k-1}(x) = f_{k-1}(x)$. If $x \ge p_{k-1}$ then the optimum choice is $z = p_{k-1}$, and $g_{k-1}(x) = f_{k-1}(p_{k-1})$, see Figure 2.

As a consequence of this, we get the following relation between the values z^*_{k-1} and z^*_k in the optimal solution:

$$z^*_{k-1} = \min\{z^*_k, p_{k-1}\} \tag{5}$$

This will be useful for recovering the optimal regression after its objective function value, i.e., the regression error, has been obtained.

In summary, *the function g_{k-1} has the same decreasing pieces as f_{k-1} but the increasing pieces are replaced by a horizontal piece of constant value $f_{k-1}(p_{k-1})$.*

Finally, to obtain f_k, we add the piecewise linear function $w_k \cdot |x - a_k|$ to g_{k-1}. It is now easy to see that f_k has the claimed properties of the lemma. ◀

5 Representing Piecewise Linear Functions

The most natural representation for a continuous piecewise linear function f would be a sorted list of breakpoints x_i with their function values $f(x_i)$, plus the slopes of the two unbounded pieces on the left and on the right. However, looking back at the discussion of the previous paragraph, the recursion (4) involves the addition of a piecewise linear function to another. The natural representation would then require all slopes to be updated.

We therefore prefer to maintain slope *differences* rather than the slopes themselves. We represent a piecewise linear function as a list of *breakpoints*, see Figure 3. Each breakpoint has a *position* – the x-value where it is located – and a *value* – the slope difference between

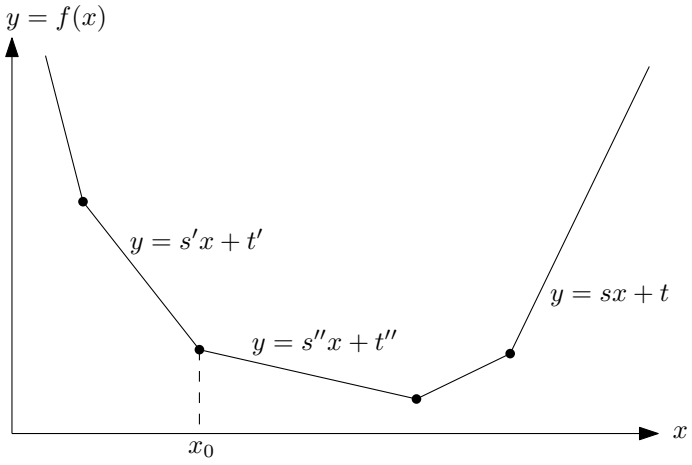


Figure 3 A piecewise linear function with four breakpoints. There is a breakpoint at position x_0 with value $s'' - s'$.

the right and the left adjacent pieces. The breakpoints are naturally ordered by position, but for the time being, we leave it unspecified whether we want to store them as a sorted list or in some other data structure. The function is convex if all breakpoints have nonnegative values.

The breakpoint data determine the function f only up to addition of an arbitrary linear function. We must specify two further parameters. Since the transition from f_{k-1} to g_{k-1} involves inspections and modifications at the right end of the function, it is most convenient to take the slope s and the intercept t of the *rightmost* linear piece $y = sx + t$.

This determines f uniquely: We proceed from right to left, and across each breakpoint, the value of the breakpoint gives us the slope $\hat{s}$ of next linear piece $y = \hat{s}x + \hat{t}$, and continuity of f allows us to fix the intercept $\hat{t}$.

Two functions are *added* by combining the list of breakpoints and adding the (s, t) parameters. If several breakpoints have the same position, they might be merged into one breakpoint, adding their values. However, this would require equal breakpoints to be found, and is not necessary; our algorithm will handle equal breakpoints just as well.

6 Carrying out the Recursion (4)

Recall from Section 4 that the function g_{k-1} has the same decreasing pieces as f_{k-1} but the increasing pieces are replaced by a horizontal piece of constant value $f_{k-1}(p_{k-1})$ and slope 0.

Algorithm 1 performs this transformation. It removes the increasing pieces from the right end of f_{k-1} one by one.

In representing the functions, we have to deal only with the *slope* s of the rightmost piece; the intercept t is not needed. Since the leftmost slope is negative, by Lemma 1c, the while-loop will terminate, and the list of breakpoints will never become empty. If f_{k-1} has a horizontal piece, the algorithm will arbitrarily choose the leftmost minimum p_{k-1}.

Finally, to obtain f_k, we must add the function $w_k \cdot |x - a_k|$ to g_{k-1}: This amounts to creating an additional breakpoint of value $2w_k$ at position a_k, and adding w_k to s.

Algorithm 1: Converting f_{k-1} to g_{k-1}.

Input: List of breakpoints of f_{k-1} and rightmost slope s
Result: Updated list of breakpoints of g_{k-1} and rightmost slope s; position p_{k-1} of the (leftmost) minimum of f_{k-1}
Let B be the rightmost breakpoint;
while $s - B.value \geq 0$ **do** // next-to-last piece is not decreasing
  $s := s - B.value$; // remove the last piece
  Delete B from the list of breakpoints;
  Let B be the rightmost remaining breakpoint;
$p_{k-1} := B.position$; // p_{k-1} is the position of the minimum of f_{k-1}.
$B.value := B.value - s$; // make the rightmost piece horizontal
$s := 0$;

7 The Weighted Regression Algorithm

We see that the algorithm only needs to access the rightmost breakpoint, and potentially delete it. A new breakpoint is inserted for each new data point a_k. This calls for a (max-)priority queue for storing the breakpoints, using *position* as the key. We use the standard priority queue operations *insert*, *findmax* (taking constant time), and *deletemax*.

Algorithm 2 shows the complete algorithm that we can now put together. The organization is slightly different from the recursion (4): an iteration of the main loop starts from g_{k-1}, turns it into f_k, and then into g_k. We start with the function $g_0(x) = 0$. The algorithm records the minimum position p_k for each function f_k. In the last loop iteration, the minimum of f_n is found as part of the construction of g_n.

Finally, the optimum solution values z_i are computed in a simple loop according to (5), starting with the minimum $z_n = p_n$ of the function f_n.

The variable s is always 0 at the beginning of the loop. Hence we can simplify the program. Also, it is advantageous to switch to the negative variable $\bar{s} \equiv -s$, because this turns all remaining subtractions into additions and makes these operations more transparent. The final computation of the optimal z_i values needs no change. The modified Algorithm 3 is shown below. Its inner loop can now be interpreted as follows: the variable $\bar{s}$ is initialized with the value $-w_k$; it accumulates and deletes the values from the top of the queue until the total value becomes positive.

Figure 4 shows the algorithm at work. The values in the queue Q are shown at selected times as if it were an ordered list with the highest keys a_i at the top. The first two iterations are not very interesting: A breakpoint of value $2w_k$ is inserted and, since it is at the top of Q, it is immediately reduced to w_k. The iteration $k = 3$ is shown in more detail: After $2w_3$ is inserted, $\bar{s}$ starts at $-w_3$. $\bar{s}$ is combined with the value w_2 at the top of the list, but the result, $w_2 - w_3$, is still negative. So it is combined with $2w_3$ to give $w_3 + w_2$, which is positive, and the iteration is completed.

8 Runtime Analysis

In total, n elements are inserted in the queue Q. Each iteration of the while-loop removes an element from Q, and therefore the overall number of executions of the while-loop is bounded by n. With a heap data structure for Q, each operation *deletemax* or *insert* can be carried out in $O(\log n)$ time. The *findmax* operation takes only constant time. The priority queue is not affected by the manipulation of the *values*, since it is ordered by *position*. Hence, the overall running time is $O(n \log n)$.

Algorithm 2: Dynamic Programming Algorithm for weighted isotonic L_1 regression.

```
Q := ∅; // priority queue of breakpoints ordered by the key position
s := 0;
for k := 1, ..., n do
    // We start from g_{k-1}.
    Q.insert(new breakpoint B with B.position := a_k, B.value := 2w_k);
    s := s + w_k; // We have computed f_k.
    B := Q.findmax;
    while s − B.value ≥ 0 do
        s := s − B.value;
        Q.deletemax;
        B := Q.findmax;
    p_k := B.position; // record the position of the minimum
    B.value := B.value − s;
    s := 0; // We have computed g_k.
// compute the optimal solution z_1, ..., z_n:
z_n := p_n;
for k := n − 1, n − 2, ..., 1 do
    z_k := min{z_{k+1}, p_k};
```

Algorithm 3: Dynamic Programming Algorithm for weighted isotonic L_1 regression, simplified version.

```
Q := ∅; // priority queue of breakpoints ordered by the key position
for k := 1, ..., n do
    Q.insert(new breakpoint B with B.position := a_k, B.value := 2w_k);
    s̄ := −w_k;
    B := Q.findmax;
    while s̄ + B.value ≤ 0 do
        s̄ := s̄ + B.value;
        Q.deletemax;
        B := Q.findmax;
    B.value := s̄ + B.value;
    p_k := B.position; // record the position of the minimum
// The solution z_1, ..., z_n is computed in the same way as in Algorithm 2.
```

▶ **Theorem 2.** *Algorithm 3 (the Dynamic Programming Algorithm) solves the weighted isotonic L_1 regression problem in $O(n \log n)$ time and $O(n)$ space.*

9 Unweighted Regression

In the unweighted case ($w_i \equiv 1$), some steps can be simplified: The variable $\bar{s}$ is always -1 before the while-loop. Thus, it can be eliminated and the while-loop turned into an if-loop. Breakpoints have value 1 or 2. Algorithm 4 shows the simplified version. It is possible to

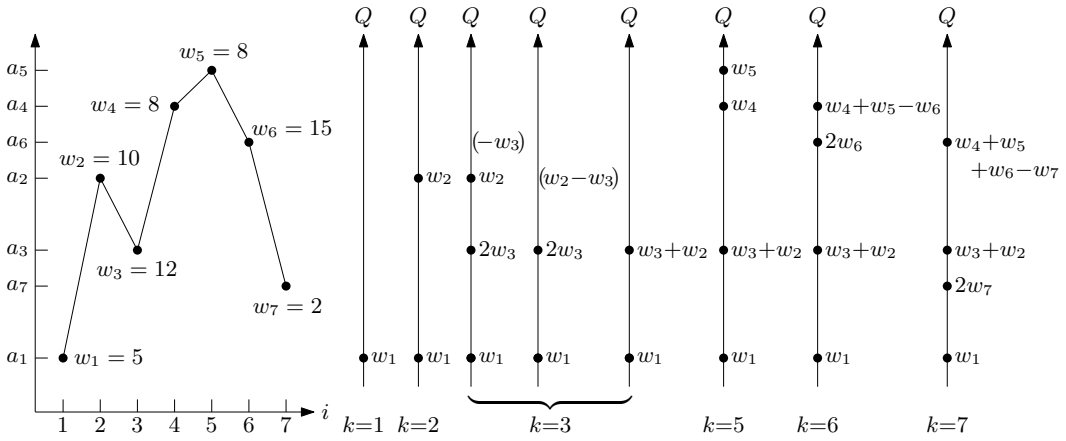


Figure 4 Running Algorithm 3 on an example.

Algorithm 4: Unweighted isotonic L_1 regression by dynamic programming.

```
Q := ∅; // priority queue of breakpoints ordered by the key position
for k := 1, ..., n do
    Q.insert(new breakpoint B with B.position := a_k, B.value := 2);
    B := Q.findmax;
    if B.value = 1 then
        Q.deletemax;
        B := Q.findmax;
    else
        B.value := 1;
    p_k := B.position;
// The solution z_1, ..., z_n is computed in the same way as in Algorithm 2.
```

write an even simpler algorithm by eliminating the *value* attribute altogether: Instead of a breakpoint of value 2, we insert two (unweighted) breakpoints at the same position. The resulting main loop has no if-statements and needs only five lines.

10 Other Error Measures: Weighted Isotonic L_2 Regression

Instead of the L_1-error, one can consider other objective functions, where the absolute value is replaced by a different convex function h:

$$\sum_{i=1}^{n} w_i \cdot h(z_i - a_i). \tag{6}$$

More generally, one can allow a separate error measure h_i for each data point:

$$\sum_{i=1}^{n} h_i(z_i). \tag{7}$$

In this setting, there is no need for a_i and w_i, because these data can be incorporated in h_i. The most commonly considered case is the (squared) L_2-error:

$$E_2 = \sum_{i=1}^{n} w_i \cdot (z_i - a_i)^2 \tag{8}$$

It is straightforward to extend our approach to this objective function.

Exercises.

1. Show that the functions f_k defined in analogy to (3) for the L_2-case are piecewise quadratic convex functions. Explore their further properties, in analogy to Lemma 1.
2. Design an appropriate efficient data structure for representing this class of functions.
3. Show how to solve the dynamic programming recursion in $O(n)$ overall time, using only a stack as a data structure.
4. Explain which properties of the objective function, $h(z-a) = |z-a|$ versus $h(z-a) = (z-a)^2$, are responsible for the difference between the runtime of $O(n \log n)$ versus $O(n)$.
5. Compare your algorithm to the Incremental PAV Algorithm of Stout [8, Fig. 7 in connection with Fig. 5] and find out whether the two algorithms carry out essentially the same calculations.
6. Adapt the algorithm to the L_4 error, $h(x) = x^4$.
7. Show that the algorithm can be extended to handle arbitrary piecewise polynomial functions h_i in (7), provided they are convex.
8. Prove that the solution is unique if the functions h_i are strictly convex.

See Appendix A for answers.

11 Other Algorithms

The most popular approach for the isotonic regression problem is the algorithm *Pool Adjacent Violators* (PAV), see for example [2] or [8]. This classical method, which has often been rediscovered, starts by combining adjacent values that are not monotone ($a_i > a_{i+1}$) into pairs, and it further combines runs into larger runs as long as the weighted medians of adjacent groups are out of order. This algorithm extends to quite general settings like (7), provided that the functions h_i are convex.

Ahuja and Orlin [1] gave the first $O(n \log n)$ algorithm for weighted isotonic L_1 regression. It is based on the PAV principle but uses scaling for speedup. The core of this *Scaling PAV* algorithm is a procedure to turn a solution for the data $\lfloor a_i/2 \rfloor$ into a solution for the original data a_i in linear time (assuming that the a_i are integral). To get a running time that is independent of the range of values, the algorithm replaces the given values a_i by $1, \ldots, n$ while keeping their relative order fixed.

Stout [8] has given a direct implementation of the PAV approach. I will refer to his algorithm as the *Incremental PAV* algorithm, because it adds the elements one at a time and completes the necessary PAV updates before looking at the next element. It requires mergeable trees (for example, AVL trees or 2-3-trees, see [5]), to achieve a running time of $O(n \log n)$.

Another natural approach is to model the problem as a minimum-cost network flow problem, see Figure 5. (I could not track down a specific source for this in the literature.) The unknown approximation values z_i are flow values along a path. Each inequality $z_i \le z_{i+1}$ becomes a flow conservation constraint, with an additional entering arc taking the slack. Since flow is nonnegative, we have to assume that all $a_i \ge 0$, which is no loss of generality. In

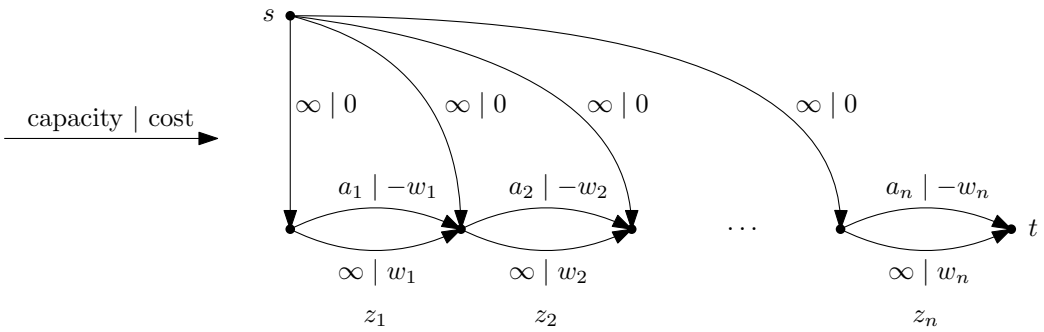


Figure 5 Minimum-cost network flow from s to t.

order to model the piecewise linear costs, each flow z_i is distributed over two parallel edges: a cheaper edge with bounded capacity and a more expensive edge with unbounded capacity. We denote by z_i the combined flow of these two edges. Then the cost of a flow $(z_1, \dots, z_n)$ in this network is

$$\begin{aligned} c(z_1, \dots, z_n) &= \sum_{i=1}^{n} \big[-w_i \min\{z_i, a_i\} + w_i \max\{z_i - a_i, 0\}\big] \\ &= \sum_{i=1}^{n} \big[-w_i(\min\{z_i - a_i, 0\} + a_i) + w_i \max\{z_i - a_i, 0\}\big] \\ &= \sum_{i=1}^{n} \big[w_i \max\{a_i - z_i, 0\} + w_i \max\{z_i - a_i, 0\} - w_i a_i\big] \\ &= \sum_{i=1}^{n} w_i \max\{a_i - z_i, z_i - a_i\} - \sum_{i=1}^{n} w_i a_i, \end{aligned}$$

which differs from the original approximation error (2) just by the constant $\sum_{i=1}^{n} w_i a_i$.

This network is series-parallel, and hence the minimum-cost flow can be found in $O(n \log n)$ time by an algorithm of Booth and Tarjan [4]. However, this algorithm also relies on mergeable trees, and moreover, it needs $O(n \log^* n)$ space to recover the optimum solution. So this approach is not preferable to Stout's algorithm. The algorithm follows the dynamic programming paradigm, and thus, in spirit, it is closer to the algorithm of this paper. We suspect that a closer study of the algorithm for the specialized network structure of Figure 5, instead of applying it out of the box, might have led to the discovery of our Dynamic Programming Algorithm.

Ahuja and Orlin [1] mistakenly credit [6] for an earlier $O(n \log n)$ time algorithm, but that paper has only an algorithm with $O(n \log^2 n)$ runtime.

11.1 Comparison

11.1.1 Simplicity

The Incremental PAV Algorithm of Stout [8] involves tree data structures (for example, AVL trees or 2-3-trees) augmented with weight information, and needs the nonstandard merging operation: two trees of size m and n with $m \le n$ have to be merged in $O(m \log \frac{n}{m}) = O(\log \binom{m+n}{n})$ time.

If might be instructive to compare the algorithms on the example of Figure 4. After item 6 is added, items 4–6 form a run $z_4 = z_5 = z_6$. In the Incremental PAV Algorithm, the three elements have been combined into a mergeable tree, in order to compute their weighted

median. Our Dynamic Programming Algorithm, by contrast, has somehow taken note of the run by forming the sum $w_4 + w_5 - w_6$. On the other hand, it still keeps $2w_6$ as a separate item. Thus, it is probably difficult to explain one algorithm in terms of the other, and the distinction between the two algorithms is more fundamental.

The Scaling PAV Algorithm of Ahuja and Orlin [1], on the other hand, needs no data structures beyond arrays and linked lists. The algorithm itself, however, is not so simple. Moreover, it requires an initial sort of the elements.

The Dynamic Programming Algorithm is very simple and requires just a priority queue, in which each element is inserted once and retrieved at most once. Thus, the priority queue has to perform the same *insert* operations and fewer *deletemax* operations than would be required for heapsort; one might expect that the Dynamic Programming Algorithm is finished while the Scaling PAV Algorithm is still busy in its sorting phase.

11.1.2 Incremental Computation (Prefix Regression)

The Dynamic Programming Algorithm processes data as they arrive, producing the solution for the first k items after reading them. (To have the objective function value (2) always ready, the algorithm must be extended to deal with the intercept t in addition to the slope s. The most convenient way to do this is to maintain the value $f_k(p_k)$.)

Stout [8] has called this problem the *prefix isotonic regression* problem: solving the regression problem for all prefixes of the input. His Incremental PAV Algorithm solves this problem readily in $O(n \log n)$ time. Stout [8, p. 295] notes that this is optimal because prefix isotonic regression can be used for sorting. He uses it as a subroutine for the *unimodal* regression problem. Our algorithm can also be used for this purpose.

Ahuja and Orlin's Scaling PAV Algorithm is not suitable for incremental computation.

11.1.3 Numerical Precision

The common solution value $z_i = z_{i+1} = \cdots = z_k$ of each run is the weighted median. Thus, any algorithm that solves the problem necessarily has to compare sums of the form $\sum_{i \in I} w_i$ in order to compute weighted medians.

In our algorithm, the k-th iteration inserts a new entry of value $2w_k$ into the queue. In addition, the starting value $\bar{s} = -w_k$ is added with some entries from the top of the queue. Everything that is calculated is built from the ground set of $2n$ elements $2w_i$ and $-w_i$ by adding subsets of these elements together in some hierarchical order. All results that are ever computed are therefore of the form $2w_i$ or of the form $\sum_{i=1}^{n} e_i w_i$, where $e_i \in \{0, 1, -1\}$, and the values of the latter form are compared with 0.

The term $+w_i$ in these expressions is formed by adding $2w_i$ and $-w_i$. This incurs a slight loss of precision of 1 bit in terms of weights, when compared with the calculations that any algorithm must necessarily perform for solving the problem.

11.1.4 Data Sensitivity

Another question is how the algorithm responds to input sequences that are almost sorted. This is a natural assumption in statistical applications, where the data "ought" to be monotone but is distorted by noise.

The Incremental PAV method will have an advantage, since values a_i that are in the correct order with respect to their neighbors and are approximated by themselves ($z_i = a_i$) will be looked at only once. Moreover, runs will usually be short, and in the $O(\log n)$ bound on the tree operations, the parameter n can be replaced by the run length.

The Scaling PAV Algorithm of Ahuja and Orlin [1], on the other hand, is completely insensitive to the data: in addition to sorting, it will always perform $\Theta(\log n)$ linear-time sweeps over the data.

The Dynamic Programming Algorithm might potentially benefit from almost sorted data. At least in the case when the input comes in truly sorted order, the algorithm will never call *deletemax*. To take advantage of almost sorted data, one would need a priority queue where it is cheaper to retrieve (by *findmax*) and delete elements that have been inserted recently.

12 Convexity Dynamic Programming

One referee has pointed out that the technique that we advocate is known in the programming contest literature under the name "convexity dp" (for "convexity dynamic programming"). In fact, the unweighted problem (Section 9) has become quite a standard problem in programming contests: It was used as problem *SEQUENCE* in the 2004 Balkan Olympiad for Informatics[1], and was even solved during the contest by one high school student, Filip Wolski, with an $O(n \log n)$ solution. Numerous programs that use just a few lines of code and implement the algorithm of Section 9 can be seen at the *Codeforces* programming contest platform[2].

Another problem of the same flavor, which can also be solved using convexity, has been posed in 2009 under the name *CCROSSX – Cross Mountain Climb Extreme*[3]. Here, the approximating sequence is not required to be increasing, but it is restricted to have a bounded difference between successive elements: $|z_i - z_{i+1}| \leq d$, for some given bound d.

We are grateful to the referee for the pointers to the programming contest community.

References

1 R. K. Ahuja and J. B. Orlin. A fast scaling algorithm for minimizing separable convex functions subject to chain constraints. *Operations Research*, 49:784–789, 2001. doi:10.1287/opre.49.5.784.10601.

2 R. E. Barlow, D. J. Bartholomew, J. M. Bremner, and H. D. Brunk. *Statistical Inference under Order Restrictions*. Wiley, 1972.

3 R. E. Barlow and H. D. Brunk. The Isotonic Regression Problem and its Dual. *Journal of the American Statistical Association*, 67(337):140–147, 1972. doi:10.1080/01621459.1972.10481216.

4 Heather Booth and Robert Endre Tarjan. Finding the minimum-cost maximum flow in a series-parallel network. *J. Algorithms*, 15:416–446, 1993. doi:10.1006/jagm.1993.1048.

5 M. R. Brown and R. E. Tarjan. A fast merging algorithm. *J. Assoc. Comp. Mach.*, 26:211–226, 1979. doi:10.1145/322123.322127.

6 P. M. Pardalos, G. L. Xue, and Y. Li. Efficient computation of an isotonic median regression. *Applied Math. Letters*, 8(2):67–70, March 1995. doi:10.1016/0893-9659(95)00013-G.

7 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry. An Introduction*. Springer, 1985.

8 Quentin F. Stout. Unimodal regression via prefix isotonic regression. *Comp. Stat. and Data Anal.*, 53:289–297, 2008. doi:10.1016/j.csda.2008.08.005.

[1] http://www.boi2004.lv/Uzd_diena1.pdf
[2] https://codeforces.com/contest/13/status/C
[3] https://www.spoj.com/problems/CCROSSX/

A Weighted Isotonic L_2 Regression

We state without proof the properties of the functions f_k that arise for the L_2 objective function.

▶ **Lemma 3.**

(a) *f_k is a piecewise quadratic convex function.*

(b) *The derivative $f'_k(x)$ is a piecewise linear increasing and concave function. In particular, it is continuous.*

(c) *The leftmost piece of $f'_k(x)$ has slope $2\sum_{i=1}^{k} w_i$. The rightmost piece has slope $2w_k$.*

In contrast to the L_1 case (Lemma 1), the breakpoints are not restricted to the points a_i.

The transition from f_{k-1} to g_{k-1} requires the elimination of the increasing part at the right rim. It is easier to carry this out at the level of the derivative: To go from f'_{k-1} to g'_{k-1}, we eliminate the positive part and replace it by the constant 0 function, see Figure 6. Afterwards, in order to produce f'_k, we increment g_{k-1} by the derivative of $w_k(x-a_k)^2$, which is the linear function $2w_k(x-a_k)$.

If we represent the quadratic pieces of the form $bx^2 - 2cx + d$ by triplets (b, c, d), we can arrange things so that the algorithm performs almost the same calculations as the Incremental PAV Algorithm of Stout [8, Fig. 7].

Like for the case of the L_1 norm (Section 5), the necessity to add functions suggests that we store differences $(\Delta b, \Delta c, \Delta d)$ between successive pieces instead of the values themselves. In fact, the differences Δd of the constant terms are redundant, because two adjacent quadratic pieces must touch at some point where they have a common tangent. In other words, the graph of the difference function $\Delta b \cdot x^2 - 2\Delta c \cdot x + \Delta d$ must touch the x-axis, and therefore $\Delta d = \Delta c^2/\Delta b$. The fact that Δd is irrelevant is also evident from the fact that f_k, its derivative being f'_k (Figure 6), is determined by f'_k up to one parameter, the integration constant. The d_i coefficients don't show up in f'_k, and therefore their differences play no role in determining f_k.

When cutting away the positive branch of the function f'_{k-1}, the algorithm must decide whether the rightmost piece, $y = 2b_1x - 2c_1$, should be completely eliminated, see Figure 6. This is the case if its intersection X with the second piece $y = 2b_2x - 2c_2$ lies to the right of the intersection with the x-axis. As shown in the illustration, this amounts to the comparison $\frac{\Delta c}{\Delta b} \geq \frac{c_1}{b_1}$.

Now, the (b, c) and $(\Delta b, \Delta c)$ values are initially created from the quadratic functions $w_kx^2 - 2w_ka_k + a_k^2 = bx^2 - 2cx + d$, and thus $(b, c) = (w_k, w_ka_k)$ During the algorithm, adjacent $(\Delta b, \Delta c)$ values are pooled together, and they become expressions of the form $\Delta b = \sum_{i=j}^{k} w_i$, $\Delta c = \sum_{i=j}^{k} w_ia_i$. These are the same as the quantities *sumw* and *sumwy* that are maintained in [8, Fig. 7]. The test $\frac{\Delta c}{\Delta b} \geq \frac{c_1}{b_1}$ is nothing but a comparison of weighted averages. Thus, indeed, the core of the Dynamic Programming Algorithm performs the same calculations as Stout's Incremental PAV Algorithm.

There are, however, some slight differences.

(i) The Incremental PAV Algorithm updates the optimal solution value E_2 directly in one step, after all positive parts of f'_k have been eliminated. For this purpose, it maintains the variables $sumwy2 = \sum_{i=j}^{k} w_ia_i^2$. In our Algorithm 3, the update of the objective function is not detailed, but it would be most natural to update it with each iteration of the while-loop.

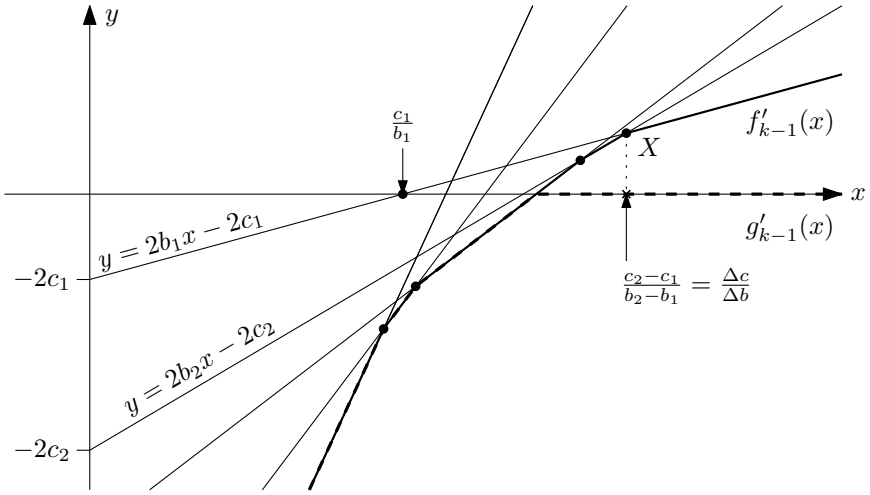


Figure 6 Transforming f'_{k-1} into g'_{k-1}.

(ii) Another difference is the determination of the optimal solution z. Stout's Incremental PAV Algorithm is straightforward: it simply sets each variable z_i to the weighted average of its run. The Dynamic Programming Algorithm, on the other hand, computes the solution by the formula (5) and achieves the same result in an indirect way, see the last part of Algorithm 2.

The reason why the L_1-regression requires $O(n \log n)$ time and the L_2-regression does not (Question 4 in Section 10) is that the absolute value function in the L_1 objective inserts breakpoints of its own, whereas the L_2 objective function is smooth, and breakpoints are created only at the right end when replacing the increasing part of f_k by a flat part.

A.1 Geometric Interpretation: Lower Envelope and Lower Convex Hull

If we look at the area below the graphs f_k and g_k, the algorithm can be interpreted geometrically as an alternating succession of the following two operations.

- Intersect the area with the negative halfplane.
- Apply an affine transformation (in particular, a shearing transformation).

We can eliminate the affine transformations and carry out all intersection operations in the original coordinate system. The problem reduces to an intersection of lower halfplanes, which result from applying the appropriate shearing transformation to the negative halfplanes, as shown in Figure 7.

▶ **Theorem 4.** *The weighted isotonic L_2 regression problem of minimizing* (8) *subject to the monotonicity constraints* (1) *can be solved by constructing the lower envelope of $n+1$ lines $F_0, F_1, \ldots, F_n$, which are given by*

$$F_k\colon\ y = 2\sum_{i=1}^{k} w_i a_i - \Big(2\sum_{i=1}^{k} w_i\Big)x.$$

If two lines F_j and F_k form adjacent edges on the lower envelope, then the optimum solution has a run $z_{j+1} = z_{j+2} = \cdots = z_k$ of values that are equal to the x-coordinate of the intersection point between F_j and F_k.

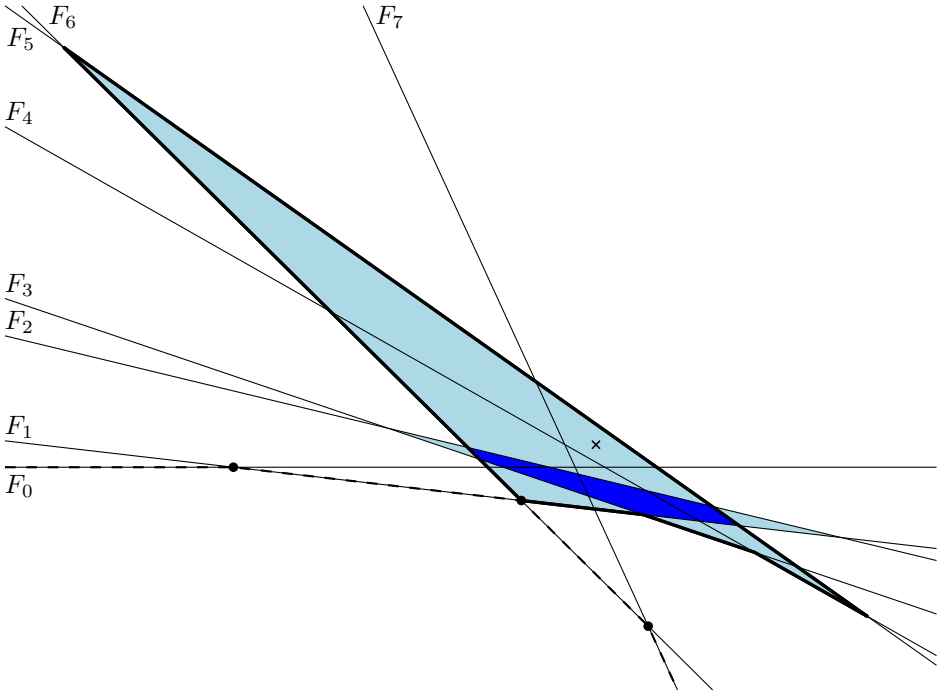


Figure 7 The L_2 isotonic regression problem as the lower envelope of lines. The approximation error E_2 equals the shaded area, with the dark shaded area counted twice.

Moreover, the approximation error E_2 of the lines can be expressed as a weighted sum of certain face areas in the line arrangement. For a face in the arrangement, we record a sequence of $n+1$ pluses and minuses, depending on whether the face lies above F_k $(+)$ or below F_k $(-)$. For example, the face marked by a cross in Figure 7 has the sequence $+++++-++$. If there are r runs of consecutive pluses, then the area of this face is counted with multiplicity $\max\{r-1, 0\}$.

This is of course not the best way to compute the approximation error E_2. It can easily be computed in linear time by substituting the optimal solution into (8). The program of Stout [8, Fig. 7] computes the approximation error incrementally in a more direct way.

The above formula arose by working out how the objective function changes when k is incremented. It turns out that one has to add a certain integral, which, in terms of the arrangement of the lines F_i, equals the area that lies (i) above F_k, (ii) below F_{k-1}, and (iii) above the lower envelope of $F_0, F_1, \ldots, F_{k-1}$. Figure 7 shows this area for $k = 6$ with a heavy outline. (Bear in mind that the line F_k or F_{k-1}, respectively, corresponds to the x-axis in the original setting of Figure 6.) Pursuing this further, one can show that the approximation error can be expressed as the sum of at most $n-1$ triangle areas. This can be translated into the weighted sum of face areas that was given above.

Using a well-known geometric duality transform, the lower-envelope problem turns into the problem of computing the lower convex hull (or *greatest convex minorant*) of a set of points: The line $y = ax + b$ becomes the point $(-a, b)$ and the point (u, v) becomes the line $y = ux + v$. This duality preserves incidences and above/below relations. After canceling the factor 2 from the point coordinates, we arrive at the following result, see Figure 8.

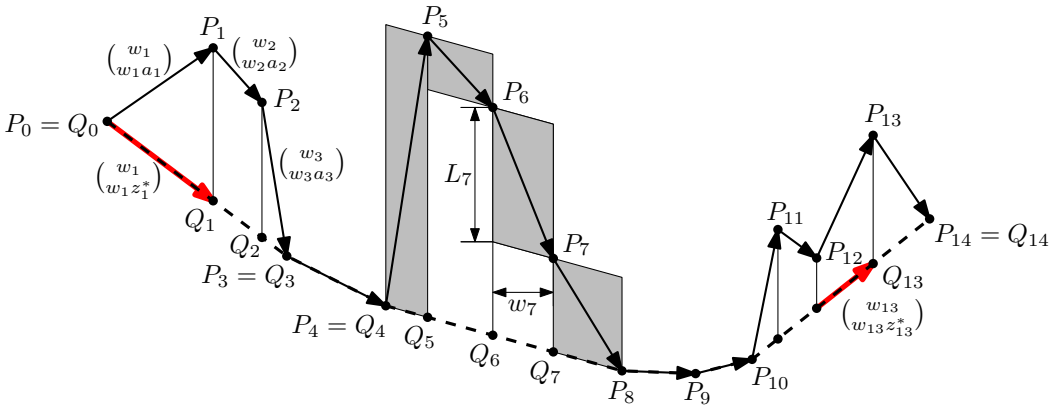


Figure 8 The regression problem as a lower convex hull. Some points Q_i on the lower hull are marked, together with two selected vectors $Q_{i-1}Q_i$; z_i^* denotes the convex hull solution.

▶ **Theorem 5.** *Consider a polygonal chain $P_0P_1 \dots P_n$ whose segments are given by the vectors $P_i - P_{i-1} = \binom{w_i}{w_i a_i}$.*

The weighted isotonic L_2 regression problem of minimizing (8) *subject to the monotonicity constraints* (1) *can be solved by constructing the lower convex hull of the chain $P_0P_1 \dots P_n$.*

If P_jP_k is an edge of the lower convex hull, then the optimum solution has a run $z_{j+1} = z_{j+2} = \dots = z_k$ of values that are equal to the slope of the edge P_jP_k.

This surprising connection between an isotonic regression problem and a basic computational geometry problem has been known for a long time [2, Section 1.2, mentioning earlier sources in Section 1.6], see also [7, Section 4.2.2]. Barlow and Brunk [3, Section 4.1, the *taut-string solution*], derived it as an instance of convex-programmining duality: The polygon $P_0P_1 \dots P_n$ is called the *cumulative sum diagram*. The set of lower envelopes $Q_0 \dots Q_n$ of $P_0 \dots P_n$ with endpoints $Q_0 = P_0$ and $Q_n = P_n$ forms a cone which is dual to the cone of increasing functions (1). From this, Theorem 5 (and the stronger inequality (11) below) follows by easy duality arguments [3, Theorem 2.1, in particular (2.4)]. A streamlined self-contained proof of Theorem 5, which does not depend on this background, is contained in the monograph of Barlow, Bartholomew, Bremner, and Brunk [2, Theorem 1.1]. We reproduce this proof in Appendix C. It appears as a clever algebraic manipulation, exploiting the fact that the optimum solution is unchanged ($z_i = z_{i+1}$) whenever the lower hull passes below P_i, see the complementarity condition (12) in Appendix C.

Since the vertices P_i are given in sorted order, the lower convex hull can be computed in linear time. The standard incremental algorithm for this task becomes the same as the Dynamic Programming Algorithm or Stout's Incremental PAV method for this case.

The approximation error E_2 is not so easy to figure out in this representation. One might be tempted to believe that it is area of the pockets between the polygonal chain and its convex hull, but this is not true: This area depends locally only linearly on the data, while the error function is quadratic. Here is an attempt at a geometric interpretation of the objective function: Enclose each edge $P_{i-1}P_i$ in a vertical parallelogram, with two sides parallel to the convex hull edge under P_{i-1} and P_i. Figure 8 shows a few of these parallelograms. If such a parallelogram has horizontal width w_i and vertical edges of length L_i, it contributes $L_i^2 w_i$ to the objective function. The objective function is the total contribution of all n edges. This is of course not a deep statement; we just measure the squared deviation of each step a_i from the average of each run, i.e., z_i.

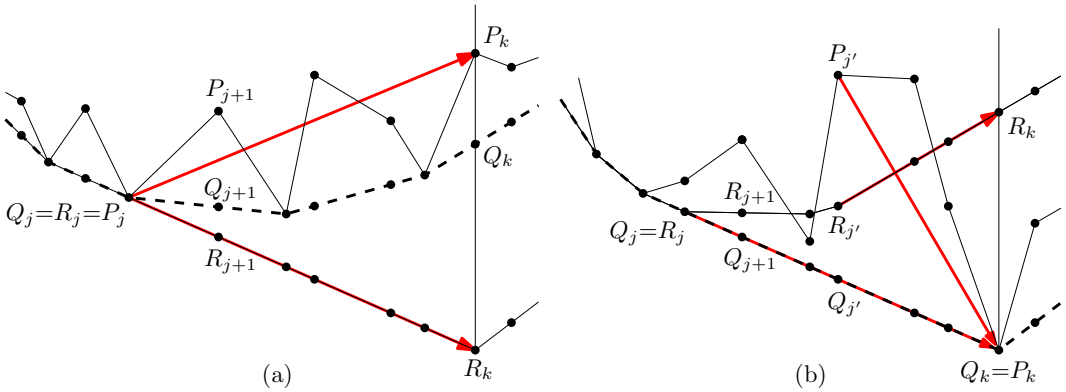


Figure 9 (a) Case 1. $z_{j+1} < z^*_{j+1}$. (b) Case 2. $z_{j+1} > z^*_{j+1}$. The segments whose slopes are compared in the proof are highlighted.

B Direct Proof of Theorem 5

For completeness, and for comparison, we give an independent elementary proof of the correspondence between isotonic L_2 regression and the convex hull (Theorem 5). This pedestrian proof does not assume any optimization background, and I hope it gives some more direct geometric insight into the correspondence.

The only tool that we need is the elementary fact that the best L_2-approximation by a run of equal values $z_j = z_{j+1} = \cdots = z_k = z$ is the weighted average:

▶ **Proposition 6.** *Let* $\mu = \sum_{i=j}^{k} w_i a_i / \sum_{i=j}^{k} w_i$ *be the weighted average of the sequence* $a_j, \ldots, a_k$. *Then*

$$\sum_{i=j}^{k} w_i (z - a_i)^2 = \sum_{i=j}^{k} w_i (\mu - a_i)^2 + \sum_{i=j}^{k} w_i (z - \mu)^2 \tag{9}$$

In particular, if we regard the left-hand side of (9) *as a function of* z, *it is a quadratic function with a unique minimum at* $z = \mu$.

If we start the polygonal chain at $P_0 = \binom{0}{0}$, the coordinates of the points $P_i = \binom{W_i}{A_i}$ are the partial sums $W_k = \sum_{i=1}^{k} w_i$ and the weighted partial sums $A_k = \sum_{i=1}^{k} w_i a_i$, for $0 \le k \le n$. The crucial observation is that the weighted average of a subsequence $a_j, \ldots, a_k$, which plays a prominent role in Proposition 6, shows up as the slope of the segment $P_{j-1}P_k$.

We denote the claimed optimal solution by z^*_i, and we set $Z^*_k = \sum_{i=1}^{k} w_i z^*_i$. Then the points $Q_i = \binom{W_i}{Z^*_i}$ form the lower convex hull $Q_0 Q_1 \ldots Q_n$: The vector $Q_i - Q_{i-1} = \binom{w_i}{w_i z^*_i}$ has slope z^*_i, and Q_i is the point where the vertical line through P_i intersects the lower hull, see Figure 8. Accordingly, $Z^*_i \le A_i$ for $i = 0, \ldots, n$. The endpoints are fixed to $Q_0 = P_0 = \binom{0}{0}$ and $Q_n = P_n$, and thus, $Z^*_0 = A_0 = 0$ and $Z^*_n = A_n$.

For comparison, we consider an arbitrary increasing sequence z_i. We define accordingly $Z_k = \sum_{i=1}^{k} w_i z_i$ and the points $R_i = \binom{W_i}{Z_i}$, forming the polygonal chain $R_0 R_1 \ldots R_n$.

We will now show that a sequence $(z_1, \ldots, z_n) \ne (z^*_1, \ldots, z^*_n)$ cannot be optimal. The idea is to identify a subsequence of equal consecutive values and to show that they can be jointly modified to improve the objective function, while keeping the sequence increasing.

Let us suppose that the two sequence agree up to z_j. We distinguish whether z_{j+1} is smaller or larger than z^*_{j+1}.

Case 1. $z_{j+1} < z^*_{j+1}$, and accordingly, R_{j+1} lies below Q_{j+1}, see Figure 9a. Let us take the maximum $k \le n$ such that $z_{j+1} = z_{j+2} = \cdots = z_k$. In other words, R_k is the next vertex after R_j. The slope of $R_jR_{j+1}\dots R_k$ is the common value $z = z_{j+1} = \cdots = z_k$. Since $R_j \dots R_k$ forms a straight line segment and $Q_j \dots Q_k$ is convex, Q_k must lie above R_k, and therefore P_k must also lie above R_k. Q_j is a vertex of the lower hull because $z^*_{j+1} > z_{j+1} \ge z_j = z^*_j$. Thus, $Q_j = R_j = P_j$. The weighted average of $a_{j+1}, \dots, a_k$, which is the slope of the segment P_jP_k, is therefore larger than z (the slope of $R_j \dots R_k$). It follows from Proposition 6 that the objective function can be improved by increasing the common value z of $z_{j+1} = z_{j+2} = \cdots = z_k$. By the maximal choice of k, some small increase of z is always possible without violating the monotonicity of the sequence.

Case 2. $z_{j+1} > z^*_{j+1}$, and accordingly, R_{j+1} lies above Q_{j+1}, see Figure 9b. Then R_j must be a vertex of the convex chain $R_0 \dots R_n$, because $z_{j+1} > z^*_{j+1} \ge z^*_j = z_j$.
We choose the largest $k \le n$ such that $z^*_j = z^*_{j+1} = \cdots = z^*_k$. In other words, Q_k is the next vertex after Q_j on the lower hull, and thus $Q_k = P_k$. Now we choose the smallest $j' \ge 0$ such that $z_{j'+1} = z_{j'+2} = \cdots = z_k$. In other words, we extend the segment $R_{k-1}R_k$ to the left until we hit the previous vertex $R_{j'}$ of the convex chain. Since R_j is a vertex, as just observed, we have $j' \ge j$. Since $P_{j'}$ lies on or above the segment Q_jQ_k,

$$\text{slope}(P_{j'}P_k) = \text{slope}(P_{j'}Q_k) \le \text{slope}(Q_jQ_k). \tag{10}$$

On the other hand, $R_jR_{j+1}\dots R_k$ branches off upwards from the segment Q_jQ_k and forms a convex chain with increasing slopes. Thus, the slope of $R_{j'}R_k$ is strictly larger than the slope of Q_jQ_k. With (10), we conclude that the slope of $R_{j'}R_k$ (the common value $z := z_{j'+1} = z_{j'+2} = \cdots = z_k$) is larger than the slope of the segment $P_{j'}P_k$ (the weighted average of $a_{j'+1}, \dots, a_k$). Proposition 6 implies that the objective function can be improved by decreasing z. By the minimal choice of j', some small decrease of z is possible without violating the monotonicity of the sequence.

We have thus shown that a sequence different from $(z^*_1, \dots, z^*_n)$ cannot be optimal. From here, there are two ways to conclude the proof of Theorem 5. (a) By observing that the objective function E_2 is continuous and goes to infinity when the argument $(z_1, \dots, z_n)$ becomes unbounded, we establish that an optimum solution exists. Since $(z^*_1, \dots, z^*_n)$ is the only solution that is not excluded by our arguments, it must be the optimum solution.

(b) If a constructive proof is preferred, one can extend the above argument with little additional effort to a procedure that transforms any solution $R_0 \dots R_n$ into the lower hull $Q_0 \dots Q_n$ in $O(n)$ steps without increasing the approximation error. ◀

C The proof of Barlow, Bartholomew, Bremner, and Brunk (1972)

For the convenience of the reader, and for comparison, we reproduce the short and elegant proof of Theorem 5 from Barlow, Bartholomew, Bremner, and Brunk [2, Theorem 1.1, pp. 12–13], translated to our notation, with a few more details and and with a minor correction and improvement. They proved the following inequality:

$$\sum_{i=1}^{n} w_i \cdot (z_i - a_i)^2 \ge \sum_{i=1}^{n} w_i \cdot (z^*_i - a_i)^2 + \sum_{i=1}^{n} w_i \cdot (z_i - z^*_i)^2 \tag{11}$$

To establish the optimality of z^*, we would only need to compare the L_2-errors of an arbitrary increasing sequence z against z^* (the first two sums). The inequality (11) gives a stronger statement than needed for this, because of the additional quadratic term on the right, which measures the deviation between z and z^*. With this term, uniqueness of the optimal solution follows directly.

We use the setup with the polygons $P_0 \dots P_n$ and $Q_0 \dots Q_n$ and their coordinates $P_i = \binom{W_i}{A_i}$ and $Q_i = \binom{W_i}{Z_i^*}$, which were introduced in Appendix B for the first proof. We will need one more observation: If the lower hull lies strictly below P_i, then the hull edge passes straight through Q_i. In other words:

$$Z_i^* < A_i \implies z_i^* = z_{i+1}^* \tag{12}$$

Let us now go into the calculation: The inequality (11) has the form $\sum(u_i + v_i)^2 \geq \sum u_i^2 + \sum v_i^2$, and the difference between the left side and the right side is $2\sum u_i v_i$. In our case, this is

$$D = 2\sum_{i=1}^{n} w_i(z_i^* - a_i)(z_i - z_i^*) = 2\sum_{i=1}^{n}(z_i^* - z_i)(w_i a_i - w_i z_i^*).$$

We want to show that this is nonnegative. We apply the partial summation formula

$$\sum_{i=1}^{n} g_i(H_i - H_{i-1}) = \sum_{i=1}^{n-1}(g_i - g_{i+1})H_i + g_n H_n - g_1 H_0 \tag{13}$$

with $g_i = z_i^* - z_i$ and $H_i - H_{i-1} = w_i a_i - w_i z_i^*$, and therefore $H_i = A_i - Z_i^*$. In our case, $H_0 = A_0 = Z_0^* = 0$, and $H_n = 0$ because $Z_n^* = A_n$. Thus, the two boundary terms in (13) vanish, and we get

$$\begin{aligned} D &= 2\sum_{i=1}^{n-1}\big[(z_i^* - z_i) - (z_{i+1}^* - z_{i+1})\big](A_i - Z_i^*) \\ &= 2\sum_{i=1}^{n-1}(z_{i+1} - z_i)(A_i - Z_i^*) - 2\sum_{i=1}^{n-1}(z_{i+1}^* - z_i^*)(A_i - Z_i^*). \end{aligned}$$

The first sum is nonnegative because $z_{i+1} \geq z_i$ and $A_i \geq Z_i^*$. The second sum is 0 because of the complementarity relation (12). ◀

An Illuminating Algorithm for the Light Bulb Problem

Josh Alman[1]
MIT CSAIL, Cambridge, MA, USA
jalman@mit.edu

Abstract

The Light Bulb Problem is one of the most basic problems in data analysis. One is given as input n vectors in $\{-1,1\}^d$, which are all independently and uniformly random, except for a planted pair of vectors with inner product at least $\rho \cdot d$ for some constant $\rho > 0$. The task is to find the planted pair. The most straightforward algorithm leads to a runtime of $\Omega(n^2)$. Algorithms based on techniques like Locality-Sensitive Hashing achieve runtimes of $n^{2-O(\rho)}$; as ρ gets small, these approach quadratic.

Building on prior work, we give a new algorithm for this problem which runs in time $O(n^{1.582} + nd)$, *regardless* of how small ρ is. This matches the best known runtime due to Karppa et al. Our algorithm combines techniques from previous work on the Light Bulb Problem with the so-called 'polynomial method in algorithm design,' and has a simpler analysis than previous work. Our algorithm is also easily derandomized, leading to a deterministic algorithm for the Light Bulb Problem with the same runtime of $O(n^{1.582} + nd)$, improving previous results.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms, Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Light Bulb Problem, Polynomial Method, Finding Correlations

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.2

Acknowledgements The author would like to thank Vitaly Feldman, Michael P. Kim, Virginia Vassilevska Williams, Ryan Williams, and anonymous reviewers for their comments on an earlier draft.

1 Introduction

In this paper, we study the problem of finding correlated vectors. Finding correlations is one of the most basic problems in data analysis. In many experiments, one gathers data about a number of different variables, and then one would like to determine which variables are correlated. By forming the vector of data points for each variable, this amounts to finding which pairs of vectors are correlated.

The most basic formalization of this problem is the so-called Light Bulb Problem, introduced by L. Valiant in 1988 [18]:

▶ **Problem 1** (Light Bulb Problem). *We are given as input a set S of n vectors from $\{-1,1\}^d$, which are all independently and uniformly random except for two planted vectors (the correlated pair) which have inner product at least $\rho \cdot d$ for some $0 < \rho \leq 1$. The goal is to find the correlated pair.*

[1] Much of this work was done while the author was working at IBM Research Almaden.

2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy Fineman and Michael Mitzenmacher; Article No. 2; pp. 2:1–2:11

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The dimension d of the vectors is called the *sample complexity* of the problem, since, in our data analysis application, it corresponds to the number of data points which must be gathered about the variables in order to determine which are correlated. When d is too small, then the problem is information-theoretically impossible. For instance, if $d < \log(n-2)$, then by the pigeonhole principle, two of the random vectors must be *equal* to each other, and there is no way to distinguish them from the planted correlated pair we are trying to find. By standard concentration inequalities, there is a constant $c > 1$ such that, whenever $d \geq c \log n$, the correlated pair is the closest pair of vectors with high probability. We would like to design algorithms for this $d = O(\log n)$ regime, so that we can find correlated pairs without increasing the sample complexity above the information-theoretic requirement.

A naïve approach to the Light Bulb Problem is to compute the inner product of each pair, which takes $\Omega(n^2)$ time. However, in many applications, n is quite large, and quadratic time is infeasible. For one example, in genome-wide association studies, scientists have gathered data on millions of genetic markers, and determining which of these are correlated is key to understanding their interactions in different biological mechanisms [13, 19].

It is not hard to see that the Light Bulb Problem is a special case of the $(1+\varepsilon)-$*approximate Hamming nearest neighbor problem*. Using Indyk and Motwani's famous Locality-Sensitive Hashing framework [8], one can solve the Light Bulb Problem in time $n^{2-O(\rho)}$. For constant $\rho > 0$, this gives a truly subquadratic runtime, but the runtime become quadratic as $\rho \to 0$. This is undesirable, as in many data analysis applications, as well as in applications to other areas like learning theory, we would like to quickly detect weak correlations with small ρ. Later work [14, 5] improved the constants in the $O(\rho)$ term, but still had the same asymptotic dependence on ρ in the exponent.

In a breakthrough result, G. Valiant [17] gave an algorithm solving the Light Bulb Problem in time $O(n^{(5-\omega)/(4-\omega)+\varepsilon} + nd) < O(n^{1.615} + nd)$, where $\omega < 2.373$ is the matrix multiplication constant, for *any* constant $\rho > 0$, no matter how small. Thereafter, Karppa et al. [9] gave an improved algorithm with a runtime of $O(n^{2\omega/3+\varepsilon} + nd) < O(n^{1.582} + nd)$. Both of these algorithms work when the sample complexity d matches, up to a constant, the information-theoretically necessary $d = \Theta(\log n)$.

In this paper, we give an algorithm with a simple analysis which matches the best known runtime and sample complexity.

▶ **Theorem 2.** *For every $\varepsilon, \rho > 0$, there is a $\kappa > 1$ such that the Light Bulb Problem for correlation ρ can be solved in randomized time $O(n^{2\omega/3+\varepsilon})$ whenever $d = \kappa \log n$ with polynomially low error.*

By leveraging our simpler analysis, we also give algorithms for several natural extensions and generalizations of the Light Bulb Problem, which are sometimes much faster than the algorithms from previous work.

1.1 Algorithm Overview

Our algorithm combines techniques from past work on subquadratic algorithms for the Light Bulb Problem [17, 9, 10] with techniques for batch nearest neighbor algorithms using the *'polynomial method in algorithm design'* [2, 1]. Our algorithm begins in a way common to both methods: we partition S into $m = n^{2/3}$ groups $S = S_1 \cup \cdots \cup S_m$. Our goal is then to simultaneously check, for each pair (S_i, S_j) of groups, whether there is a correlated pair of vectors in $S_i \times S_j$. We will do this by, for each group S_i, constructing two vectors $A_i, B_i \in \mathbb{R}^t$, for $t \approx n^{2/3}$, such that the inner product $\langle A_i, B_j \rangle$ is large if and only if there is a correlated pair of vectors in $S_i \times S_j$. By using fast matrix multiplication, we can quickly compute all of these inner products and find the correlated pair.

We differ from past work in how the A_i and B_i vectors are constructed. In [17, 9], a sophisticated random sampling technique is used, including an involved probabilistic analysis to keep t low, and in [10], that technique is derandomized. We instead use the polynomial method: we first design a polynomial (see (3) below) which, it is not hard to show, has the desired properties. We then convert it into A_i, B_i vectors by dividing it up into monomials. By designing a polynomial whose degree is not too high, we get that the resulting number of monomials, and hence t, is also not too high. Our proof of correctness is straightforward, and it almost entirely avoids arguments about tail distributions of sums of dependent variables, including a multitude of calculations and casework, which are prevalent in the past work.

That said, our algorithm can be seen as the 'best of both worlds': past work on the polynomial method has focused on designing subquadratic time algorithms, but not on optimizing *how* subquadratic the runtime is. We use ideas from past work on the Light Bulb Problem (our overall approach to the problem comes from [17], and the last paragraph in the proof of Theorem 2 uses a clever trick of [9]) in order to optimize our runtime here.

1.2 Deterministic Light Bulb Problem

In some cases, one would like a deterministic algorithm which is guaranteed to find correlations in subquadratic time. Since our polynomial construction and evaluation process is entirely deterministic, we can get such an algorithm easily. However, as the inputs to the Light Bulb Problem come from a random distribution, we need to be careful about what a deterministic algorithm means in this setting. For instance, there is a small chance that a random pair of vectors will be just as correlated as the correlated pair, in which case a deterministic algorithm has no hope of finding the true correlated pair.

Almost All Instances

One option is to design an algorithm which correctly solves *almost all instances*. This is the notion which was introduced and used in the past work by Karppa et al. [10] on deterministic algorithms for the Light Bulb Problem. We say that an algorithm is correct on almost all instances if the probability of drawing an instance where the algorithm fails is $1/\mathrm{poly}(n)$. For this notion, we match the runtime of the best randomized algorithm:

▶ **Theorem 3.** *For every $\varepsilon, \rho > 0$, there is a $\kappa > 0$ such that the Light Bulb Problem for correlation ρ can be solved in deterministic time $O(n^{2\omega/3+\varepsilon})$ on almost all instances whenever $d = \kappa \log n$.*

Our runtime of $O(n^{2\omega/3+\varepsilon}) \leq O(n^{1.582})$ is faster than the runtime of Karppa et al. [10], which is at best $O(n^{1.996})$. Our algorithm also uses more straightforward and elementary techniques. The original algorithm of Karppa et al. relies heavily on random sampling. In order to derandomize this, Karppa et al. use heavy-duty techniques including constructing 'correlation amplifiers' using the explicit expander graphs of Reingold, Vadhan, and Wigderson [15]. We avoid any such complications, since our algorithm replaces random sampling with a deterministic polynomial construction. In fact, one can view our polynomials in (3), below, as a smaller, elementary construction of their notion of a correlation amplifier.

Our algorithm for Theorem 3 uses two ideas to derandomize the algorithm for Theorem 2 without changing the runtime. First, we use a standard technique in derandomization: by examining the proof of correctness of Theorem 2, we will see that the random bits it uses only need to be *pairwise independent*, rather than fully independent, which means only $O(\mathrm{polylog}\, n)$ independent random bits are needed for the algorithm to succeed with high

probability. Second, in the proof of Theorem 3, we use the fact that, with the exception of the correlated pair of vectors, the vectors in the input set S of the Light Bulb Problem *are* random vectors, and we can use them as the source of random bits we need. This technique of using the input as a source of randomness has been used in a number of past derandomization results; see eg. [7, 20].

Promise that random vectors aren't too correlated

Although the algorithm of [10] is presented as working on almost all instances, it implicitly works in a stronger regime. It solves a promise version of the Light Bulb Problem, which we introduce here, in which we are guaranteed that no pair of random vectors is too correlated:

▶ **Problem 4** (Promise Light Bulb Problem with parameter w)**.** *We are given as input a set S of n vectors from $\{-1,1\}^d$, where two of the vectors (the correlated pair) have inner product at least $\rho \cdot d$ for some $0 < \rho \leq 1$, and every other pair of vectors has inner product at most $w\sqrt{d \log n}$. The goal is to find the correlated pair.*

To emphasize: the inputs to the Promise Light Bulb Problem are not necessarily chosen randomly; they can be chosen adversarially as long as they satisfy the guarantee.

By a Chernoff bound, a random instance of the Light Bulb Problem will also satisfy this guarantee with probability $1 - 1/\text{poly}(n)$, for a sufficiently large constant w. Hence, deterministically solving the Promise Light Bulb Problem is sufficient to solve the Light Bulb Problem on almost all instances, and this is the approach that [10] takes. One benefit of the Promise Light Bulb Problem is that it doesn't let us use the 'artificial' trick of using vectors from a randomly chosen input as the source of randomness. Without using that trick, we can nonetheless solve the Promise Light Bulb Problem deterministically, with running time $O(n^{4\omega/5+\varepsilon}) \leq O(n^{1.8983})$:

▶ **Theorem 5.** *There is a constant $w > 0$ such that, for every $\varepsilon, \rho > 0$, there is a $\kappa > 0$ such that the Promise Light Bulb Problem with parameter w for correlation ρ can be solved in deterministic time $O(n^{4\omega/5+\varepsilon})$ whenever $d = \kappa \log n$.*

While this algorithm is slower than our aforementioned algorithms, it is nonetheless still faster than the previous best deterministic runtime [10] of $O(n^{1.996})$, and it follows without much more work from our deterministic polynomial construction.

1.3 Generality of the Light Bulb Problem

As the Light Bulb Problem is so basic, a number of other important problems can be reduced to it as well. Here we give a couple of examples from prior work.

Correlations on the Euclidean Sphere

In all the above, we have been discussing finding correlated vectors from the domain $\{-1,1\}^d$. What if we are more generally interested in finding correlated vectors from the d-dimensional Euclidean sphere[2]? There is a randomized hashing algorithm by Charikar [3] that 'rounds' the Euclidean sphere to $\{-1,1\}^d$ in such a way that all of our algorithms above will still work, with ρ only decreasing by a constant factor. The hash function simply picks a uniformly random hyperplane through the origin, and outputs 1 or -1 depending on which side of the

[2] The d-dimensional Eucliedan sphere is the set of points $x \in \mathbb{R}^d$ such that $x_1^2 + \cdots + x_d^2 = 1$.

hyperplane a point lies on. Since our runtime for the Light Bulb Problem in Theorem 2 does not change when ρ changes by a constant factor, we can thus achieve the same guarantees for the Light Bulb Problem on the Euclidean sphere.

Learning Sparse Parities with Noise and More

L. Valiant [18] first introduced the Light Bulb Problem as a basic example of a correlated learning problem. More generally, the Light Bulb Problem can be seen as a special case of several different problems in learning theory, including learning sparse parities with noise, learning sparse juntas with or without noise, and learning sparse DNFs. Surprisingly, Feldman et al. [6] showed that all these more general learning problems can be reduced to the Light Bulb Problem as well, and the fastest known algorithms for them come from applying this reduction followed by the best Light Bulb Problem algorithms. Hence, our algorithm gives a new, simpler algorithm matching the best known runtimes for these problems as well. We refer to [17, Appendix A] for a more detailed discussion of these reductions.

2 Preliminaries

We assume familiarity with basic facts about combinatorics and probability, and in particular, the union bound, Chernoff bound, and Chebyshev inequality. For an integer $d \geq 0$, we write $[d] := \{1, 2, \ldots, d\}$. For a vector $x \in \{-1, 1\}^d$, we will write x_i to denote the ith entry of x for any $i \in [d]$, and $x_M := \prod_{i \in M} x_i$ for any $M \subseteq [d]$.

Polynomial Multilinearization

For a multivariate polynomial $p : \mathbb{R}^d \to \mathbb{R}$, its *multilinearization* is the polynomial $\hat{p} : \mathbb{R}^d \to \mathbb{R}$ which one gets when one expands p into a sum of monomials, and then for each monomial, and each variable in that monomial, one reduces the exponent of that variable mod 2 to either 0 or 1. For instance, if $p(x_1, x_2, x_3) = x_1 x_2^5 x_3^2 + 3x_2^2$, then $\hat{p}(x_1, x_2, x_3) = x_1 x_2 + 3$. Notice that $x_i^2 = 1$ whenever $x_i \in \{-1, 1\}$, and so for any p, and any $x \in \{-1, 1\}^d$, we always have that $p(x) = \hat{p}(x)$. The number of multilinear monomials on d variables of degree exactly r is $\binom{d}{r}$. Hence, if p has degree r, then the number of monomials in $\hat{p}$ is at most $\sum_{i=0}^{r} \binom{d}{i}$.

We will use the two bounds on binomial coefficients to bound the number of monomials in $\hat{p}$. First, if $0 \leq k_1 \leq k_2 \leq n/2$, then $\binom{n}{k_1} \leq \binom{n}{k_2}$. Second, for any $1 \leq k \leq n$, Stirling's approximation shows that

$$\binom{n}{k} \leq \frac{n^k}{k!} \leq \left(\frac{e \cdot n}{k}\right)^k. \tag{1}$$

Matrix Multiplication Notation

Let $M(a, b)$ denote the runtime to compute the product of an $a \times b$ matrix with a $b \times a$ matrix, whose entries are integers of magnitude at most $2^{\text{polylog}(ab)}$. For instance, $M(n, n) \leq O(n^\omega)$ where $\omega \leq 2.373$ [21, 11] is the matrix multiplication exponent. Since a $n \times n^{1+\varepsilon} \times n$ matrix multiplication can be decomposed into n^ε different $n \times n \times n$ multiplications, we see that for $\varepsilon \geq 0$,

$$M(n, n^{1+\varepsilon}) \leq O(n^{\omega+\varepsilon}). \tag{2}$$

3 Algorithm for the Light Bulb Problem

In this section, we give our algorithm for the Light Bulb Problem, proving our main result, Theorem 2.

▶ **Theorem 2** (Restated). *For every $\varepsilon, \rho > 0$, there is a $\kappa > 0$ such that the Light Bulb Problem for correlation ρ can be solved in randomized time $O(n^{2\omega/3+\varepsilon})$ whenever $d = \kappa \log n$ with polynomially low error.*

For two constants $\gamma, k > 0$ to be determined, we will pick $\kappa = \gamma k^2/\rho^2$. Let $S \subseteq \{-1, 1\}^d$ be the set of input vectors, and let $x', y' \in S$ denote the correlated pair which we are trying to find. For distinct $x, y \in S$ other than the correlated pair, the inner product $\langle x, y\rangle$ is a sum of d uniform independent $\{-1, 1\}$ values. Let $v := \gamma(k/\delta) \log n$. By a Chernoff bound, for large enough γ, we have $|\langle x, y\rangle| \leq v$ with probability at least $1 - 1/n^3$. Hence, by a union bound over all pairs of uncorrelated vectors, we have $|\langle x, y\rangle| \leq v$ for all such x, y with probability at least $1 - 1/n$. We assume henceforth that this is the case. Meanwhile, $\langle x', y'\rangle \geq \rho d = kv$.

Arbitrarily partition S into $m := n^{2/3}$ groups $S_1, \ldots, S_m$ of size $g := n/m = n^{1/3}$ each. We can compute the inner product between each pair of vectors which was assigned to the same group in time $O(m \cdot g^2 \cdot d) = \tilde{O}(n^{4/3})$, and if we find the correlated pair, we can return it and end the algorithm. Otherwise, we may assume the correlated vectors are in different groups, and we continue.

For each $x \in S$, our algorithm picks a value $a^x \in \{-1, 1\}$ independently and uniformly at random. For a constant $\tau > 0$ to be determined, let $r = \lceil \log_k(\tau n^{1/3}) \rceil$, and define the polynomial $p : \mathbb{R}^d \to \mathbb{R}$ by $p(z_1, \ldots, z_d) = (z_1 + \cdots + z_d)^r$. Our goal is, for each $(i, j) \in [m]^2$, to compute the value

$$C_{i,j} := \sum_{x \in S_i} \sum_{y \in S_j} a^x \cdot a^y \cdot p(x_1 y_1, \ldots, x_d y_d).$$

Solving the problem using $C_{i,j}$

Let us first explain why we are interested in computing $C_{i,j}$. Denote $p(x, y) := p(x_1 y_1, \ldots, x_d y_d)$. Intuitively, $p(x, y)$ is computing an *amplification* of $\langle x, y\rangle$. $C_{i,j}$ is then summing these amplified inner products for all pairs $(x, y) \in S_i \times S_j$. We will pick our parameters so that the amplified inner product of the correlated pair is large enough to stand out from the sums of inner products of random pairs.

Let us be more precise. Recall that for uncorrelated x, y we have $|\langle x, y\rangle| \leq v$, and hence $|p(x, y)| \leq v^r$. Similarly, we have $|p(x', y')| \geq (kv)^r \geq \tau n^{1/3} v^r$. For $x, y \in S$, define $a^{(x,y)} := a^x \cdot a^y$. Notice that, for $i \neq j$, $C_{i,j} = \sum_{x \in S_i, y \in S_j} a^{(x,y)} p(\langle x, y\rangle)$, where the $a^{(x,y)}$ are *pairwise independent* random $\{-1, 1\}$ values.

We will now analyze the random variable $C_{i,j}$ where we think of the vectors in S as fixed, and only the values a^x as random.

Consider first when the correlated pair are not in S_i and S_j. Then, $C_{i,j}$ has mean 0, and (since variance is additive for pairwise independent variables) $C_{i,j}$ has variance at most $|S_i| \cdot |S_j| \cdot \max_{x \in S_i, y \in S_j} |p(\langle x, y\rangle)|^2 \leq n^{2/3} \cdot v^{2r}$. For sufficiently large constant τ, by the Chebyshev inequality, we have that $|C_{i,j}| \leq \tau n^{1/3} v^r/3$ with probability at least $3/4$. Let $\theta = \tau n^{1/3} v^r/3$, so $|C_{i,j}| \leq \theta$ with probability at least $3/4$.

Meanwhile, if $x' \in S_i$ and $y' \in S_j$, then $C_{i,j}$ is the sum of $a^{(x',y')} p(\langle x', y'\rangle)$ and a variable C' distributed as $C_{i,j}$ was in the previous paragraph. Hence, since $|p(\langle x', y'\rangle)| \geq \tau n^{1/3} v^r = 3\theta$, and $|C'| \leq \theta$ with probability at least $3/4$, we get by the triangle inequality that $|C_{i,j}| \geq 2\theta$ with probability at least $3/4$.

Hence, if we repeat the process of selecting the a^x values for each $x \in S$ independently at random $O(\log n)$ times, whichever pair S_i, S_j has $|C_{i,j}| \geq 2\theta$ most frequently will be the pair containing the correlated pair with polynomially low error, and then a brute force within this set of $O(n^{1/3})$ vectors can find the correlated pair in $\tilde{O}(n^{2/3})$ time. In all, by a union bound over all possible errors, this will succeed with polynomially low error.

Computing $C_{i,j}$

It remains to give the algorithm to compute $C_{i,j}$. Before doing this, we will rearrange the expression for $C_{i,j}$ into one which is easier to compute. Since we are only interested in the values of p when its inputs are all in $\{-1, 1\}$, we can replace p with its multilinearization $\hat{p}$. Let $M_1, \ldots, M_t$ be an enumeration of all subsets of $[d]$ of size at most r, so $t = \sum_{i=0}^{r} \binom{d}{i}$. Then, there are coefficients $c_1, \ldots, c_t \in \mathbb{Z}$ such that $\hat{p}(x) = \sum_{s=1}^{t} c_s x_{M_s}$. Rearranging the order of summation, we see that we are trying to compute

$$\begin{aligned} C_{i,j} &= \sum_{s=1}^{t} \sum_{x \in S_i} \sum_{y \in S_j} a^x \cdot a^y \cdot c_s \cdot x_{M_s} \cdot y_{M_s} \\ &= \sum_{s=1}^{t} \left[c_s \cdot \left(\sum_{x \in S_i} a^x \cdot x_{M_s} \right) \cdot \left(\sum_{y \in S_j} a^y \cdot y_{M_s} \right) \right]. \end{aligned} \tag{3}$$

In order to compute $C_{i,j}$, we first need to compute the coefficients c_s. Notice that c_s depends only on $|M_s|$ and r. We can thus derive a simple combinatorial expression for c_s, and hence compute all of the c_s coefficients in $\text{poly}(r) = \text{polylog}(n)$ time. Alternatively, by starting with the polynomial $(z_1 + \cdots + z_d)$ and then repeatedly squaring then multilinearizing, we can easily compute all the coefficients in $O(t^2 \, \text{polylog}(n))$ time; this slower approach is still fast enough for our purposes.

Define the matrices $A, B \in \mathbb{Z}^{m \times t}$ by $A_{i,s} = \sum_{x \in S_i} a^x \cdot x_{M_s}$ and $B_{i,s} = c_s \cdot A_{i,s}$. Notice from (3) that the matrix product $C := AB^T$ is exactly the matrix of the values $C_{i,j}$ we desire. A simple calculation (see Lemma 6 below) shows that for any $\varepsilon > 0$, we can pick a sufficiently big constant $k > 0$ such that $t = O(n^{2/3+\varepsilon})$. Since $m = O(n^{2/3})$, if we have the matrices A, B, then we can compute this matrix product in $M(n^{2/3}, n^{2/3+\varepsilon}) = O(n^{2\omega/3+\varepsilon})$ time, completing the algorithm.

Unfortunately, computing the entries of A and B naively would take $\Omega(m \cdot t \cdot g) = \Omega(n^{5/3})$ time, which is slower than we would like. We will instead use a clever trick due to Lovett [12], which was first applied in this context by Karppa et al. [9]: we will compute those entries using *another* matrix multiplication. Let $N_1, \ldots, N_u$ be an enumeration of all subsets of $[d]$ of size at most $\lceil r/2 \rceil$. For each $i \in [m]$, define the matrices $L^i, \tilde{L}^i \in \mathbb{Z}^{u \times g}$ (whose columns are indexed by elements $x \in S_i$) by $L^i_{s,x} = x_{N_s}$ and $\tilde{L}^i{}_{s,x} = a^x \cdot x_{N_s}$. Then, compute the product $P^i := L^i \tilde{L}^{i\,T}$. We can see that $P^i_{s,s'} = \sum_{x \in S_i} a^x \cdot x_{N_s \oplus N_{s'}}$, where $N_s \oplus N_{s'}$ is the symmetric difference of N_s and $N_{s'}$. Since any set of size at most r can be written as the symmetric difference of two sets of size at most $\lceil r/2 \rceil$, each desired entry $A_{i,s}$ can be found as an entry of the computed matrix P^i. Similar to our bound on t from before (see Lemma 6 below), we see that for big enough constant k, we have $u = O(n^{1/3+\varepsilon})$. Computing the entries of the L^i matrices naively takes only $O(m \cdot u \cdot g \cdot r) = \tilde{O}(n \cdot u) = \tilde{O}(n^{4/3+\varepsilon})$ time, and then computing the products P^i takes $O(m \cdot \max(u, g)^\omega) = O(n^{(2+\omega)/3+\varepsilon})$ time; both of these are dominated by $O(n^{2\omega/3+\varepsilon})$. This completes the algorithm! Finally, we perform the computations mentioned above:

▶ **Lemma 6.** *For every $\varepsilon > 0$, there is a $k > 0$ such that (with the same notation as in the proof of Theorem 2 above) we can bound $t = O(n^{2/3+\varepsilon})$, and $u = O(n^{1/3+\varepsilon})$.*

Proof. Recall that $d = O(k^2 \log(n))$, and $r = \log_k(O(n^{1/3}))$. Hence, by the bound (1),

$$t \leq (r+1) \cdot \binom{d}{r} \leq (r+1) \cdot (ed/r)^r \leq O(k^2 \log(k))^{\log_k(O(n^{1/3}))} = n^{2/3+O(\log\log(k)/\log(k))}.$$

For any $\varepsilon > 0$ we can thus pick a sufficiently large k so that $t \leq O(n^{2/3+\varepsilon})$. We can similarly bound $\binom{d}{r/2} \leq O(n^{1/3+\varepsilon})$ which implies our desired bound on u. ◀

4 Deterministic Algorithms

We now present our two deterministic algorithms for the Light Bulb Problem. Each is a slight variation on the algorithm from the previous section.

▶ **Theorem 3** (Restated). *For every $\varepsilon, \rho > 0$, there is a $\kappa > 0$ such that the Light Bulb Problem for correlation ρ can be solved in deterministic time $O(n^{2\omega/3+\varepsilon})$ on almost all instances whenever $d = \kappa \log n$.*

Proof. The only randomness used by our algorithm for Theorem 2 was our choice of an independently and uniformly random $a^x \in \{-1, 1\}$ for each $x \in S$. Since this requires $\Theta(n)$ random bits, and we repeat the entire algorithm $\Theta(\log n)$ times to get our desired correctness guarantee, the total number of random bits used is $\Theta(n \log n)$.

However, the only property of the a^x variables which we use in the proof of correctness is that they are *pairwise*-independent. By standard constructions[3], only $O(\log n)$ independent random bits are needed to generate n pairwise-independent random bits. Thus, our entire algorithm actually only needs $O(\log^2 n)$ independent random bits.

Our entirely deterministic algorithm then proceeds as follows. Pick the same κ as in Theorem 2. Let $S \subseteq \{-1, 1\}^d$ be the input vectors. Arbitrarily pick a subset $S' \subseteq S$ of $|S'| = \Theta(\log n)$ of the input vectors, and let $R = S \setminus S'$ be the remaining vectors.

We begin by testing via brute-force whether either vector of the correlated pair is in S'. This can be done in $O(|S'| \cdot |S| \cdot d) = O(n \log^2(n))$ time. If we find the correlated pair (a pair with inner product at least $\rho \cdot d$), then we output it, and otherwise, we can assume that the vectors in S' are all uniformly random vectors from $\{-1, 1\}^d$. In other words, we can use them as $d \cdot |S'| = \Theta(\log^2 n)$ independent uniformly random bits. We thus use them as the required randomness to run the algorithm from Theorem 2 on input vectors R. That algorithm has polynomially low error, which implies the desired correctness guarantee. ◀

▶ **Theorem 5** (Restated). *There is a constant $w > 0$ such that, for every $\varepsilon, \rho > 0$, there is a $\kappa > 0$ such that the Promise Light Bulb Problem with parameter w for correlation ρ can be solved in deterministic time $O(n^{4\omega/5+\varepsilon})$ whenever $d = \kappa \log n$.*

Proof. The guarantee of the Promise Light Bulb Problem is that, when we pick a sufficiently large w, the uncorrelated vectors have as small inner product as we assumed they did in the first paragraph in the proof of Theorem 2. In other words, there is a quantity v such that $|\langle x, y\rangle| \leq v$ for all $x, y \in S$ other than the correlated pair, and moreover, $\langle x', y'\rangle \geq kv$ for a constant $k > 0$ with $k \to \infty$ as $w \to \infty$.

[3] For one example, to generate $2^\ell - 1$ pairwise-independent bits, pick only ℓ bits $b_1, \ldots, b_\ell \in \{-1, 1\}$ independently and uniformly at random, and then output, for each $I \subseteq [\ell]$, the product $\prod_{i \in I} b_i$.

The algorithm is then almost identical to Theorem 2, except we need to remove the only use of randomness: the randomness used to pick the a^x values. To do this, we will simply pick $a^x = 1$ for all x.

In order to guarantee the correctness of our algorithm, we must now change the parameters slightly. Instead of partitioning the input into $m = n^{2/3}$ groups of size $g = n^{1/3}$, we will instead partition into $m = n^{4/5}$ groups of size $g = n^{1/5}$. Similarly, instead of picking r (the exponent in the polynomial p) to be $\log_k(O(n^{1/3}))$, we will pick $r = \log_k(3n^{2/5})$, so that $p(x', y') \geq (kv)^r = 3n^{2/5}v^r$.

With these choices, for any i and j such that the correlated pair are not in S_i and S_j, we have $|C_{i,j}| \leq |S_i| \cdot |S_j| \cdot n^{2/5} = n^{2/5}v$, whereas if $x' \in S_i$ and $y' \in S_j$ then by the triangle inequality, $|C_{i,j}| \geq p(x', y') - |S_i| \cdot |S_j| \cdot n^{2/5} \geq 2n^{2/5}v^r$. Hence, the correlated pair must be in whichever S_i and S_j with $i \neq j$ has the largest $|C_{i,j}|$.

The algorithm to compute the $C_{i,j}$ values is identical to that of Theorem 2. We now get that $t = \sum_{i=0}^{r} \binom{d}{i} \leq O(n^{4/5+\varepsilon})$ and similarly, $u \leq O(n^{2/5+\varepsilon})$, which leads to a final runtime of $O(n^{4\omega/5+\varepsilon})$, as desired. ◀

5 Conclusion

Faster Algorithms?

In this paper, we give an algorithm for the Light Bulb Problem and some variants. A natural question remains: can one improve the $O(n^{2\omega/3})$ runtime? It seems like substantially new techniques might be necessary. We currently reduce the problem to a $n^{2/3} \times n^{2/3} \times n^{2/3}$ matrix multiplication; with a further reduction in the dimensions, even using the cubic matrix multiplication algorithm would give a subquadratic algorithm for the Light Bulb Problem. This would be surprising, since recent progress on the problem has relied heavily on fast matrix multiplication.

It should nonetheless be noted that, despite using fast matrix multiplication, the algorithms in this paper can be quite practical. For instance, using Strassen's original algorithm [16], which is frequently used in practice, gives $\omega \approx 2.81$, and hence a subquadratic runtime for the Light Bulb Problem of about $O(n^{2\omega/3}) \leq O(n^{1.874})$.

Finding General Correlations

Past work on the Light Bulb Problem has also approached a more general problem of finding correlations:

▶ **Problem 7** (Finding Correlations). *We are given as input two sets $X, Y \subseteq \{-1, 1\}^d$ of n vectors each, with the promise that for at most q pairs of $x, y \in X \times Y$, we have $|\langle x, y\rangle| \geq \rho d$ (x and y are* correlated*), and for all other pairs of $x, y \in X \times Y$, we have $|\langle x, y\rangle| \leq \tau d$ (x and y are* uncorrelated*) for some constants $0 < \tau < \rho \leq 1$. Our goal is to find all q of the correlated pairs of vectors.*

Again, as $\rho \to 0$, hashing techniques give runtimes which approach quadratic. However, if τ is also comparatively small (say, there is a constant $\sigma > 1$ such that $\rho/\tau \geq \sigma$), we might hope to achieve a truly subquadratic runtime, no matter how small ρ becomes. With only a slight modification of our algorithm for Theorem 5, we can achieve this:

▶ **Proposition 8.** *For all constants $\eta, c > 0$, and $\sigma > 1$, there exists a constant $\varepsilon > 0$ such that Finding Correlations can be solved in $O(n^{2-\varepsilon})$ deterministic time when $\rho/\tau \geq \sigma$, $q \leq n^{2-\eta}$ and $d = c\log(n)$.*

Proposition 8 is somewhat weaker than the results from past work [17, 10, 9], which only require that $\log(1/\tau)/\log(1/\rho)$ be bounded below by a constant. However, our algorithm benefits from the same simplicity as our Light Bulb Problem algorithms, and it is also deterministic (in the usual sense – there is no distribution on inputs the Finding Correlations problem). Like before, only [10] gives a deterministic algorithm, and it involves the same aforementioned heavy-duty techniques which we avoid. We omit the details of this algorithm here for clarity of exposition, as the algorithm is almost identical to that of Theorem 5. We also note that Chen [4, Lemma 3.2] recently gave an algorithm similar to Proposition 8 for the very related 'approximate maximum inner product' problem, which also made use of the polynomial $p(z_1, \ldots, z_d) = (z_1 + \cdots + z_d)^r$.

References

1 Josh Alman, Timothy M Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *FOCS*, 2016.

2 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *FOCS*, 2015.

3 Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

4 Lijie Chen. On The Hardness of Approximate and Exact (Bichromatic) Maximum Inner Product. In *CCC*, 2018.

5 Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, 2010.

6 Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. On agnostic learning of parities, monomials, and halfspaces. *SIAM Journal on Computing*, 39(2):606–645, 2009.

7 Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. *Lecture notes in computer science*, 2483:209–223, 2002.

8 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

9 Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. In *SODA*, 2016.

10 Matti Karppa, Petteri Kaski, Jukka Kohonen, and Padraig Ó Catháin. Explicit Correlation Amplifiers for Finding Outlier Correlations in Deterministic Subquadratic Time. In *ESA*, 2016.

11 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, 2014.

12 Shachar Lovett. Computing Polynomials with Few Multiplications. *Theory of Computing*, 7(1):185–188, 2011.

13 Jonathan Marchini, Peter Donnelly, and Lon R Cardon. Genome-wide strategies for detecting multiple loci that influence complex diseases. *Nature genetics*, 37(4):413, 2005.

14 Ramamohan Paturi, Sanguthevar Rajasekaran, and John Reif. The light bulb problem. *Information and Computation*, 117(2):187–192, 1995.

15 Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of mathematics*, pages 157–187, 2002.

16 Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

17 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *Journal of the ACM*, 62(2):13, 2015.

18 Leslie G Valiant. Functionality in Neural Nets. In *AAAI*, 1988.

19 Xiang Wan, Can Yang, Qiang Yang, Hong Xue, Nelson LS Tang, and Weichuan Yu. Detecting two-locus associations allowing for interactions in genome-wide association studies. *Bioinformatics*, 26(20):2517–2525, 2010.

20 R Ryan Williams. Natural proofs versus derandomization. *SIAM Journal on Computing*, 45(2):497–529, 2016.

21 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, 2012.

Simple Concurrent Labeling Algorithms for Connected Components

Sixue Liu
Princeton University
sixuel@cs.princeton.edu

Robert E. Tarjan
Princeton University and Intertrust Technologies
ret@cs.princeton.edu

Abstract

We present new concurrent labeling algorithms for finding connected components, and we study their theoretical efficiency. Even though many such algorithms have been proposed and many experiments with them have been done, our algorithms are simpler. We obtain an $O(\lg n)$ step bound for two of our algorithms using a novel multi-round analysis. We conjecture that our other algorithms also take $O(\lg n)$ steps but are only able to prove an $O(\lg^2 n)$ bound. We also point out some gaps in previous analyses of similar algorithms. Our results show that even a basic problem like connected components still has secrets to reveal.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Connected Components, Concurrent Algorithms

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.3

Funding Research at Princeton University partially supported by an innovation research grant from Princeton and a gift from Microsoft.

Acknowledgements We thank Dipen Rughwani, Kostas Tsioutsiouliklis, and Yunhong Zhou for telling us about [18], for extensive discussions about the problem and our algorithms, and for insightful comments on our results.

1 Introduction

The problem of finding the connected components of an undirected graph with n vertices and m edges is fundamental in algorithmic graph theory. Any kind of graph search, such as depth-first or breadth-first search, solves it in linear time sequentially, which is best possible. The problem becomes more interesting in a concurrent model of computation. In the heyday of the theoretical study of PRAM algorithms, many more-and-more efficient algorithms for the problem were discovered, culminating in 2001 with the $O(\lg n)$-time, $O((m+n)/\lg n)$-processor randomized EREW PRAM algorithm of Halperin and Zwick [8], where lg is the base-two logarithm. Most of this work was motivated by obtaining the best bounds, not the simplest algorithms.

With the growth of the internet, the world-wide web, and cloud computing, computing connected components on huge graphs has become commercially important, and practitioners have put versions of the PRAM algorithms into use. Many of these algorithms are quite complicated, and even some of the simple ones have been further simplified when implemented. There is evidence that such simple algorithms perform well in practice, but claims about their theoretical performance are unsound.

Given this situation, our goal here is to develop and analyze the simplest possible efficient algorithms for the problem and to rigorously analyze their efficiency. In exchange for algorithmic simplicity, we are willing to allow analytic complexity. Our computational framework is based on the MPC (Massive Parallel Computing) model [3], which is a variant of the BSP (Bulk Synchronous Parallel) model [20]. The MPC model is more powerful than PRAM models in that it allows programmable resolution of write conflicts, but it is a realistic model of cloud computing platforms. Our algorithms also work on a CRCW (concurrent read, concurrent write) PRAM in which write conflicts are resolved in favor of the smallest value written. This model is stronger than the more standard ARBITRARY CRCW PRAM, in which write conflicts are resolved arbitrarily (one of the writes succeeds, but the algorithm has no control over which), but is weaker than the MPC model.

We develop several algorithms that are all simpler than existing algorithms. We conjecture that all of them take $O(\lg n)$ concurrent steps and $O(m \lg n)$ messages. We prove this bound for two of them using a novel multi-round analysis. For the others we obtain an $O(\lg^2 n)$ step bound and $O(m \lg^2 n)$ message bound.

Our paper contains five sections, in addition to this introduction, §2 presents our algorithmic framework, §3 presents our algorithms, §4 discusses related work, §5 completely analyzes one of our algorithms and partially analyzes the others, and §6 closes with some remarks and open problems.

2 Algorithmic Framework

Given an undirected graph with vertex set $[n] = \{1, 2, \ldots, n\}$ and m edges, we wish to compute its connected components via a concurrent algorithm. More precisely, for each component we want to label all its vertices with a unique vertex in the component, so that two vertices are in the same component if and only if they have the same label. To state bounds simply, we assume $n > 1$ and $m > 0$. We denote the ends of an edge e by $e.v$ and $e.w$.

We use the MPC (Massive Parallel Computing) model [3], specialized to our problem. (We discuss the MPC model further in §4.) We consider algorithms that operate in synchronous concurrent steps, each of which can send messages between the edges and vertices. Each edge and each vertex has a local memory. In a concurrent step, each edge and vertex can update its local memory based on the messages sent to it in the previous step, and then send a message to each vertex and edge that it knows about. Initially a vertex knows only about itself, and an edge knows only about its two ends. Thus in the first concurrent step only edges can send messages, and only to their ends. A vertex or edge *knows about* another vertex or edge once it has received a message containing the vertex or edge. We ignore contention resulting from many messages being sent to the same vertex or edge during a step, and the need for a vertex or edge to send many messages during a step. We measure the efficiency of an algorithm primarily by the number of concurrent steps and secondarily by the number of messages sent.

Even in this unrealistically strong model, there is a non-constant lower bound on the number of steps needed to compute connected components:

▶ **Theorem 1.** *Computing connected components takes $\Omega(\lg d)$ steps, where d is the maximum of the diameters of the components.*

Proof. Let u be the vertex that eventually becomes the label of all vertices in the component. Some vertex v is at distance at least $d/2$ from u; otherwise, the diameter of the component is less than d, a contradiction. An induction on the number of steps shows that after k steps

a vertex has only received messages containing vertices within distance 2^k. Since v must receive a message containing u, the computation takes at least $\lg d - 1$ steps. If $d = 1$, the computation takes at least one step. ◀

If messages can be arbitrarily large, it is easy to solve the problem in $O(\lg d)$ steps, by first sending each edge end to the other end, and then repeatedly sending from each vertex the entire set of vertices it knows about to all these vertices [15]. If there is a large component, however, such an algorithm is not practical, for at least two reasons: it requires huge memory at each vertex, and the last step can send a number of messages quadratic in n. Hence we restrict the local memory of a vertex or edge to hold only a small constant number of vertices and edges. We also restrict messages to hold only a small constant number of vertices and edges, along with an indication of the message type, such as a label request or a label update.

All our algorithms maintain a label for each vertex u, initially u itself. Labels are updated step-by-step until none changes, after which all vertices in a component have the same label, which is one of the vertices in the component. At any given time the current labels define a univalent digraph (directed graph) whose arcs lead from vertices to their labels. We call this the *label digraph*. If these arcs form no cycles other than loops (arcs of the form (u, u)), then this graph is a forest of trees rooted at the self-labeled vertices: the parent of u is its label unless the label of u is u, in which case u is a root. We call this the *label forest*. Each tree in the forest is a *label tree*.

All our algorithms maintain the label digraph as a forest; that is, they maintain acyclicity except for self-labels. (We know of one previous algorithm that does not maintain the label digraph as a forest: see §4.) Henceforth we call the label of a vertex u its *parent* and denote it by $u.p$, and we call a label tree just a *tree*. If v is a vertex such that $v.p \neq v$, v is a *child* of $v.p$. A tree is *flat* if the parent of every child is the root. (Some authors call such a tree a *star*.) A vertex is a *leaf* if it is not a root and it has no children. We call a one-vertex tree a *singleton*.

During the computation, all the vertices in a tree are in a single connected component, but there may be more than one tree per component. At the end of the computation, there is one flat tree per component, whose root is the component label.

3 Algorithms

A simple way to guarantee acyclicity is to maintain the parent of a vertex to be the minimum of the vertices it knows about. We call this *minimum labeling*. (An equivalent alternative is to maintain the label of a vertex to be the maximum of the vertices it knows about.) All our algorithms use minimum labeling. Each algorithm proceeds in rounds, each of which updates parents using vertices received from edges, reduces tree depths using *shortcutting*, and possibly alters edges.

The most obvious way to use edges to update parents is for each edge to send its minimum end to its maximum end:

CONNECT: *for each edge* e, *send* $\min\{e.v, e.w\}$ *to* $\max\{e.v, e.w\}$.

This method does not give a correct algorithm unless we combine it with some form of edge alteration or edge addition. In the absence of changes to the edge set, we use one of the following methods instead of CONNECT:

PARENT-CONNECT: *for each edge* e, *request* $e.v.p$ *from* $e.v$ *and* $e.w.p$ *from* $e.w$; *send the minimum of the received vertices to the maximum of the received vertices.*

EXTENDED-CONNECT: *for each edge* e, *request* $e.v.p$ *from* $e.v$ *and* $e.w.p$ *from* $e.w$; *let the received values be* x *and* y, *respectively*; *if* $y < x$ *then send* y *to* v *and to* x *else send* x *to* w *and to* y.

The straightforward way to update labels using messages sent from edges is to replace each parent by the minimum of the received vertices:

UPDATE: *for each vertex v, replace $v.p$ by the minimum of $v.p$ and vertices received from edges.*

Such updating can move subtrees between trees in the forest, producing behavior that is hard to analyze. A more conservative alternative is to update only the parents of roots:

ROOT-UPDATE: *for each root v, replace $v.p$ by the minimum of $v.p$ and vertices received from edges.*

Shortcutting is the key to obtaining a logarithmic step bound. Shortcutting replaces the parent of each vertex by its grandparent:

SHORTCUT: *for each vertex v request $v.p.p$ from $v.p$; replace $v.p$ by the received vertex.*

Edge alteration replaces each edge end by its parent:

ALTER: *for each edge e, request $e.v.p$ from $e.v$ and $e.w.p$ from $e.w$; let the returned vertices be x and y, respectively; if $x = y$ then delete e else replace $e.v$ and $e.w$ by x and y.*

Choosing one of the three connection methods, one of the two update methods, and whether or not to alter edges produces one of twelve algorithms. Not all these algorithms are correct: if we use CONNECT, we must choose ALTER to alter edges; if we do not, the partition of vertices defined by the trees does not change after the first connect. Furthermore not all the correct algorithms are as simple as possible: if we use ALTER, using CONNECT gives a correct algorithm, and we see no advantage in using PARENT-CONNECT or EXTENDED-CONNECT, which are more complicated. (We have no theoretical or practical evidence to justify this intuition, however.) Finally, if we use ROOT-UPDATE, PARENT-CONNECT and EXTENDED-CONNECT are equivalent. This leaves us with five algorithms:

- Algorithm P: *repeat* {PARENT-CONNECT; UPDATE; SHORTCUT} *until no parent changes.*
- Algorithm E: *repeat* {EXTENDED-CONNECT; UPDATE; SHORTCUT} *until no parent changes.*
- Algorithm A: *repeat* {CONNECT; UPDATE; SHORTCUT; ALTER} *until no parent changes.*
- Algorithm R: *repeat* {PARENT-CONNECT; ROOT-UPDATE; SHORTCUT} *until no parent changes.*
- Algorithm RA: *repeat* {CONNECT; ROOT-UPDATE; SHORTCUT; ALTER} *until no parent changes.*

We call an iteration of the repeat loop in each of these algorithms a *round*. Two of these algorithms, R and RA, are equivalent:

▶ **Theorem 2.** *Algorithms R and RA do the same parent updates in each step.*

Proof. Consider running algorithms R and RA concurrently. We prove by induction on the number of steps that (i) parents are the same in both algorithms and (ii) if edge e has original ends $e.v = v$ and $e.w = w$ and has ends $e.v = x$ and $e.w = y$ in RA, then $v.p = x$ and $w.p = y$ in both R and RA. Both (i) and (ii) hold initially. Suppose they hold at the beginning of a round. Then the CONNECT and ROOT-UPDATE in RA do the same parent changes as the PARENT-CONNECT and ROOT-UPDATE in R, so (i) holds after these steps. The SHORTCUT steps in R and RA also do the same parent changes, so (i) holds after this step. The ALTER at the end of RA re-establishes (ii). ◀

Henceforth we treat R and RA as different implementations of the same algorithm. Algorithms A and P are not equivalent in the same sense, as the example in Figure 1 shows.

To prove correctness, we need the following key result:

▶ **Lemma 3.** *All our algorithms maintain the invariant that any two tree roots in the same component are connected by a path of current edges.*

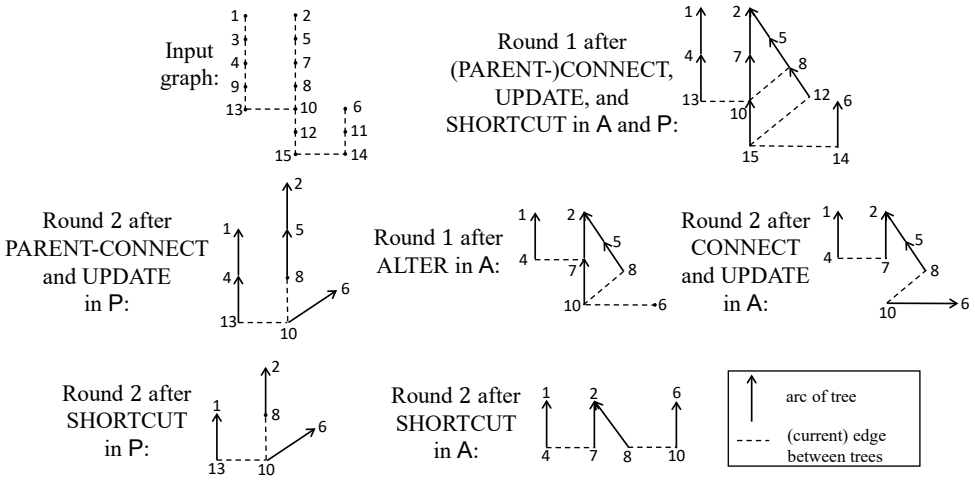


Figure 1 A graph on which A and P do different parent updates. Only necessary vertices, edges, and tree arcs are shown. In round 1 before the ALTER, all arcs and edges are the same in both algorithms. In round 2, the parent of 10 becomes 6 in both algorithms. In round 3 (not shown), P changes the parent of 6 to 1 using the original edge $(10, 13)$, but A changes the parent of 6 to 2 using the new edge $(2, 6)$ altered from $(8, 10)$.

Proof. This is a tautology for P, E, and R. We prove it for A by induction on the number of steps. (The same proof works for RA.) A SHORTCUT changes no edges nor roots; a CONNECT followed by an UPDATE changes no edges and only converts roots to non-roots. Thus both preserve the invariant, since both preserve every path of current edges, and vertices that are roots after an UPDATE are roots before the preceding CONNECT. Given a path between two roots, an ALTER preserves each root it contains, including the ends of the path, and replaces each edge end that is a child by its parent. Thus an ALTER also preserves the invariant. ◀

▶ **Theorem 4.** *All our algorithms are correct.*

Proof. For each algorithm, an induction on the number of steps proves that the vertex sets of the trees are subsets of the vertex sets of the connected components. We prove that each algorithm cannot stop until there is one flat tree per component, which implies correctness. Consider a component whose vertices are in two or more trees just before a round. By Lemma 3, some current edge connects a vertex in one of these trees with a vertex in another tree. If one of these trees is not flat, the SHORTCUT during the round will change some parent if no parents change before the SHORTCUT. If both are flat, some parent will change before the SHORTCUT. ◀

In §5 we prove the following step bounds: $O(d)$ for A and E; $O(\lg^2 n)$ for A, P, and E; and $O(\lg n)$ for R and RA. We conjecture that P, A, and E in fact have an $O(\lg n)$ step bound. We leave a tighter analysis of these algorithms as an open problem. We have included algorithm E because it is similar to an existing algorithm (see §4), and we can prove a step bound that depends only on the diameter.

We conclude this section with five observations.

First, our algorithms need very little memory: one cell per vertex to hold its parent, and two cells per edge to hold its current ends.

Second, although we have used a message-passing framework, it is straightforward to implement our algorithms on a CRCW PRAM in which write conflicts are resolved in favor

of the smallest value written. We say more about the effect of write conflict resolution on connected components algorithms in §4.

Third, in algorithm RA, the condition for edge deletion can be strengthened, since when an edge causes a parental root update it connects a child with its parent, and it can never again cause a parent update. Thus the condition for deletion in the ALTER can be strengthened to "*if* $x = y$ *or* $e.v = y$ *or* $e.w = x$ *then delete* e". This optimization does *not* work in algorithm A.

Fourth, algorithms like R that use root updating are *monotone* in that the vertex set of each new tree in the label forest is a union of vertex sets of old trees. This is *not* true of algorithms such as A, P, and E, which *can* and in general *do* move subtrees between trees. Monotonicity seems to make analysis simpler. Many but not all previous algorithms are monotone.

Fifth, each of our algorithms does one SHORTCUT per round, but we could increase this to two or indeed to any number. Our $O(\lg^2 n)$ bound for P, E, and A extends to the variants of these algorithms that do any fixed positive number of SHORTCUT steps each round. Similarly, our $O(\lg n)$ bound for R and RA extends to the variants of these algorithms that do any fixed positive number of SHORTCUT steps each round. On the other hand, if we modify R (or RA) to do enough SHORTCUT steps each round to flatten all the trees, the worst-case number of steps becomes $\Theta(\lg^2 n)$, as we now show.

Algorithm S: *repeat* {PARENT-CONNECT; ROOT-UPDATE;
repeat SHORTCUT *until no parent changes*} *until no parent changes.*
Algorithm SA: *repeat* {CONNECT; ROOT-UPDATE;
repeat SHORTCUT *until no parent changes*; ALTER} *until no parent changes.*

In both of these algorithms, all trees are flat just before a CONNECT, which implies that CONNECT only changes the parents of roots. A proof like that of Theorem 2 shows that S and SA are equivalent in that they make the same parent changes. We thus treat them as alternative implementations of the same algorithm.

▶ **Theorem 5.** *Algorithm S (and SA) takes* $O(\lg^2 n)$ *steps.*

Proof. The inner repeat loop stops after at most $\lceil \lg n \rceil$ steps with all trees flat. We claim that if there are two or more trees, two rounds of the outer loop at least halve their number, from which the theorem follows. Call a root *minimal* if its tree has edges connecting it only with trees having greater roots. A CONNECT makes every non-minimal root into a non-root. Suppose there are k roots just before a CONNECT, of which j are minimal. If $j < k/2$, the CONNECT reduces the number of roots to at most $j < k/2$. Divide the minima into the i that get a new child as a result of the CONNECT and the $j - i$ that do not. If x is a minimal that does not get a new child, there must be a root $y > x$ such that the trees rooted at x and y are connected by an edge, and y has parent less than x after the CONNECT. The next CONNECT will make x a child. That is, the current CONNECT makes at least i roots into non-roots, and the next CONNECT makes at least $j - i$ into non-roots, for a total of at least $j \geq k/2$, giving the claim in this case as well. ◀

We show by example that the bound in Theorem 5 is tight. For convenience we consider algorithm SA. During the execution, at the beginning of each round it suffices to consider only the induced subgraph of vertices that are roots. This is because in each round before the ALTER, every tree is flat, so the ALTER makes all edge ends roots.

Observe that if there is a tree path of $2^k + 1$ vertices just before the SHORTCUT steps in round j, there will be $\Omega(k + 1)$ SHORTCUT steps in round j. If in every round there is a *new* path of $2^k + 1$ vertices, and the algorithm requires many rounds, the input graph will be a

bad example. Our example is a disjoint union of certain graphs based on this observation. To produce a given graph G with m' edges and vertex set $[n']$ at the end of round j, it suffices to start with a graph $g(G)$ (for the *generator* of G) at the beginning of round j constructed as follows: For each vertex i in G, add two vertices, i and $i+n'$ to $g(G)$; for each vertex i in G, add an edge $(i, i+n')$ to $g(G)$; for each edge (i, i') in G, add an edge $(i+n', i'+n')$ to $g(G)$. $g(G)$ contains all the vertices of G but none of its edges; $g(G)$ contains $2n'$ vertices and $n'+m$ edges.

Consider the effect of a round of SA on $g(G)$. The CONNECT step makes i the parent of $i+n'$ for $i \in [n']$. The SHORTCUT steps do nothing. The ALTER converts each edge $(i+n', i'+n')$ for $i \in [n']$ into (i, i') and deletes each edge $(i, i+n')$ for $i \in [n']$. Thus after the round, G is the induced subgraph on the vertices that are roots. By induction, $g^r(G)$ is converted by r rounds of SA into G. $g^r(G)$ contains $n'2^r$ vertices and $n'(2^r-1)+m'$ edges.

For any positive integer $k > 1$, consider the disjoint union of the graphs P, $g(P)$, $g^2(P)$, $\ldots$, $g^{k-1}(P)$, where P is a path of 2^k+1 vertices from 1 to 2^k+1, with the vertices renumbered so that all vertices are distinct and the order within each connected component is preserved. If SA is run on this graph, round i for each i does $\Omega(k+1)$ SHORTCUT steps on the path produced by $i-1$ rounds on $g^{i-1}(P)$, so the total number of steps is $\Omega(k^2)$. The total number of vertices in this graph is $n = (2^k+1)(2^k-1) = 2^{2k}-1$. (The number of edges is $2^{2k}-k+1$: the input graph is a set of $k-1$ trees.) Thus the number of steps is $\Omega(\lg^2 n)$, making the bound in Theorem 5 tight.

4 Related Work

Previous work on concurrent algorithms for connected components was done by two different communities in two overlapping eras. First, theoretical computer scientists developed provably efficient algorithms for various versions of the PRAM (parallel random-access machine). This work began in the late 1970's and reached a natural conclusion in the work of Halperin and Zwick [8, 9], who gave $O(\lg n)$-time, $O((m+n)/\lg n)$-processor randomized algorithms for the EREW (exclusive read, exclusive write) PRAM. The EREW PRAM is the weakest PRAM model, and computing connected components in this model requires $\Omega(\lg n)$ time [5]. To solve the problem sequentially takes $O(n+m)$ time, so the Halperin-Zwick algorithms minimize both the time and the total work (number of processors times time). One of their algorithms not only finds the connected components but also a spanning tree of each. The main theoretical question remaining open is whether a deterministic algorithm can achieve the same bounds.

Halperin and Zwick's paper contains a table listing results preceding theirs, and we refer the reader to their paper for these results. Our interest is in simple algorithms for a more powerful computational model, so we content ourselves here with discussing simple labeling algorithms related to ours. (The Halperin-Zwick algorithms and many of the preceding ones are *not* simple.) First we review variants of the PRAM model and how they relate to our algorithmic framework.

The three main variants of the PRAM model, in increasing order of strength, are EREW, CREW (concurrent read, exclusive write), and CRCW (concurrent read, concurrent write). The CRCW PRAM has four standard versions that differ in how they handle write conflicts: (i) COMMON: all writes to the same location at the same time must be the of the same value; (ii) ARBITRARY: among concurrent writes to the same location, an arbitrary one succeeds; (iii) PRIORITY: among concurrent writes to the same location, the one done by

the highest-priority processor succeeds; (iv) COMBINING: values written concurrently to a given location are combined using some symmetric function. As mentioned in §3, our algorithms can be implemented on a COMBINING CRCW PRAM, with minimization as the combining function.

The first $O(\lg n)$-time PRAM algorithm was that of Shiloach and Vishkin [17]. It runs on an ARBITRARY CRCW PRAM, as do the other algorithms we discuss, except as noted. The following is a version of their algorithm in our framework:

Algorithm SV: *repeat* {SHORTCUT; PARENT-CONNECT; ARBITRARY-ROOT-UPDATE; MAX-PARENT-CONNECT; PASSIVE-ROOT-UPDATE; SHORTCUT} *until no parent changes.*

This algorithm uses the notion of a passive tree. (Shiloach and Vishkin used the term "stagnant".) A tree is *passive* if it did not change in the previous round. A passive tree is necessarily flat, but a flat tree need not be passive. Our description of their algorithm omits the extra computation needed to keep track of which trees are passive.

The algorithm uses variants of some of the methods in §3. Method ARBITRARY-ROOT-UPDATE replaces each parent of a root by *any* of the most-recently received vertices other than itself, if there are any, instead of the minimum; MAX-PARENT-CONNECT sends for each edge the maximum of the parents of the edge ends to the minimum instead of vice-versa; PASSIVE-ROOT-UPDATE replaces each root of a passive tree by any just-received vertex other than itself, if there are any:

ARBITRARY-ROOT-UPDATE: *for each root v replace $v.p$ by any just-received vertex other than v, if there is one.*

MAX-PARENT-CONNECT: *for each edge e, request $e.v.p$ from $e.v$ and $e.w.p$ from $e.w$; send the maximum of the received vertices to the minimum of the received vertices.*

PASSIVE-ROOT-UPDATE: *for each root v of a passive tree, replace $v.p$ by any just-received vertex other than v, if there is one.*

Algorithm SV does neither minimum nor maximum labeling: the first update in a round connects roots to smaller vertices, the second connects roots to larger vertices. The proof that the algorithm creates no cycles is non-trivial, as is the efficiency analysis. Most interesting for us, the first three steps of the algorithm are algorithm R, but with arbitrary parent updates rather than minimum ones. Shiloach and Vishkin claimed that the second SHORTCUT can be omitted, but they showed by example that omission of PASSIVE-ROOT-UPDATE results in an algorithm that can take $\Omega(n)$ rounds. Their example also shows that the efficiency of algorithm R depends on resolving concurrent parent updates by minimum value rather than arbitrarily. That is, our stronger model of computation is critical in obtaining a simpler algorithm.

Awerbuch and Shiloach presented a simpler $O(\lg n)$-time algorithm and gave a simpler efficiency analysis [2]. Our analysis of algorithm RA in §5 uses a variant of their potential function. Their algorithm does only one SHORTCUT per round and in updates only changes the parents of roots of flat trees, using the following method:

FLAT-ROOT-UPDATE: *for each root v of a flat tree, replace $v.p$ by any just-received vertex other than v, if there is one.*

Algorithm AS: *repeat* {PARENT-CONNECT; FLAT-ROOT-UPDATE; MAX-PARENT-CONNECT; FLAT-ROOT-UPDATE; SHORTCUT} *until no parent changes.*

The algorithm needs to do additional computation to keep track of flat trees. Although Awerbuch and Shiloach do not mention it, their analysis shows that replacing the first update in their algorithm with ARBITRARY-ROOT-UPDATE produces a correct $O(\lg n)$-time algorithm. The resulting algorithm is algorithm SV with the first SHORTCUT deleted and PASSIVE-TREE-CONNECT replaced by FLAT-TREE-CONNECT.

An even simpler but randomized $O(\lg n)$-time algorithm was proposed by Reif [16]:

Algorithm Reif: *repeat* {*for each vertex flip a coin*; RANDOM-CONNECT; ARBITRARY-ROOT-UPDATE; SHORTCUT} *until no parent changes.*

RANDOM-CONNECT: *for each edge* (v, w) *if* $v.p$ *flipped heads and* $w.p$ *flipped tails then send* $v.p$ *to* $w.p$; *if* $w.p$ *flipped heads and* $v.p$ *flipped tails then send* $w.p$ *to* $v.p$.

Reif's algorithm keeps the label trees flat. As a result, RANDOM-CONNECT only sends vertices to roots. This allows replacement of ARBITRARY-ROOT-UPDATE by a method that for each vertex sets its parent equal to any just-received vertex if there is one. Reif's algorithm is simpler than both SV and AS, but our algorithm RA is even simpler, and it is deterministic.

We know of only one algorithm, that of Johnson and Metaxis [12], that does not maintain acyclicity. Their algorithm runs in $O((\lg n)^{3/2})$ time on an EREW PRAM. It does a form of SHORTCUT to eliminate any non-trivial cycles that it creates.

Algorithms that run on a more restricted form of PRAM, or use fewer processors (and thereby do less work) use various kinds of edge alteration, along with simulation and other techniques to resolve read and write conflicts. Such algorithms are much more complicated than those above. Again we refer the reader to [8] for results and references.

The second era of concurrent connected components algorithms was that of the experimentalists. It began in the 1990's and continues to the present. Experimentation has expanded greatly with the growing importance of huge graphs (the internet, the world-wide web, relationship graphs, and others) and the development of cloud computing frameworks. These trends make concurrent algorithms for connected components both practical and useful. The general approach of experimentalists has been to take one or more algorithms in the literature, possibly simplify or modify them, implement the resulting suite of algorithms on one or more computing platforms, and report the results of experiments done on some collection of graphs. Examples of such studies include [6, 7, 10, 21, 14, 18].

Our interest is in the theoretical efficiency of such simple algorithms, and in the theoretical power of the new concurrent computing platforms as compared to the classical PRAM model. One theoretical model used to study algorithms on such platforms is the MPC (Massive Parallel Computing) model. A large number of virtual processes run concurrently in supersteps. In each superstep, a process can do arbitrary computation based on messages it received during the previous step, and then send messages to any or all other processes. The next superstep begins once all messages are received. Our framework for connected components algorithms is the MPC model, specialized for the kind of algorithms we consider.

Our work started with a study of the following algorithm of Stergio, Rughwani, and Tsioutsiouliklis [18]:

Algorithm SRT: *repeat* {
for each edge (v, w) *if* $v.p < w.p$ *then send* $v.p$ *to* w *else send* $w.p$ *to* v;
for each vertex v *let* $new(v)$ *be the minimum of* $u.p$ *and the vertices received by* v;
for each vertex v *if* $new(v) < v.p$ *then send* $new(v)$ *to* $v.p$;
for each vertex v *send* $new(v).p$ *to* v;
for each vertex v *let* $v.p$ *be the minimum of* $v.p$ *and the vertices received by* v}
until no parent changes.

They implemented this algorithm on the Hronos computing platform and solved problems with trillions of edges. They claimed a bound of $O(\lg n)$ steps for their algorithm. But we are unable to make sense of their proof. This algorithm moves subtrees between trees, which makes it hard to analyze. We conjecture, however, that our proof of $O(\lg^2 n)$ steps for algorithms P, E, and A extends to their algorithm.

Our algorithm E is a variant of algorithm SRT that is bit simpler. Algorithm P is a further simplification. Algorithm R is algorithm P with updates restricted to roots. We are able to obtain an $O(\lg n)$ step bound for algorithms R and RA, but not for E, P, nor A. Our

diameter-based analysis of algorithm E is also valid for algorithm SRT and gives a bound of $O(d)$ steps. We think that algorithms SRT, E, P, and A all take $O(\lg n)$ steps but have no proof.

A second paper with an analysis gap is that of Yan et al. [21]. They consider algorithms in the PREGEL framework [13], which is a graph-processing platform designed on top of the MPC model. All the algorithms they consider can be expressed in our framework. They give a version of algorithm SV with the first SHORTCUT deleted, and modify it further by replacing "MAX-PARENT-CONNECT; PASSIVE-ROOT-UPDATE" with code equivalent to "PARENT-CONNECT; FLAT-ROOT-UPDATE". These two steps do nothing, since any connections they might make are done by the previous steps "PARENT-CONNECT; ARBITRARY-ROOT-UPDATE". Their termination condition, that all trees are flat, is also incorrect. They claim an $O(\lg n)$ step bound for their algorithm, but since they use arbitrary updating of roots, their algorithm, once the termination condition is corrected, takes $\Omega(n)$ steps on the example of Siloach and Vishkin.

A third, more recent paper with an analysis gap is that of Burkhardt [4]. He proposes an algorithm that does a novel form of edge alteration: it converts each original edge into two oppositely directed arcs and alters these arcs independently. The algorithm does not do SHORTCUT explicitly, but it does do a variant of SHORTCUT implicitly. The algorithm maintains both old and new vertex labels, which increases its complexity. Burkhardt claims a step bound of $O(\lg d)$, but a counterexample in [1] disproves this claim. He also claims a linear space bound, but we are unable to verify his proof of this. We conjecture that Burkhardt's algorithm and simpler variants take $O(\lg n)$ steps but have no proof. We think that our $O(\lg^2 n)$ analysis should apply to his algorithm but have not verified this.

Very recently, Andoni et al. have used the power of the MPC model to obtain a randomized algorithm running in $O(\log d \log\log_{m/n} n)$ steps [1]. Their algorithm uses the distance-doubling technique of [15] (discussed in §2) but controlled to keep message sizes sufficiently small. The algorithm relies heavily on the ability to sort in $O(1)$ steps on the MPC model. We think an appropriate version of their algorithm can be implemented on a CRCW PRAM, possibly with the same asymptotic bound as theirs, and we are working to achieve this. Any such algorithm will be much more complicated than those we present here, however.

5 Analysis

In this section we prove an $O(d)$ step bound for algorithms E and A, an $O(\lg^2 n)$ bound for algorithms P, E, and A, and an $O(\lg n)$ bound for algorithm R (and RA). We begin with some assumptions and preliminary results. We assume the graph is connected and contains at least two vertices, which is without loss of generality since each algorithm operates concurrently and independently on each component. For brevity, we use CONNECT to denote the steps preceding SHORTCUT in a round, no matter which algorithm we are considering. We denote an edge e with $e.v = v$ and $e.w = w$ by (v, w). For the analysis only, we assume that ALTER does not delete an edge (v, w) when $v.p = w.p$ but instead converts it into a loop (an edge having both ends the same) $(v.p, w.p)$. Once an edge becomes a loop, it remains a loop. Loops do not affect the parent changes done by A (or RA); they merely allow us to treat a vertex that is both ends of a loop as having an incident edge.

We partition the vertices into two colors, as follows: in algorithm A, a vertex is *green* if it is a root or has an incident edge, and *red* otherwise; in all other algorithms a vertex is *red* if it is a leaf (a non-root with no children) and *green* otherwise.

▶ **Lemma 6.** *If a* CONNECT *changes the parent of a vertex v to w, then both v and w are green.*

Proof. In algorithm A, v and w must be the ends of the edge causing the edge, so both are green. In the other algorithms, there must be an edge (x, y) such that $x.p = v$ and $y.p = w$. Hence both v and w are green. ◀

▶ **Lemma 7.** *A* SHORTCUT *in an algorithm other than A changes all green non-roots with no green children to red, and changes no other vertex colors.*

Proof. Let v be any vertex just before a SHORTCUT. If v is red, it is a non-root with no child. The SHORTCUT cannot give it a child, so it stays red. If v is a green root, it stays a root, so it stays green. If v is a green non-root, v is a non-root after the SHORTCUT, and it has a child after the SHORTCUT if and only if it has a grandchild before the SHORTCUT, which is true if and only if it has a green child before the SHORTCUT. ◀

▶ **Lemma 8.** *In algorithm A, (i) each green vertex is a root or has a green parent; (ii) each red vertex has a green grandparent, and has a green parent just after a* SHORTCUT*; and (iii) a* SHORTCUT *followed by an* ALTER *changes all green non-roots with no green children to red, and changes no other vertex colors.*

Proof. By induction of the number of steps. The lemma is true initially since all vertices are roots and hence green. By Lemma 6, a CONNECT preserves the lemma, since it does not change any vertex colors nor the parent of any red vertex. A SHORTCUT preserves (i) and (ii), since before the SHORTCUT any vertex has a green grandparent or parent or is itself green, so after the SHORTCUT it has either a green parent or is green. Given that all vertices have green parents after a SHORTCUT, the subsequent ALTER changes each edge end that is a green non-root to its parent, and does not affect ends that are roots. (Here we use the non-deletion of loops.) This makes no red vertex green by (i), makes a green vertex red if and only if it is a non-root with no green children, giving (iii), and leaves each red vertex with a green parent or grandparent by (i) and (ii). ◀

▶ **Lemma 9.** *The parent of a vertex never increases. Once a vertex is a non-root, it stays a non-root.*

Proof. The first part of the lemma follows by induction on the number of parent changes: each such change decreases the parent. By the first part, once $v.p < v$ this inequality continues to hold, which gives the second part of the lemma. ◀

5.1 A Diameter Bound

▶ **Theorem 10.** *Algorithm E takes $O(d)$ steps.*

Proof. Let u be the minimum vertex. We prove by induction on i that after i rounds all vertices at distance i or less from u have parent u. This is true for $i = 0$. Let w be a vertex at distance $i > 0$ from u. Then there is an edge (w, v) with v at distance $i - 1$ from u. By the induction hypothesis, v has parent u after round $i - 1$, which it will send to w in round i, as a result of which w will have parent u after round i. After at most d rounds, all vertices will have parent u, so the algorithm stops after at most $d + 1$ rounds. ◀

Theorem 10 also holds for algorithm A, but the proof is more elaborate. We need a strengthening of the result of Lemma 3, as well as Lemma 8.

▶ **Lemma 11.** *Let u be the minimum vertex. After $d-i$ rounds of algorithm* A*, each green vertex has a path of at most $d-i$ current edges connecting it with u.*

Proof. By induction on i. The lemma is true initially, since very vertex has a path to u containing at most d edges. Suppose it is true just before round i. Let v be a green vertex at the end of round i. If v is a root at the end of round i, then it was a root at the start of round i, and by the induction hypothesis there was a path P of at most $d-i$ edges connecting v with u at the start of round i. If v is not a root at the end of round i, then it was not a root at the beginning of the SHORTCUT in round i, and by Lemma 8 it had a green child, say w, at this time. By the induction hypothesis, there was a path P of at most $d-i$ edges connecting w with u at the start of round i. In either case we can assume P is loop-free, since deleting loops leaves it a path with the same ends. The ALTER in round i converts P into a path P' connecting v with u containing at most $d-i$ edges. Delete all loops from P'. Let (x,u) be the last edge on P. After the CONNECT in round i, x must be a child of u. Hence the ALTER in round i converts this edge to a loop. Deleting this loop from P' gives a path satisfying the lemma for v after round i. ◀

▶ **Theorem 12.** *Algorithm A takes $O(d)$ steps.*

Proof. Consider the state after d steps. By Lemma 11, the only green vertex is the minimum vertex, and by Lemma 8 all red vertices are children or grandchildren of this minimum vertex. The algorithm stops after at most two more rounds. ◀

5.2 A Log-Squared Bound

We conjecture that all our algorithms take $O(\lg n)$ steps, but for P, E, and A we are only able to prove something weaker:

▶ **Theorem 13.** *Algorithms P, E, and A take $O(\lg^2 n)$ steps.*

To prove Theorem 13, we shall show that $O(\lg n)$ rounds reduce the number of green vertices by at least a factor of two. Given Lemma 6, it is convenient to consider the situation just before a SHORTCUT. By Lemma 7 or 8 depending on the algorithm, a SHORTCUT converts all green non-roots with no green children to red. We shall prove that a CONNECT converts all green roots with no green children into non-roots. This allows us to bound the number of rounds in which there are many green vertices with no green children. To bound the number of rounds in which there are many green vertices but few that have no green children, we prove that in such a situation the green vertices form many long vertex-disjoint tree paths, on which we can quantify the effect of a SHORTCUT. We proceed with the details. We assume throughout this section that the algorithm is P, E, or A.

▶ **Lemma 14.** *Let v be a root with no green children just before the* SHORTCUT *in round i. Either v is the only root, or the* CONNECT *in round $i+1$ makes v a non-root.*

Proof. Suppose v is not the only root just before the SHORTCUT in round i. In algorithm A, just before this SHORTCUT there must be an edge (v,w) with w not in the tree rooted at v. If $w < v$, then the edge formed from (v,w) by the ALTER in round i will cause the CONNECT in round $i+1$ to make v a non-root. If $w > v$, then $w.p < v$ after the CONNECT in round i; otherwise, this CONNECT would have made w, a green vertex, a child of v. The ALTER in round i converts (v,w) into an edge (v,x) with $x < v$, which causes the CONNECT in round $i+1$ to make v a non-root in this as well.

A similar argument applies to algorithms P and E. Just before the SHORTCUT in round i, there must be an edge (x,y) with $x.p = v$ and with $y.p$ not in the tree rooted at v. If

$y.p < v$, (x, y) will cause the CONNECT in round $i + 1$ to make v a non-root. If $y.p > v$, then $y.p.p < v$ after the CONNECT in round i; otherwise, this CONNECT would have made $y.p$, a green vertex, a child of v. After the SHORTCUT in round i, $y.p < v$, so in this case also (v, w) causes the CONNECT in round $i + 1$ to make v a non-root. ◀

▶ Remark. Lemma 14 does *not* hold for algorithm R: a root with no green children can remain a root for many rounds.

The *depth* of a vertex v is the number of proper ancestors in its tree; that is, the number of tree arcs on the path from v to the root.

▶ **Lemma 15.** *If there are at least n' green vertices, with less than n'/k having no green child, then there is a green vertex of depth at least k.*

Proof. Every green vertex has a green descendant with no green children (possibly itself). Any two green vertices of the same depth are unrelated and hence must have distinct green descendants with no green children. Hence the number of green vertices of any given depth is less than n'/k. Since there are at least n' green vertices, and less than n'/k of each depth from 0 to $k - 1$, there must be at least one of depth k. ◀

▶ **Lemma 16.** *If there are n' green vertices, with less than $n'/(2k)$ having no green child, then there is a set of vertex-disjoint tree paths of green vertices, each containing at least $k + 1$ vertices, that together contain at least $n'/2$ green vertices.*

Proof. Find a green vertex of maximum depth and delete the tree path from it to the root of its tree. This deletion creates no new green vertex with no green child. Repeat this step until there are fewer than $n'/2$ vertices. By Lemma 15, each path deleted contains at least $k + 1$ vertices. ◀

To quantify the effect of a SHORTCUT on long paths, we borrow an idea from the analysis of compressed-tree algorithms for disjoint set union [19]. Suppose there are n' green vertices. For the purpose of the analysis only, we renumber the green vertices from 1 to n' in the order of their original numbers and identify each green vertex by its new number. We define the *level* $v.l$ of a green child v to be $v.l = \lfloor \lg(v - v.p) \rfloor$. The level of a green child is between 0 and $\lg n'$. By Lemma 9, which holds for the new vertex numbers as well as the original ones, the level of a child never decreases. We show that if k is big enough a SHORTCUT increases by $\Omega(k)$ the sum of the levels of the green vertices on a tree path of k green vertices.

▶ **Lemma 17.** *Let n' be the number of green vertices. Consider a tree path of at least $k + 2$ green vertices, where $k \geq 2 \lg n'$. A* SHORTCUT *increases the sum of the levels of the children on the path by at least $k/4$.*

Proof. Let u be a green vertex on the path other than the last two. Let v be the parent and w the grandparent of u, and let i and j be the levels of u and v, respectively. A SHORTCUT increases the level of u from $i = \lfloor \lg(u-v) \rfloor$ to $\lfloor \lg(u-w) \rfloor = \lfloor \lg(u-v+v-w) \rfloor \geq \lfloor \lg(2^i+2^j) \rfloor$. If $i < j$, this increases the level of u to at least j; if $i = j$, it increases the level of u to $i + 1$.

Let $x_1, x_2, \ldots, x_{k+1}$ be the vertices on the path, excluding the last one. For each i from 1 to k, let $\Delta_i = x_{i+1}.l - x_i.l$. The sum of Δ_i's is $\Sigma = x_{k+1}.l - x_1.l > 0 - \lg n'$ since they form a telescoping series. Let k_+, k_0, and k_-, respectively be the number of positive, zero, and negative Δ_i's, and let Σ_+ and Σ_- be the sum of the positive Δ_i's and the sum of the negative Δ_i's, respectively. Then $\Sigma = \Sigma_+ + \Sigma_-$, which implies $\Sigma_+ > -\Sigma_- - \lg n'$. Since the Δ_i's are integers, $\Sigma_+ \geq k_+$ and $-\Sigma_- \geq k_-$. Combining inequalities, we obtain $2\Sigma_+ > k_+ + k_- - \lg n'$. Adding k_0 to both sides gives $2\Sigma_+ + k_0 > k - \lg n' \geq k/2$. Hence $\Sigma_+ + k_0 \geq \Sigma_+ + k_0/2 > k/4$. The lemma follows from the argument in the previous paragraph. ◀

Now we have all the pieces. It remains to put them together.

▶ **Lemma 18.** *Suppose there are $n' \geq 16$ green vertices just before a round. After $O(\lg n')$ rounds, there are at most $n'/2$ green vertices.*

Proof. At the beginning of the round, renumber the green vertices from 1 to n' in the order of their original numbers and identify each green vertex by its new number. Consider a round in which there are still at least $n'/2$ green vertices just before the SHORTCUT. We consider three cases:

(i) There are at least $n'/(16 \lg n' + 8)$ green roots with no green children. By Lemma 14, all such roots become non-roots during the next CONNECT. This can happen at most $16 \lg n' + 7$ times, since there is always a green root.

(ii) There are at least $n'/(16 \lg n' + 8)$ green non-roots with no green children. By Lemma 7 or 8 depending on the algorithm, the SHORTCUT makes all such vertices red. This can happen at most $16 \lg n' + 8$ times.

(iii) There are fewer than $n'/(8 \lg n' + 4)$ green vertices with no green children. By Lemma 16, there is a set of vertex-disjoint green paths, each containing at least $2 \lg n' + 2$ vertices, that together contain at least $n'/4$ vertices. By Lemma 17, for any such path containing $k + 2$ vertices, the SHORTCUT increases the levels of the vertices on the path by at least $k/4 \geq k/5 + 2/5$, since $k \geq 2 \lg n' \geq 8$. Summing over all the paths, the SHORTCUT increases the sum of levels by at least $n'/20$. This can happen at most $20 \lg n'$ times.

We conclude that after $O(\lg n')$ rounds, the number of green vertices is at most $n'/2$. ◀

Theorem 13 is immediate from Lemma 18.

5.3 A Logarithmic Bound

The proof of Theorem 13 fails for algorithm R, because Lemma 14 is false for this algorithm. But we can get an even better bound by using a different technique, that of Awerbuch and Shiloach [2] extended to cover a constant number of rounds rather than just one.

▶ **Theorem 19.** *Algorithms R (and hence RA) takes $O(\lg n)$ steps.*

To prove Theorem 19, we begin with some preliminary results and some terminology.

▶ **Lemma 20.** *After two rounds of algorithm R, each tree contains at least two vertices.*

Proof. Let v be a vertex and (v, w) an incident edge. If $w < v$, then v becomes a non-root in the first CONNECT. If $w > v$, then the first CONNECT either makes w a child of v or gives w a parent less than v. In the latter case, if v is still a root before the second CONNECT then this CONNECT will make v a non-root. We conclude that after the first two CONNECT steps v is in a tree with at least two vertices. ◀

By Lemma 20, after round two, all trees have height at least one. When we speak of a tree *in round k*, we mean a tree existing at the end of the round. We say a CONNECT *link* trees T_1 and T_2 if it makes the root of one of the trees a child of a vertex in the other. If the CONNECT makes the root of T_1 (respectively T_2) a child of a vertex in T_2 (respectively T_1), we say the CONNECT *links* T_1 to T_2 (respectively T_2 to T_1).

▶ **Definition 21.** A tree in round $k > 2$ is *passive* in round k if the tree existed at the beginning of the round (round k does not change it), and *active* otherwise. All trees in round 2 are *active* in round 2.

▶ **Lemma 22.** *For any integer $k > 2$, if trees T_1 and T_2 are passive in round $k-1$, the* CONNECT *in round k does not link T_1 and T_2, and there is no edge with one end in T_1 and the other in T_2.*

Proof. If T_1 and T_2 were linked in round k, there would be an edge connecting them that caused the link. Since T_1 and T_2 did not change in round $k-1$, such an edge would have caused them to link in round $k-1$, and hence they would not be passive. ◀

We measure progress in the algorithm using a *potential function*. It is like that of Awerbuch and Shiloach, but modified to guarantee that the total potential never increases, and to give passive trees, which can linger indefinitely, a potential of zero.

▶ **Definition 23.** The *potential* $\phi_k(T)$ of a tree T in round $k \geq 2$ is

$$\phi_k(T) = \begin{cases} 0 & \text{if } T \text{ is passive in round } k \\ 3 & \text{if } T \text{ is active and flat in round } k \\ h+1 & \text{if } T \text{ has height } h \geq 2 \text{ in round } k \end{cases}$$

The *total potential* in round $k \geq 2$ is the sum of the potentials of all the trees in the round.

We shall obtain a bound on the total potential that decreases by a constant factor each round (other than the first two). Theorem 19 is immediate from such a bound. It suffices to consider each active tree individually, since algorithm R is monotone and the total potential in a round is the sum of the potentials of the trees in the round. We shall track an active tree T backward through the rounds in order to see what earlier trees were combined to form it. We show that these earlier trees had enough potential to give the desired potential decrease.

▶ **Definition 24.** Let T be a tree that is active in round $k > 2$. For i such that $2 \leq i < k$, the *constituent trees* of T in round i are those in round i whose vertices are in T. The *potential* $\Phi_i(T)$ of T in round i is the sum of the potentials of the constituent trees of T in round i. In particular, $\Phi_k(T) = \phi_k(T)$.

▶ **Lemma 25.** *Let T be active in round $k > 2$. For i such that $2 \leq i < k$, the constituent trees of T in round i include at least one active tree.*

Proof. By induction on i for decreasing i. The lemma holds for $i = k$ by assumption and for $i = 2$ since all trees in round 2 are active. Suppose it holds for $i > 3$. If the constituent trees of T in round $i-1$ are all passive, the CONNECT in round i changes none of these trees by Lemma 22. Since all these trees are flat, the SHORTCUT in round i also changes none of them. Thus all these trees are passive in round i, contradicting the induction hypothesis. ◀

▶ **Lemma 26.** *Let T be active in round $k > 2$. Then $\Phi_{k-1}(T) \geq \Phi_k(T)$, and if $\Phi_k(T) \geq 5$ then $\Phi_{k-1}(T) \geq (6/5)\Phi_k(T)$.*

Proof. Let h be the sum of the heights of the constituent trees of T in round $k-1$, let t be the number of these constituent trees that are active in round $k-1$, and let f be the number of these trees that are active and flat. Then $\Phi_{k-1}(T) = h + t + f$. Consider the tree T' formed from the constituent trees of T by the CONNECT in round k. The SHORTCUT in round k transforms T' into T. By Lemma 22, along any path in T', there cannot be consecutive vertices from two different passive trees. By Lemma 6, all leaves of constituent trees are leaves of T'. It follows that T' has height at most $h + t + 1$: the active constituent trees contribute at most h vertices to a longest path, at most $t + 1$ roots of passive trees

are on the path, at most one leaf of some constituent tree is on the path, and the path has length one less than its number of vertices. Thus $\Phi_k(T) \leq \lceil (h+t+1)/2 \rceil + 1$ if $h+t > 2$, $\Phi_k(T) = 3$ if $h+t = 2$.

We prove the lemma by induction on $h+t$, which is at least 2, since at least one of the constituent trees must be active by Lemma 25. If $h+t=2$, there is one active constituent tree, and it is flat, so $f=1$. If $h+t=3$, there is one active constituent tree, and it is not flat. In both these cases, $\Phi_{k-1}(T) = 3$ and $\Phi_k(T) = 3$, so the lemma holds. Suppose $h+t > 3$. Then $\Phi_k(T) \leq \lceil (h+t+1)/2 \rceil + 1$. If $h+t=4$, $\Phi_{k-1}(T) \geq 4$ and $\Phi_k(T) \leq 4$; if $h+t=5$, $\Phi_{k-1}(T) \geq 5$ and $\Phi_k(T) \leq 4$; if $h+t=6$, $\Phi_{k-1}(T) \geq 6$ and $\Phi_k(T) \leq 5$. Thus the lemma holds in these cases. Each increase of $h+t$ by two increases the lower bound on $\Phi_{k-1}(T)$ by two and increases the upper bound on $\Phi_k(T)$ by one, which preserves the inequality $\Phi_{k-1}(T) \geq (6/5)\Phi_k(T)$, so the lemma holds for all $h+t$ by induction: if $\Phi_k(T) \geq 5$, it must be the case that $h+t \geq 6$. ◀

▶ **Corollary 27.** *Let T be an active tree of height at least four in round $k \geq 3$. Then $\Phi_{k-1}(T) \geq (6/5)\Phi_k(T)$.*

Corollary 27 gives us the desired potential drop for any active tree of height at least four. It remains to consider active trees of heights one, two, and three. Since active trees of height one and two have the same potential, namely three, we shall handle these cases together. This gives us two cases: height at most two, and height three. In order to obtain the desired potential drop, we need to look backward up to two rounds in the case of height three, up to five in the case of height at most two.

▶ **Lemma 28.** *Let T be an active tree of height 3 in round $k \geq 3$. Let $j = \max\{2, k-2\}$. Then $\Phi_j(T) \geq (5/4)\Phi_k(T)$.*

Proof. If the constituent trees of T in round $k-1$, or in round $k-2$ if $k > 3$, include at least two active trees (of total potential at least six) or one active tree of height at least four (of potential at least five), then the lemma holds by Lemma 26, since $\Phi_k(T) = 4$. If not, by Lemma 25 the constituent trees of T in round $k-1$, and in round $k-2$ if $k > 3$, include exactly one active tree, of height exactly three. In this case there must be at least two constituent trees of T in round $k-1$, and in round $k-2$ if $k > 3$, since the tree formed from these constituent trees by the CONNECT in the next round must have height at least five, in order for the SHORTCUT in this round to produce a tree of height three. But this implies $k > 4$, since all trees in round 2 are active.

We conclude that the lemma holds except in the following case: $k > 4$ and the constituent trees of T in rounds $k-2$ and $k-1$ each include exactly one active tree, of height exactly three. Let T_2 and T_1, respectively, be the active constituent trees of T in rounds $k-2$ and $k-1$. Let T_1' and T', respectively, be the trees formed from the constituent trees of T by the CONNECT steps in rounds $k-1$ and k. The SHORTCUT in round $k-1$ transforms T_1' into T_1, and the SHORTCUT in round k transforms T' into T. Both T_1' and T' have height exactly five. The passive constituent trees of T in round $k-1$ are a proper subset of those in round $k-2$; specifically, those that are not combined with T_2 to form T_1' in the CONNECT of round $k-1$. By Lemma 22, no edge connects two passive constituent trees of T in round $k-2$.

Call a passive constituent tree of T in round $k-2$ *primary* if it has an edge connecting it to the root of T_2 or to a child of the root of T_2. Since T_1' has height five, its root must be different from that of T_2: if not, T_1' would have height at most four by the argument in the proof of Lemma 26. If the root of T_1' is different from that of T_2, its root must be the minimum of the roots of the primary trees of T in round $k-2$. The root of T' must also be

different from the root of T_1 (which is the same as the root of T_1'). But the root of T' cannot be the root of one of the primary passive constituent trees of T in round $k-2$, since none of these roots are smaller than the root of T_1'. Nor can it be the root of one of the non-primary passive constituent trees of T in round $k-2$, since such a tree has no edge connecting it to the root or a child of the root of T_2, implying that it has no edge connecting it to the root, a child of the root, or a grandchild of the root of T_1', further implying that it has no edge connecting it to the root or a child of the root of T_1, making it impossible for the CONNECT in round k to link T_1 to it. Thus this case is impossible. ◀

The analysis of an active tree of height at most two is like that in Lemma 28 but more complicated: we must consider *all* the passive constituent trees in the first relevant round, not just the primary ones. At a high level the argument is the same: if in one of the four rounds preceding round k there are two active constituent trees, or one of height at least three, we obtain the desired potential drop; if not, the algorithm eventually runs out of passive trees to link to the one active tree, which is impossible. We proceed with the details.

▶ **Lemma 29.** *Let T be an active tree of height at most two in round $k \geq 4$. Let $j = \max\{2, k-5\}$. Then $\Phi_j(T) \geq (4/3)\Phi_k(T)$.*

Proof. If for some i such that $j \leq i < k$ the constituent trees of T in round i include at least two active trees (of total potential at least six) or one active tree of height at least three (of potential at least four), then the lemma holds by Lemma 26, since $\Phi_k(T) = 3$. If not, by Lemma 25 the constituent trees of T in each round from j to $k-1$ include exactly one active tree, of height at most two. In this case the passive constituent trees of T in round i are a (not necessarily proper) subset of those in round i', for $j \leq i < i' < k$. Furthermore there must be at least two constituent trees of T in round $k-2$, and hence in round j, since otherwise the active constituent tree of T in round $k-1$ would be flat (because the CONNECT in round $k-1$ does nothing), and this tree would be equal to T, making T passive, a contradiction. Since all trees in round 2 are active, this makes the lemma true if $k \leq 7$.

We conclude that the lemma holds except in the following case: $k > 7$ and the constituent trees of T in each round from j to $k-1$ inclusive each include exactly one active tree, of height at most two. For each round i from j to k inclusive, let T_i be the active constituent tree of T in round i (so $T_k = T$), and for each round i from $j+1$ to k inclusive, let T_i' be the tree formed from the constituent trees of T by the CONNECT in round i. For $j < i \leq k$, the SHORTCUT in round i transforms T_i' into T_i.

By Lemma 22, no edge connects two passive constituent trees of T in round j, nor in any later round. Call a passive constituent tree of T in round j *primary* if it has an edge connecting it to the root of T_j or to a child of the root of T_j, *secondary* otherwise. Every passive constituent tree of T in round j must have an edge connecting it to T_j, since it does not have an edge connecting it to another passive constituent tree, and some CONNECT must link it with an active tree by the end of round k. Since T_j has height at most two, each secondary constituent tree of T in round j has an edge connecting it with a grandchild of the root of T_j.

We consider two cases: the roots of T_j and T_{j+1}' are the same, or they are different. (See Figure 2.) In the former case, the roots of all primary constituent trees of T in round j are greater than the root of T_j, and the CONNECT in round $j+1$ makes all of them children of the root of T_j. In the latter case, the root of T_{j+1}' is the minimum of the roots of the primary constituent trees of T in round j, and each such tree other than the one of minimum root is linked to T_j in round $j+1$ or to T_{j+1} in round $j+2$.

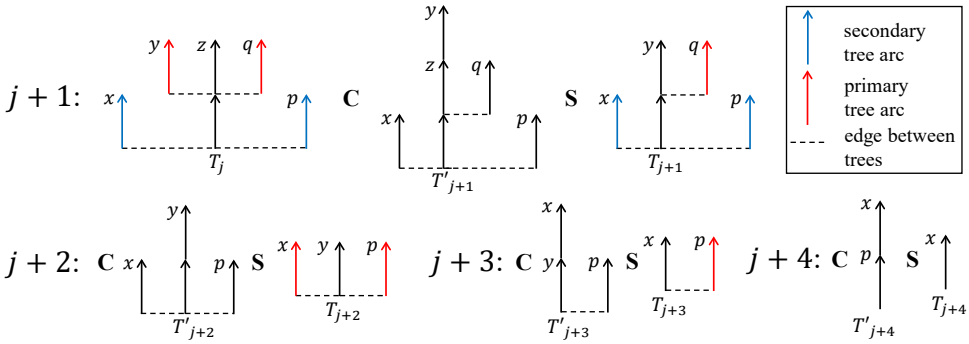


Figure 2 Worst-case illustration for the case of height at most two from round $j+1$ to round $j+4$. Only necessary vertices, arcs, and edges are shown. Vacant before **C** (CONNECT) or **S** (SHORTCUT) denotes the same forest as in the previous step. The vertices (roots) satisfy $x < p < y < q < z$.

Now consider the secondary constituent trees of T in round j. If the roots of T_j and T'_{j+1} are the same, then after the SHORTCUT in round $j+1$ each secondary constituent tree whose vertices are not in T_{j+1} has an edge connecting it with a child of the root of T_{j+1}. By the argument in the preceding paragraph, each such tree will be linked with T_{j+1} in round $j+2$ or with T_{j+2} in round $j+3$. If the roots of T_j and T'_{j+1} are different, none of the secondary constituent trees of T in round j has an edge connecting it with the root or a child of the root of T_{j+1} at the end of round $j+1$. In this case, the roots of T_{j+1} and T_{j+2} must be the same, so after the SHORTCUT in round $j+2$ each secondary constituent tree whose vertices are not in T_{j+2} has an edge connecting it with a child of the root of T_{j+2}. Each such tree will be linked with T_{j+2} in round $j+3$ or with T_{j+3} in round $j+4$. Furthermore the roots of T_{j+3} and T_{j+4} must be the same.

It follows that there is only one constituent tree of T in round $j+4$, and this tree is flat. But this tree must be T, making T passive in round $j+5=k$, a contradiction. Thus this case is impossible. ◀

Having covered all cases, we are ready to put them together into a proof of Theorem 19. Let $a = (4/3)^{1/5} \simeq 1.06$. We denote the number of vertices in a tree T by $|T|$.

▶ **Lemma 30.** *Let T be an active tree in round $k \geq 2$. Then $\Phi_k(T) \leq 2|T|/a^{k-2}$.*

Proof. By induction on k. The lemma holds for $k=2$ and $k=3$ since each active tree T after round two has at least two vertices and potential at most $|T|+1$, which is at most $2|T|$ in round two and at most $2|T|/a$ in round three since $a < 4/3$.

To prove the the lemma for $k \geq 4$, suppose the lemma holds for all k' such that $2 \leq k' < k$. We consider three cases. If the height of T exceeds three, then $\Phi_k(T) \leq (5/6)\Phi_{k-1}(T) \leq (5/6)2|T|/a^{k-3}$ by Corollary 27, the induction hypothesis, and the linearity of the potential function. The lemma holds for T since $a < 6/5$. If the height of T equals three, then $\Phi_k(T) \leq (4/5)\Phi_{k-2}(T) \leq (4/5)2|T|/a^{k-4}$ by Lemma 28, the induction hypothesis, and the linearity of the potential function. The lemma holds for T since $a^2 < 5/4$. If the height of T is at most two, then $\Phi_k(T) \leq (3/4)\Phi_j(T)$ by Lemma 26 and Lemma 29, where $j = \max\{2, k-5\}$. By the induction hypothesis and the linearity of the potential function, $\Phi_k(T) \leq (3/4)2|T|/a^{j-2}$. The lemma holds for T since $k - j \leq 5$ and $a = (4/3)^{1/5}$. ◀

Proof of Theorem 19. By Lemma 30, if k is such that $2n/a^{k-2} \leq 2$, then no tree can be active. This inequality is equivalent to $k \geq \lg n / \lg a + 2$. Every round except the last one has at least one active tree. Thus the number of rounds is at most $\lceil \lg n / \lg a \rceil + 2$. ◀

6 Remarks

We have presented several very simple label-update algorithms to compute connected components concurrently. We have shown that two of our algorithms, algorithms R and RA, take $O(\lg n)$ steps and $O(m \lg n)$ total messages, and the others take $O(\lg^2 n)$ steps and $O(m \lg^2 n)$ total messages. Crucial to our algorithms is the use of minimum-value resolution of write conflicts. We do not have tight efficiency analyses for our algorithms other than R and RA, and we leave obtaining such analyses as an open problem. We think our algorithms are simple enough to merit experimental study, but we leave this for future work.

Our analysis of algorithm R is novel in that it studies what happens over several rounds. Previous algorithms were designed so that they could be analyzed one round at a time. We have sacrificed simplicity in the analysis for simplicity in the algorithm.

We have ignored message contention. We think it is most fruitful to handle such contention separately from the underlying algorithm. Dealing with contention is a topic for future work, as is reducing the amount of synchronization required and developing incremental and batch-update algorithms. We think that concurrent algorithms for disjoint set union [11], the incremental version of the connected components problem, may be adaptable to give good asynchronous concurrent algorithms for the connected components problem with batch edge additions.

Another interesting extension of the connected components problem is to construct a forest of spanning trees of the components. It is easy to modify algorithm R to do this: when an edge causes a root to become a child, add the corresponding original edge to the spanning forest. It is not so obvious how to extend algorithms such as A, P, and E that move subtrees. This is perhaps another reason to favor algorithm R in practice.

References

1 Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685, 2018.

2 Baruch Awerbuch and Yossi Shiloach. New Connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM. *IEEE Trans. Computers*, 36(10):1258–1263, 1987.

3 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017.

4 Paul Burkhardt. Graph connectivity in log-diameter steps using label propagation. *CoRR*, 2018. `arXiv:1808.06705`.

5 Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes. *SIAM J. Comput.*, 15(1):87–97, 1986.

6 Steve Goddard, Subodh Kumar, and Jan F. Prins. Connected components algorithms for mesh-connected parallel computers. In *Parallel Algorithms, Proceedings of a DIMACS Workshop, Brunswick, New Jersey, USA, October 17-18, 1994*, pages 43–58, 1994.

7 John Greiner. A Comparison of Parallel Algorithms for Connected Components. In *SPAA*, pages 16–25, 1994.

8 Shay Halperin and Uri Zwick. An Optimal Randomised Logarithmic Time Connectivity Algorithm for the EREW PRAM. *J. Comput. Syst. Sci.*, 53(3):395–416, 1996.

9 Shay Halperin and Uri Zwick. Optimal randomized EREW PRAM algorithms for finding spanning forests. *Journal of Algorithms*, 39(1):1–46, 2001.

10 Tsan-Sheng Hsu, Vijaya Ramachandran, and Nathaniel Dean. Parallel implementation of algorithms for finding connected components in graphs. *Parallel Algorithms: Third DIMACS Implementation Challenge, October 17-19, 1994*, 30:20, 1997.

11 Siddhartha V. Jayanti and Robert E. Tarjan. A Randomized Concurrent Algorithm for Disjoint Set Union. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 75–82, 2016.

12 Donald B. Johnson and Panagiotis Takis Metaxas. A Parallel Algorithm for Computing Minimum Spanning Trees. *J. Algorithms*, 19(3):383–401, 1995.

13 Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 135–146, 2010.

14 Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! But at what COST? In *15th Workshop on Hot Topics in Operating Systems, HotOS XV, Kartause Ittingen, Switzerland, May 18-20, 2015*, 2015.

15 Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Anish Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 50–61, 2013.

16 John H Reif. Optimal Parallel Algorithms for Graph Connectivity. Technical report, Harvard University Cambridge, MA, Aiken Computation Lab, 1984.

17 Yossi Shiloach and Uzi Vishkin. An O(log n) Parallel Connectivity Algorithm. *J. Algorithms*, 3(1):57–67, 1982.

18 Stergios Stergiou, Dipen Rughwani, and Kostas Tsioutsiouliklis. Shortcutting Label Propagation for Distributed Connected Components. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 540–546, 2018.

19 Robert Endre Tarjan and Jan van Leeuwen. Worst-case Analysis of Set Union Algorithms. *J. ACM*, 31(2):245–281, 1984.

20 Leslie G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, 1990.

21 Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. Pregel Algorithms for Graph Connectivity Problems with Performance Guarantees. *PVLDB*, 7(14):1821–1832, 2014.

A Framework for Searching in Graphs in the Presence of Errors

Dariusz Dereniowski[1]
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology
Narutowicza 11/12, 80-233 Gdańsk, Poland
deren@eti.pg.edu.pl
https://orcid.org/0000-0003-4000-4818

Stefan Tiegel
Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
tiegels@student.ethz.ch

Przemysław Uznański
Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
przemyslaw.uznanski@inf.ethz.ch
https://orcid.org/0000-0002-8652-0490

Daniel Wolleb-Graf
Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
daniel.graf@inf.ethz.ch
https://orcid.org/0000-0002-6137-5725

Abstract

We consider a problem of searching for an unknown target vertex t in a (possibly edge-weighted) graph. Each *vertex-query* points to a vertex v and the response either admits that v is the target or provides any neighbor s of v that lies on a shortest path from v to t. This model has been introduced for trees by Onak and Parys [FOCS 2006] and for general graphs by Emamjomeh-Zadeh et al. [STOC 2016]. In the latter, the authors provide algorithms for the error-less case and for the independent noise model (where each query independently receives an erroneous answer with known probability $p < 1/2$ and a correct one with probability $1 - p$).

We study this problem both with adversarial errors and independent noise models. First, we show an algorithm that needs at most $\frac{\log_2 n}{1-H(r)}$ queries in case of *adversarial* errors, where the adversary is bounded with its rate of errors by a known constant $r < 1/2$. Our algorithm is in fact a simplification of previous work, and our refinement lies in invoking an amortization argument. We then show that our algorithm coupled with a Chernoff bound argument leads to a simpler algorithm for the independent noise model and has a query complexity that is both simpler and asymptotically better than the one of Emamjomeh-Zadeh et al. [STOC 2016].

Our approach has a wide range of applications. First, it improves and simplifies the Robust Interactive Learning framework proposed by Emamjomeh-Zadeh and Kempe [NIPS 2017]. Secondly, performing analogous analysis for *edge-queries* (where a query to an edge e returns its endpoint that is closer to the target) we actually recover (as a special case) a noisy binary search algorithm that is asymptotically optimal, matching the complexity of Feige et al. [SIAM J. Comput. 1994]. Thirdly, we improve and simplify upon an algorithm for searching of *unbounded* domains due to Aslam and Dhagat [STOC 1991].

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases graph algorithms, noisy binary search, query complexity, reliability

[1] Partially supported by National Science Centre (Poland) grant number 2015/17/B/ST6/01887.

2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy Fineman and Michael Mitzenmacher; Article No. 4; pp. 4:1–4:17

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.4

Related Version https://arxiv.org/abs/1804.02075

1 Introduction

Consider the following game played on a simple connected graph $G = (V, E)$:

> Initially, the *Responder* selects a *target* $v^* \in V$. In each *round*, the Questioner asks a *vertex-query* by pointing to a vertex v of G, and the Responder provides a *reply*. The reply either states that v is the target, i.e., $v = v^*$, or provides an edge incident to v that lies on a shortest path to the target, breaking ties arbitrarily. A specific number of replies can be erroneous (we call them *lies*). The goal is to design a strategy for the Questioner that identifies v^* using as few queries as possible.

We remark that this problem is known, among several other names, as Rényi-Ulam games [38, 41], *noisy binary search* or *noisy decision trees* [20, 24, 5]. One needs to put some restriction as how often the Responder is allowed to lie. Following earlier works, we focus on the most natural probabilistic model, in which each reply is independently correct with a certain fixed probability.

This problem has interesting applications in noisy interactive learning [1, 18, 25, 29, 40]. In general terms, the learning process occurs as a version of the following scheme. A user is presented with some *information* – this information reflects the current state of knowledge of the system and should take into account earlier interactions with the user (thus, the process is interactive). Then, the user responds, which provides a new piece of data to the system. In order to model such dynamics as our problem, one needs to place some rules: what the information should look like and what is allowed as a valid user's response. A crucial element in those applications is the fact that the learning process (reflected by queries and responses) does not require an explicit construction of the underlying graph on which the process takes place. Instead, it is enough to argue that there *exists* a graph whose vertices reflect possible states. Moreover, this graph needs to have the property that a valid user's response reveals an edge lying on a shortest path to the state that needs to be determined by the system. Specific applications pointed out in [18] are the following. In *learning a ranking* the system aims at learning user's preference list [36, 30]. An information presented to the user is some list, and as a response the user swaps two consecutive elements on this list which are in the wrong order with respect to the user's target preference list. Or, the response may reveal which element on a presented list has the highest rank. Both versions of the response turn out to be consistent with our graph-theoretic game over a properly defined graph, whose vertex set is the set of all possible preference lists. Another application is *learning a clustering*, where the user's reply tells the system that in the current clustering some cluster needs to be split (the reply does not need to reveal how) or two clusters should be merged [3, 4]. Yet another application includes learning a binary classifier. The strength that comes from a graph-theoretic modeling of those applications as our game is that, although the underlying graph structure has usually exponential number of vertices (for learning a ranking it is $l!$, where l is the maximum length of the preference list), the number of required queries is asymptotically logarithmic in this size [19, 18]. Thus, the learning strategies derived from the algorithms in [19] and [18] turn out to be quite efficient. We stress out that the lies in the query game reflect the fact that the user may sometimes provide incorrect replies. We

also note that any improvement of those algorithms, at which we aim in this work, leads to immediate improvements in the above-mentioned applications.

In [19], the authors provide an algorithm with the following *query complexity*, i.e., the worst-case number of vertex-queries:

$$\frac{1}{1-H(p)}\left(\log_2 n + \mathcal{O}(\frac{1}{C}\log n + C^2 \log \delta^{-1})\right), \text{where } C = \max\left((\frac{1}{2}-p)\sqrt{\log\log n}, 1\right) \quad (1)$$

that identifies the target with probability at least $1-\delta$, where n is the number of vertices of an input graph and $H(p) = -p\log p - (1-p)\log(1-p)$ is the entropy and p is the success probability of a query. It is further observed that when $p < 1/2$ is constant (w.r.t. to n), then (1) reduces to $\frac{\log_2 n}{1-H(p)} + o(\log n) + \mathcal{O}(\log^2 \delta^{-1})$. However, this complexity deteriorates when $1/2 - p = \mathcal{O}(1/\sqrt{\log\log n})$, and then (1) becomes $\mathcal{O}(\frac{1}{1-H(p)}(\log n + \log \delta^{-1}))$.

1.1 Our Contribution – Improved Query Complexity

In our analysis, we first focus on an *adversarial* model called *linearly bounded*, in which a rate of lies $r < 1/2$ is given at the beginning of the game and the Responder is restricted so that at most rt lies occur in a game of length t. It turns out that this model is easier to analyze and leads to the following theorem whose proof is postponed to Section 3.3.

▶ **Theorem 1.** *In the linearly bounded error model, with known error rate $r < 1/2$, the target can be found in at most $\frac{\log_2 n}{1-H(r)}$ vertex queries.*

This bound is strong enough to make an improvement in the probabilistic model. By a simple application of Chernoff bound, we get the following query complexity.

▶ **Theorem 2.** *In the probabilistic error model with error probability $p < 1/2$, the target can be found using at most*

$$\frac{1}{1-H(p)}\left(\log_2 n + \mathcal{O}(\sqrt{\log n \log \delta^{-1}}) + \mathcal{O}(\log \delta^{-1})\right)$$

vertex queries, correctly with probability at least $1-\delta$.

By an application of Young's inequality[2] and assuming that $p < 1/2$ is constant, we derive a query complexity of

$$\frac{\log_2 n}{1-H(p)} + o(\log n) + \mathcal{O}(\log \delta^{-1} \log\log \delta^{-1}).$$

Query complexity comparison

We compare, in the independent noise model, the precise query complexities of [19], i.e. (1) with Theorem 2. Observe that $\log n \cdot \frac{1}{C} + \log \delta^{-1} \cdot C^2 \geq 2\sqrt{\log n \log \delta^{-1} \cdot \sqrt{C}} \geq 2\sqrt{\log n \log \delta^{-1}}$ and that $\log \delta^{-1} \cdot C^2 \geq \log \delta^{-1}$, both holding since $C \geq 1$. Thus, our bound from Theorem 2 for all ranges of parameters asymptotically improves the one in (1).

Note that the compared bounds are with respect to worst-case strategy lengths. Our bounds can be made *in expectation* smaller by a factor of roughly $(1-\delta)$ using the same techniques as in [5] and [19].

[2] $ab \leq \frac{a^p}{p} + \frac{b^q}{q}$ for $1/p + 1/q = 1$ and $a, b \geq 0$, from which follows that for $0 < A \leq B$ it holds $\sqrt{AB} = \mathcal{O}(A/\log A) + \mathcal{O}(B \log B)$. Thus, if $\log n \leq \log \delta^{-1}$, then we bound the term $\mathcal{O}(\sqrt{\log n \log \delta^{-1}})$ by $\mathcal{O}(\log n/\log\log n) + \mathcal{O}(\log \delta^{-1} \log\log \delta^{-1})$, and otherwise by the term $\mathcal{O}(\log \delta^{-1})$.

1.2 Our Contribution – Simplified Algorithmic Techniques

The crucial underlying idea behind the algorithm from [19] that reaches the query complexity in (1) is as follows. The algorithm maintains a weight function μ for the vertex set of the input graph $G = (V, E)$ so that, at any given time, $\mu(v)$ represents the likelihood that v is the target. Initially, all vertices have the same weight. For a given μ, define a *potential* of a vertex v to be $\Phi_\mu(v) = \sum_{u \in V} \mu(u)d(v, u)$, where $d(u, v)$ is the distance between the vertices u and v in G. A vertex q that minimizes this potential function is called a *weighted median*, or a *median* for short, $q = \arg\min_{v \in V} \Phi_\mu(v)$. The vertex to be queried in each iteration of the algorithm is a median (ties are broken arbitrarily). After each query, the weights are updated: the weight of each vertex that is compatible with the reply is multiplied by p, and the weights of the remaining vertices are multiplied by $1 - p$.

The above scheme for querying subsequent vertices is the main building block of the algorithm that reaches the query complexity in (1). However, the analysis of the algorithm reveals a problematic case, namely the vertices that account for at least half of the total weight, call them *heavy*. On one side, such vertices are good candidates to include the target, so they are 'removed' from the graph to be investigated later. However, the need to investigate them in this separate way leads to an algorithm that has three phases, where the first two end by trimming the graph by leaving only the heavy vertices for the next phase. The first two phases are sequences of vertex queries performed on a median. The last phase uses yet a different majority technique. The duration of each of the first two phases are dictated by complicated formulas, which makes the algorithm difficult to analyze and understand.

We propose a simpler algorithm than the one in [19]. In each step, we simply query a median until just one candidate target vertex remains. Our improvement lies in a refined analysis in how such a query technique updates the weights, which has several advantages. It not only leads to a better query complexity but also provides a much simpler proof. Moreover, it results in a better understanding as how querying a median works in general graphs. We point out that this technique is quite general: it can be successfully applied to other query models – the details can be found in the appendix.

1.3 Related Work

Regarding the problem of searching in graphs without errors, many papers have been devoted to trees, mainly because it is a structure that naturally generalizes paths, which represents the classical binary search (see e.g. [27] for search in a path with non-uniform query times). This query model in case of trees is equivalent to several other problems, including vertex ranking [15] or tree-depth [33]. There exist linear-time algorithms for finding optimal query strategies [34, 39]. A lot of effort has been done to understand the complexity for trees with non-uniform query times. It turns out that the problem becomes hard for trees [17, 16]. Also refer the reader to works on a closely related query game with edge queries [10, 11, 14, 28, 31]. For general graphs, a strategy that always queries a 1-median (the minimizer of the sum of distances over all vertices) has length at most $\log_2 n$ [19].

To shift our attention to searching in graphs with errors, two works have been recently published on probabilistic models [19, 18]. These models are further generalized in [12] by considering the case of identifying two targets t_1 and t_2, where each answer to a query gives an edge on a shortest path to t_1 with probability p_1 or to t_2 with probability $p_2 = 1 - p_1$, respectively. Furthermore, there exists a closely related model in which the search is restricted in such a way, that each query performed to a vertex v must be followed by a vertex query to one of its neighbors – see [7, 21, 23, 22, 26] – in this context errors are usually referred to as unreliable advice.

An extensive amount of work has been devoted to searching problems in the presence of lies in a non-graph-theoretic context. The main tool of analysis is the concept of *volume* introduced by Berlekamp [6] – see also [9, 13] for a more detailed descriptions. We skip references to very numerous works that deal with fixed number of lies, pointing to surveys in [9, 13, 35]. For general queries, it is known [37] that a strategy of length $\log n + L \log_2 \log_2 n + \mathcal{O}(L \log L)$ exists, where n is the size of the search space and L is an upper bound on the number of lies. An almost optimal approximation strategy can be found in [32], which is actually given for a more general model of q-ary queries. For the most relevant model in our context, the probabilistic model, we remark on the early works, which bound strategy lengths to $\mathcal{O}(\frac{1}{\mathrm{poly}(\varepsilon)} \log n \log \delta^{-1})$, where $p < \frac{1}{2}$ and $\varepsilon = \frac{1}{2} - p$, with confidence probability $1 - \delta$ [2, 8]. A strategy of length $\mathcal{O}(\varepsilon^{-2}(\log n + \log \delta^{-1}))$ is given in [20]. Finally, [5] gives the best known bound of $\frac{1}{1-H(p)}(\log_2 n + \mathcal{O}(\log \log n) + \mathcal{O}(\log \delta^{-1}))$. We note that we arrive at a strategy matching asymptotically the complexity of [20] as a by-product from our graph-theoretic analysis (presented in the appendix).

2 Preliminaries

We now introduce the notation regarding the dynamics of the game. We assume an input graph with non-uniform edge lengths, and we denote said lengths by $\omega(e)$. We denote by $d(u, v)$ the *distance* between two vertices u and v, which is the length of a shortest path in G between u and v. We first focus on a simplified error model where the Responder is allowed a fixed number of lies, with the upper bound denoted as L. During the game, the Questioner keeps track of a *lie counter* ℓ_v for each vertex v of G. The value of ℓ_v equals the number of lies that must have already occurred assuming that v is actually the target v^*. The Questioner will utilize a constant $\Gamma > 1$ that will be fixed later. The goal of having this parameter is that we can tune it in order to obtain the right asymptotics. We define a *weight* $\mu_t(v)$ of a vertex v at the end of a round $t > 0$:

$$\mu_t(v) = \mu_0(v) \cdot \Gamma^{-\ell_v},$$

where $\mu_0(v)$ is the initial weight of v. For subsets $U \subseteq V$, let $\mu(U) = \sum_{v \in U} \mu(v)$. For brevity we write μ_t in place of $\mu_t(V)$. For a queried vertex q and an answer v, a vertex u is *compatible* with the answer if $u = v$ when $q = v$, or v lies on a shortest path from q to u.

As soon as there is only one vertex v left with $\ell_v \leq L$, the Questioner can successfully detect the target, $v^* = v$. We will set the initial weight of each vertex v to be $\mu_0(v) = 1$. Thus, $\mu_0 = n$ and $\mu_T \geq \Gamma^{-L}$ if the strategy length is T.

Based on the weight function μ, we define a *potential* of a vertex v:

$$\Phi(v) = \sum_{u \in V} \mu(u) \cdot d(v, u).$$

We write $\Phi_t(v)$ to refer to the value of a potential at the end of round t. Any vertex $x \in V$ minimizing $\Phi(x)$ is called 1-*median*.

Denote for an edge $\{v, u\}$, $N(v, u) = \{x \mid d(u, x) + \omega(\{v, u\}) = d(v, x)\}$ to be the set of all vertices to which some shortest path from v leads through u. Thus, $N(v, u)$ consists of the compatible vertices for the answer u when v has been queried. For any $S \subseteq V$, we write for brevity $\overline{S} = V \setminus S$, and for singletons $\overline{\{v\}}$ we further shorten to $\overline{v}$. We say that a vertex v is α-*heavy*, for some $0 \leq \alpha \leq 1$, if $\mu(v) > \alpha \cdot \mu(V)$. For a queried vertex q, if the answer is q, then such a reply is called a *yes-answer*; otherwise it is called a *no-answer*.

Algorithm VERTEX: Vertex queries for a fixed number of L lies.

```
for v ∈ V do
    μ(v) = 1
    ℓ_v = 0
while more than one vertex x ∈ V has ℓ_x ≤ L do
    q = arg min_{x∈V} Φ(x)
    query the vertex q
    for all nodes u not compatible with the answer do
        ℓ_u = ℓ_u + 1
        μ(u) = μ(u)/Γ
return the only x such that ℓ_x ≤ L
```

3 Vertex Searching

We now formally state the search strategy for a fixed number of lies – see Algorithm VERTEX. We combine our weight together with the idea of querying a 1-median [19]. As announced earlier, it turns out that our bound together with an appropriately selected weight function are strong enough so that we do not need the additional stages enhanced with a majority selection used in [19] in order to gain asymptotic improvements. We also note that we can easily introduce technical modifications to this strategy by changing the initial weight, the value of Γ or the stopping condition. We will do this to conclude several results for various error models (see the appendix).

3.1 Analysis of the Strategy

In this subsection we prove the following main technical contribution.

▶ **Theorem 3.** *Algorithm VERTEX finds the target in at most*

$$\frac{1}{\log_2(2\Gamma/(\Gamma+1))}\log_2 n + \frac{\log_2 \Gamma}{\log_2(2\Gamma/(\Gamma+1))}\cdot L$$

vertex queries.

Note that, due to the values of the initial and the final weight, it is enough to argue that the weight decreases on average, i.e., in an amortized way, by a factor of $(\Gamma+1)/(2\Gamma)$ per round. We first handle two cases (see Lemmas 4 and 5) when the weight decreases appropriately after a single query. These cases are a no-answer, and a yes-answer but only when the queried vertex is not 1/2-heavy. In the remaining case, i.e., when the queried vertex q is 1/2 heavy, it is not necessarily true that the weight decreases by the desired factor – this particularly happens in case of a yes-answer to such a query. This case is handled by the amortized analysis: we pair such yes-answers with no-answers to the query on q and show that in each such pair the weight decreases appropriately.

▶ **Lemma 4.** *If Algorithm VERTEX receives a no-answer in a round $t+1$, then $\mu_{t+1} \leq \frac{\Gamma+1}{2\Gamma}\mu_t$.*

Proof. Let q be the vertex queried in round $t+1$. Assume that the reply is some neighbor v of q. By [19], Lemma 4, we get that $\mu_t(N(q,v)) \leq \mu_t/2$. Moreover, because the lie counter increases by one for all vertices in $\overline{N(q,v)}$ and does not change for all vertices in $N(q,v)$ in round $t+1$, it follows that $\mu_{t+1} = \mu_t(N(q,v)) + \frac{1}{\Gamma}\mu_t(\overline{N(q,v)}) \leq \frac{\Gamma+1}{2\Gamma}\mu_t$. ◀

▶ **Lemma 5.** *Suppose that Algorithm VERTEX queries in round $t+1$ a vertex q that is not $1/2$-heavy. If a yes-answer is received, then $\mu_{t+1} \leq \frac{\Gamma+1}{2\Gamma}\mu_t$.*

Proof. The lie counter increments for each vertex of G except for q and remains the same for q in round $t+1$: $\mu_{t+1}(q) = \mu_t(q)$ and $\mu_{t+1}(\overline{q}) = \frac{1}{\Gamma}\mu_t(\overline{q})$. Since q is not $1/2$-heavy at the beginning of round $t+1$, $\mu_t(q) \leq \mu_t/2$. Thus, we get $\mu_{t+1} = \mu_t(q) + \frac{1}{\Gamma}\mu_t(\overline{q}) \leq \frac{\Gamma+1}{2\Gamma}\mu_t$. ◀

Now we turn to the proof of Theorem 3. Consider a maximal interval $[t_1, t_2]$, where $t_1 \leq t_2$ are integers, such that there exists a vertex q that is $1/2$-heavy in each round $t_1, \ldots, t_2$, and q is not $1/2$-heavy in round t_2+1. Call it a *q-interval*. Note that $t_1 > 0$ and q is not $1/2$-heavy in round $t_1 - 1$. We permute the replies given by the Responder in the q-interval to obtain a *new* sequence of replies as follows. The replies in rounds $1, \ldots, t_1 - 1$ and $t_2 + 1$ onwards are the same in both sequences. Note that in the interval $[t_1, t_2]$ the number of yes-answers, denote it by p, is smaller than or equal to the number of no-answers. Reorder the replies in the q-interval so that the yes-answers occur in rounds $t_1 + 2i$ for each $i \in \{0, \ldots, p-1\}$. In other words, we pair the yes-answers with no-answers so that a yes-answer in round $t_1 + 2i$ is paired with a no-answer in round $t_1 + 2i + 1$; we call such two rounds a *pair*. Following the pairs, some remaining, if any, no-answers follow in rounds $t_1 + 2p, \ldots, t_2$. Perform this transformation as long as a q-interval exists for some $q \in V$. Denote by μ' the weight of the new sequence.

Denote by t', if it exists, the minimum integer such that for some vertex v and for each $t > t'$, v is $1/2$-heavy at the end of the round t. If no such t' exists, then let t' be defined to be the number of rounds of the strategy.

We first analyze what happens, in the new sequence, in rounds i and $i+1$ that are a pair in an arbitrary q-interval for some vertex q. After such two rounds the lie counter for q increases by one, and the lie counter of any other vertex increases by at least one. This in particular implies that q is a 1-median throughout the entire q-interval in the new sequence. Moreover, the two replies in these rounds result in weight decrease by a factor of at least Γ, $\mu'_{i+1} \leq \mu'_{i-1}/\Gamma$. Since $\frac{1}{\Gamma} < (\frac{1+\Gamma}{2\Gamma})^2$, the overall progress after the pair is as required.

We now prove that for each $t \in \{0, \ldots, t'-1\}$ that does not belong to any pair it holds

$$\mu'_{t+1} \leq \frac{\Gamma+1}{2\Gamma}\mu'_t. \tag{2}$$

Recall that for each $t \leq t'$ that does not belong to any q-interval, $\mu'_t(v) = \mu_t(v)$ for each $v \in V$. If the answer to this query is a no-answer, then (2) follows from Lemma 4. Lemma 4 also applies to no-answers of a q-interval that do not belong to any pair since, as argued above, q is a 1-median throughout the q-interval. If the answer is a yes-answer, then since the queried vertex q is not $1/2$-heavy due to the choice of q-intervals, Inequality (2) follows from Lemma 5.

If t' is the last round in the original search strategy, then the proof is completed. Otherwise, consider the suffix of the original sequence of replies, consisting of rounds t for $t > t'$. In all these rounds, by definition, some vertex q is $1/2$-heavy. Also by definition, both sequences μ and μ' are identical in this suffix. One can check that if a vertex is heavy at the end of some round, then in the subsequent round Algorithm VERTEX does query this vertex. Thus, the vertex q is queried in all rounds of the suffix, and hence q is the target. Thus, it is enough to observe how the weight decreases on $\overline{q}$ in case of a yes-answer in a round $t > t'$: $\mu'_t(\overline{q}) = \mu'_{t-1}(\overline{q})/\Gamma \leq \frac{\Gamma+1}{2\Gamma}\mu'_{t-1}(\overline{q})$. This completes the proof of Theorem 3.

3.2 Proof of Theorem 1

Proof. We turn our attention to the model with a rate of lies bounded by a fraction $r < 1/2$ (linearly bounded error model). Our result, Theorem 1, is obtained on the basis of Algorithm VERTEX and the precise bound from Theorem 3. In particular, we run Algorithm VERTEX with $\Gamma = \frac{1-r}{r}$ and with a fixed bound on number of lies $L = \frac{\log_2 n}{1-H(r)} r$. By Theorem 3, Algorithm VERTEX asks then at most $\frac{\log_2 n}{\log_2(2\cdot(1-r))} + \frac{\log_2 \frac{1-r}{r}}{\log_2(2\cdot(1-r))} \cdot L = \frac{\log_2 n}{1-H(r)} \cdot \frac{1-H(r)+r\log_2 \frac{1-r}{r}}{1+\log_2(1-r)} = \frac{\log_2 n}{1-H(r)} = L/r$ queries. This bound concludes the proof, since the number of lies is within r fraction of strategy length. ◀

3.3 Proof of Theorem 2

Proof. Let $\varepsilon > 0$ be such that $p = \frac{1}{2}(1-\varepsilon)$. We run the strategy from Theorem 1 with an error rate $r = \frac{1}{2}(1-\varepsilon_0)$, where $\varepsilon_0 = \varepsilon / \left(1 + \sqrt{8 \ln \delta^{-1} / \ln n}\right)$. By Theorem 1 the strategy length is $Q = \frac{\log_2 n}{1-H(r)}$ which is (up to lower-order terms) $2\varepsilon_0^{-2} \ln n$, thus at least $\varepsilon_0^{-2} \ln n$ for n large enough. The expected number of lies is $\mathbb{E}[L] = p \cdot Q$ and by the Chernoff bound,

$$\begin{aligned}\Pr[Q - L \le (1-r)\cdot Q] &\le \exp\left(-\frac{1}{2}\left(1 - \frac{1-r}{1-p}\right)^2 \cdot (1-p) \cdot \frac{\ln n}{\varepsilon_0^2}\right) \\ &\le \exp\left(-\frac{1}{8}\left(\frac{\varepsilon - \varepsilon_0}{\varepsilon_0}\right)^2 \ln n\right) = \delta.\end{aligned}$$

The bound $Q = \frac{\log_2 n}{1-H(p)}(1 + \mathcal{O}(\sqrt{\ln \delta^{-1} / \ln n}) + \mathcal{O}(\ln \delta^{-1} / \ln n))$ follows then from $1 - H(x) \sim (1/2 - x)^2$. ◀

4 Conclusions

We note that also other query models have been studied in the graph-theoretic context, including edge queries. In an *edge query*, the Questioner points to an edge and the Responder tells which endpoint of that edge is closer to the target, breaking ties arbitrarily. It turns out that edge queries are more challenging to analyze, i.e., our technique for vertex queries does not transfer without changes. This is mostly due to a possible lack of edges that subdivide the search space equally enough. This issue can be patched by treating heavy vertices in a separate way. We provide a strategy of query complexity $\mathcal{O}(\frac{1}{\varepsilon^2}\Delta \log \Delta(\log n + \log \delta^{-1}))$. This generalizes the noisy binary search of [20] to general graphs, and has the advantage of being a weight-based strategy.

We additionally show the generalizations of our strategies to searching in unbounded domains, where one is concerned in searching e.g., the space of all positive integers with comparison queries. The goal is to minimize the number of queries as a function of N, the (unknown) position of the target. By adjusting the initial distribution of the weight to decay at polynomial rate with respect to the distance from the point 0, we almost automatically get desired solutions, e.g., a strategy of query complexity $\mathcal{O}(\frac{1}{\varepsilon^2}(\log N + \log \delta^{-1}))$ for searching in the probabilistic error model, improving upon $\mathcal{O}(\text{poly}(\varepsilon^{-1}) \log N \log \delta^{-1})$ of [2].

References

1 Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, 1987. doi:10.1007/BF00116828.

2 Javed A Aslam. *Noise tolerant algorithms for learning and searching*. PhD thesis, Massachusetts Institute of Technology, 1995.

3 Pranjal Awasthi, Maria-Florina Balcan, and Konstantin Voevodski. Local algorithms for interactive clustering. *Journal of Machine Learning Research*, 18:3:1–3:35, 2017. URL: http://jmlr.org/papers/v18/15-085.html.

4 Maria-Florina Balcan and Avrim Blum. Clustering with Interactive Feedback. In *Algorithmic Learning Theory, 19th International Conference, ALT 2008, Budapest, Hungary, October 13-16, 2008. Proceedings*, pages 316–328, 2008. doi:10.1007/978-3-540-87987-9_27.

5 Michael Ben-Or and Avinatan Hassidim. The Bayesian Learner is Optimal for Noisy Binary Search (and Pretty Good for Quantum as Well). In *FOCS*, pages 221–230, 2008. doi:10.1109/FOCS.2008.58.

6 Elvyn R. Berlekamp. *Block Coding For The Binary Symmetric Channel With Noiseless, Delayless Feedback*, pages 61–88. Wiley & Sons, New York, 1968.

7 Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching a Tree with Permanently Noisy Advice. *CoRR*, abs/1611.01403, 2016. arXiv:1611.01403.

8 Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *STOC*, pages 130–136, 1993. doi:10.1145/167088.167129.

9 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms: Reliable Computation with Unreliable Information*. Springer Publishing Company, Incorporated, 2013.

10 Ferdinando Cicalese, Tobias Jacobs, Eduardo Sany Laber, and Caio Dias Valentim. The binary identification problem for weighted trees. *Theor. Comput. Sci.*, 459:100–112, 2012. doi:10.1016/j.tcs.2012.06.023.

11 Ferdinando Cicalese, Balázs Keszegh, Bernard Lidický, Dömötör Pálvölgyi, and Tomás Valla. On the tree search problem with non-uniform costs. *Theor. Comput. Sci.*, 647:22–32, 2016. doi:10.1016/j.tcs.2016.07.019.

12 Argyrios Deligkas, George B. Mertzios, and Paul G. Spirakis. Binary Search in Graphs Revisited. In *MFCS*, pages 20:1–20:14, 2017. doi:10.4230/LIPIcs.MFCS.2017.20.

13 Christian Deppe. *Coding with Feedback and Searching with Lies*, pages 27–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-32777-6_2.

14 Dariusz Dereniowski. Edge ranking of weighted trees. *Discrete Applied Mathematics*, 154(8):1198–1209, 2006. doi:10.1016/j.dam.2005.11.005.

15 Dariusz Dereniowski. Edge ranking and searching in partial orders. *Discrete Applied Mathematics*, 156(13):2493–2500, 2008. doi:10.1016/j.dam.2008.03.007.

16 Dariusz Dereniowski, Adrian Kosowski, Przemyslaw Uznański, and Mengchuan Zou. Approximation Strategies for Generalized Binary Search in Weighted Trees. In *ICALP*, pages 84:1–84:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.84.

17 Dariusz Dereniowski and Adam Nadolski. Vertex rankings of chordal graphs and weighted trees. *Inf. Process. Lett.*, 98(3):96–100, 2006. doi:10.1016/j.ipl.2005.12.006.

18 Ehsan Emamjomeh-Zadeh and David Kempe. A General Framework for Robust Interactive Learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 7085–7094, 2017. URL: http://papers.nips.cc/paper/7283-a-general-framework-for-robust-interactive-learning.

19 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *STOC*, pages 519–532, 2016. doi:10.1145/2897518.2897656.

20 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with Noisy Information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

21 Nicolas Hanusse, David Ilcinkas, Adrian Kosowski, and Nicolas Nisse. Locating a target with an agent guided by unreliable local advice: how to beat the random walk when you have a clock? In *PODC*, pages 355–364, 2010. doi:10.1145/1835698.1835781.

22 Nicolas Hanusse, Dimitris J. Kavvadias, Evangelos Kranakis, and Danny Krizanc. Memoryless search algorithms in a network with faulty advice. *Theor. Comput. Sci.*, 402(2-3):190–198, 2008. doi:10.1016/j.tcs.2008.04.034.

23 Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004. doi:10.1016/S0166-218X(03)00189-6.

24 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283478.

25 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

26 Evangelos Kranakis and Danny Krizanc. Searching with Uncertainty. In *SIROCCO'99, 6th International Colloquium on Structural Information & Communication Complexity, Lacanau-Ocean, France, 1-3 July, 1999*, pages 194–203, 1999.

27 Eduardo Sany Laber, Ruy Luiz Milidiú, and Artur Alves Pessoa. On binary searching with non-uniform costs. In *SODA*, pages 855–864, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365796.

28 Tak Wah Lam and Fung Ling Yue. Optimal Edge Ranking of Trees in Linear Time. *Algorithmica*, 30(1):12–33, 2001. doi:10.1007/s004530010076.

29 Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2(4):285–318, 1987. doi:10.1007/BF00116827.

30 Tie-Yan Liu. *Learning to Rank for Information Retrieval.* Springer, 2011. doi:10.1007/978-3-642-14267-3.

31 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In *SODA*, pages 1096–1105, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347202.

32 S. Muthukrishnan. On Optimal Strategies for Searching in Presence of Errors. In *SODA*, pages 680–689, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314672.

33 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

34 Krzysztof Onak and Pawel Parys. Generalization of Binary Search: Searching in Trees and Forest-Like Partial Orders. In *FOCS*, pages 379–388, 2006. doi:10.1109/FOCS.2006.32.

35 Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoretical Computer Science*, 270(1):71–109, 2002. doi:10.1016/S0304-3975(01)00303-6.

36 Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 239–248, 2005. doi:10.1145/1081870.1081899.

37 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.

38 Alfréd Rényi. On a problem of information theory. *MTA Mat. Kut. Int. Kozl.*, 6B:505–516, 1961.

39 Alejandro A. Schäffer. Optimal Node Ranking of Trees in Linear Time. *Inf. Process. Lett.*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

40 Burr Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. doi:10.2200/S00429ED1V01Y201207AIM018.

41 Stanislaw M. Ulam. *Adventures of a Mathematician*. Scribner, New York, 1976.

A Analysis of the Generic Strategies for Edge Queries

We recall a different format of queries called *edge queries*, where in each round the Questioner selects an edge $\{u, v\}$ of an input graph and the Responder replies with the endpoint of $\{u, v\}$ that is closer to the target. Again, ties are broken arbitrarily. The edge-query model naturally generalizes comparison queries in linearly or partially ordered data. In case of edge-queries we consider graphs with unit edge lengths.

We start by giving the notation regarding edge queries. The *degree* of a vertex v, denoted by $\deg(v)$, is the number of its neighbors in G. We denote by $\Delta = \max_{v \in V} \deg(v)$ the maximum degree of G. We define an edge-vertex distance $d(e, v) = \min(d(x, v), d(y, v))$ for an edge $e = \{x, y\}$. Similarly as for vertex queries, based on a weight function μ and distance d, we define a potential of an edge e:

$$\Phi(e) = \sum_{u \in V} \mu(u) \cdot d(e, u).$$

Again, we write $\Phi_t(e)$ to refer to this value at the end of round t. Any edge e minimizing $\Phi(e)$ is called 1-*edge-median*. For an edge $e = \{u, v\}$ and one of its endpoints,

$$N(e, v) = \{w \mid d(v, w) \leq d(u, w)\}, \qquad N_<(e, v) = \{w \mid d(v, w) < d(u, w)\}.$$

For edge-queries we give a strategy that is a bit more complicated – see Algorithm EDGE. Intuitively, as opposed to the vertex-query case, there may be no edges in the graph that 'subdivide' the search space evenly enough. This already happens as soon as one of the vertices is $\frac{1}{\Delta+1}$-heavy. If this is the case, and say vertex v is $\frac{1}{\Delta+1}$-heavy, we cyclically query edges incident to v in an appropriate greedy order. We continue to do so until all other vertices have been eliminated, and hence v must be the target, or v is no longer $\frac{1}{\Delta+1}$-heavy. If none of the vertices is $\frac{1}{\Delta+1}$-heavy, we simply query a 1-edge-median. The absence of such heavy vertices essentially ensures, that this decreases the weight sufficiently.

This results in a more involved proof given in Section 3.1. Similarly as for vertex queries, we also first provide an analysis for a fixed number of lies (see Theorem 6) and then from this bound we derive appropriate bounds for other models (Theorems 7 and 8).

▶ **Theorem 6.** *Let $\Gamma > 1$. Algorithm EDGE finds the target in at most $\frac{\log n + L \log \Gamma}{\log(1 + \frac{\Gamma - 1}{\Gamma \Delta + 1})}$ edge queries.*

▶ **Theorem 7.** *In the linearly bounded error model with error rate $r = \frac{1}{\Delta+1}(1 - \varepsilon)$ for some $0 < \varepsilon \leq 1$, the target can be found in at most $2\varepsilon^{-2}\Delta \ln n$ edge queries.*

▶ **Theorem 8.** *In the probabilistic error model with error rate $p = \frac{1}{2}(1 - \varepsilon)$ for some $0 < \varepsilon \leq 1$ there exists a strategy that finds the target using at most $\mathcal{O}(\varepsilon^{-2}\Delta \log \Delta \cdot (\log n + \log \delta^{-1}))$ edge queries, correctly with probability at least $1 - \delta$.*

Algorithm EDGE: Edge queries for fixed number of L lies.

for $v \in V$ **do**
  $\mu(v) = 1$
  $\ell_v = 0$
while more than one vertex x satisfies $\ell_x \leq L$ **do**
  if there exists v such that $\mu(v) > \mu/(\Delta+1)$ **then** ▷ v is $\frac{1}{\Delta+1}$-heavy
    for $i = 1$ **to** $\deg(v)$ **do** ▷ greedy ordering of neighbors
      select an edge e_i incident to v to maximize $\mu(\bigcup_{j\leq i} N_<(e_j, v))$
    $i = 1$
    do ▷ cyclically query edges incident to v
      query e_i
      for all nodes u not compatible with the answer **do**
        $\ell_u = \ell_u + 1$
        $\mu(u) = \mu(u)/\Gamma$
      if the answer to the last query is v **then**
        $i = (i+1) \bmod \deg(v)$
    while $\mu(v) > \mu/(\Delta+1)$ and there exists more than one x with $\ell_x \leq L$
  else
    $e = \arg\min_{x\in E} \Phi(x)$
    query e
    for all nodes u not compatible with the answer **do**
      $\ell_u = \ell_u + 1$
      $\mu(u) = \mu(u)/\Gamma$
return v such that $\ell_v \leq L$

Proof of Theorem 6

We first prove two technical lemmas and then we give the proof of the theorem.

▶ **Lemma 9.** *Let $\Gamma > 1$. Suppose that Algorithm EDGE queries in round $t+1$ an edge e_q incident to a vertex q such that $e_q = \arg\min_{x\in E} \Phi_t(x)$. If $\deg(q) > 1$, then*

$$\mu_t(\overline{N(e_q, q)}) \geq \frac{1}{\deg(q)}(\mu_t - \mu_t(q)). \tag{3}$$

Proof. Denote $e_q = \{q, v\}$. For each neighbor w of q define

$$N_w^{\cap} = N(e_q, q) \cap N_<(\{q, w\}, w).$$

Consider an edge $e' = \{q, w\}$ that maximizes $\mu_t(N_w^{\cap})$. If X is the set of neighbors of q, then by definition and by the fact that e_q lies on no shortest path from q to any vertex in $N_<(e_q, v)$, i.e., $N_v^{\cap} = \emptyset$, it holds

$$N(e_q, q) \setminus \{q\} \subseteq \bigcup_{w'\in X} N_{w'}^{\cap} = \bigcup_{w'\in X\setminus\{v\}} N_{w'}^{\cap}.$$

Hence (since e' maximizes $\mu_t(N_w^{\cap})$) we obtain that

$$\mu_t(N_w^{\cap}) \geq \frac{1}{\deg(q)-1}(\mu_t(N(e_q, q)) - \mu_t(q)). \tag{4}$$

For brevity, we extend our notation in the following way: for an edge e and a subset S of vertices, $\Phi_t(e,S) = \sum_{z\in S}\mu_t(z)\cdot d(e,z)$. Note that for any $S\subseteq V$ and any edge e, $\Phi_t(e) = \Phi_t(e,S)+\Phi_t(e,\overline{S})$. We obtain

$$\begin{aligned}\Phi_t(e', N(e_q,q)) &= \Phi_t(e', N_w^{\cap}) + \Phi_t(e', N(e_q,q)\setminus N_w^{\cap})\\ &= \sum_{u\in N_w^{\cap}}\mu_t(u)\cdot(d(q,u)-1) + \sum_{u\in N(e_q,q)\setminus N_w^{\cap}}\mu_t(u)\cdot d(q,u)\\ &= \sum_{u\in N(e_q,q)}\mu_t(u)\cdot d(q,u) - \mu_t(N_w^{\cap})\\ &\le \Phi_t(e_q, N(e_q,q)) - \frac{1}{\deg(q)-1}(\mu_t(N(e_q,q))-\mu_t(q)),\end{aligned} \tag{5}$$

where the last inequality is due to (4). For any vertex u, $d(e',u)\le d(e_q,u)+1$ because e_q and e' are adjacent. Using this fact we obtain:

$$\begin{aligned}\Phi_t(e', \overline{N(e_q,q)}) &= \sum_{u\notin N(e_q,q)}\mu_t(u)\cdot d(e',u)\\ &\le \sum_{u\notin N(e_q,q)}\mu_t(u)\cdot d(e_q,u) + \sum_{u\notin N(e_q,q)}\mu_t(u)\\ &= \Phi_t(e_q,\overline{N(e_q,q)}) + \mu_t(\overline{N(e_q,q)}).\end{aligned} \tag{6}$$

Finally, by (5) and (6) we get:

$$\begin{aligned}\Phi_t(e') &= \Phi_t(e', N(e_q,q)) + \Phi_t(e',\overline{N(e_q,q)})\\ &\le \Phi_t(e_q, N(e_q,q)) - \frac{\mu_t(N(e_q,q))-\mu_t(q)}{\deg(q)-1} + \Phi_t(e_q,\overline{N(e_q,q)}) + \mu_t(\overline{N(e_q,q)})\\ &= \Phi_t(e_q) + \mu_t(\overline{N(e_q,q)}) - \frac{1}{\deg(q)-1}(\mu_t(N(e_q,q))-\mu_t(q)).\end{aligned}$$

By assumption, $\Phi_t(e_q)\le\Phi_t(e')$. Therefore,

$$\frac{1}{\deg(q)-1}(\mu_t(N(e_q,q))-\mu_t(q)) \le \mu_t(\overline{N(e_q,q)}),$$

which can be rewritten as in (3). ◀

▶ **Lemma 10.** *Let $\Gamma>1$. Suppose that Algorithm EDGE queries in round $t+1$ an edge incident to a vertex q that is not $\frac{1}{\Delta+1}$-heavy in this round, and the answer is q. Then, $\mu_{t+1}\le(1-\frac{\Gamma-1}{\Gamma(\Delta+1)})\mu_t$.*

Proof. Let $e_q=\{q,v\}$ be the edge queried in round $t+1$. Suppose first that $\deg(q)>1$. By Lemma 9,

$$\mu_t(\overline{N(e_q,q)}) \ge \frac{1}{\deg(q)}(\mu_t-\mu_t(q)) \ge \frac{1}{\Delta}(\mu_t-\mu_t(q)). \tag{7}$$

Because e_q is the queried edge in round $t+1$ and the reply is q, the lie counter remains unchanged for the vertices in $N(e_q,q)$ and decreases by one in the complement $\overline{N(e_q,q)}$. Hence,

$$\mu_{t+1} = \mu_t(N(e_q,q)) + \frac{1}{\Gamma}\mu_t(\overline{N(e_q,q)}) = \mu_t - \frac{\Gamma-1}{\Gamma}\mu_t(\overline{N(e_q,q)}).$$

Thus, by (7) and by the fact that $\mu_t(q) \leq \frac{1}{\Delta+1}\mu_t$ for q that is not $\frac{1}{\Delta+1}$-heavy in round t,

$$\mu_{t+1} \leq \left(1 - \frac{\Gamma-1}{\Gamma\Delta} \cdot \frac{\Delta}{\Delta+1}\right)\mu_t,$$

which completes the proof in the case when $\deg(q) > 1$.

If $\deg(q) = 1$, then in round t the lie counter increases by one for each vertex in $\overline{q}$. Thus, again by the fact that q is not $\frac{1}{\Delta+1}$-heavy,

$$\mu_{t+1} = \mu_t(q) + \frac{1}{\Gamma}\mu_t(\overline{q}) \leq \left(\frac{1}{\Delta+1} + \frac{1}{\Gamma}\right)\mu_t \leq \left(1 - \frac{\Gamma-1}{\Gamma(\Delta+1)}\right)\mu_t. \quad \blacktriangleleft$$

Proof of Theorem 6. Having proved the technical lemmas, we now turn to the proof of Theorem 6. It is enough to argue that every query, amortized, multiplies the weight by a factor of $1 - \frac{\Gamma-1}{\Gamma(\Delta+1)} = 1/(1 + \frac{\Gamma-1}{\Gamma\Delta+1})$. If there is no $\frac{1}{\Delta+1}$-heavy vertex, then the theorem follows from Lemma 10. Hence suppose in the rest of the proof that there exists a $\frac{1}{\Delta+1}$-heavy vertex and denote this vertex by q.

For the amortized analysis, consider a sequence of t consecutive queries to edges $e_1, \ldots, e_t$, $t \leq \deg(q)$, performed while q is $\frac{1}{\Delta+1}$-heavy; call such a sequence a *segment*. Suppose this sequence starts in round t'. Denote $e_i = \{q, v_i\}$, $i \in \{1, \ldots, t\}$, and let

$$Q_1 = \bigcup_{i=1}^{t} N_<(e_i, v_i), \qquad Q_2 = V \setminus (Q_1 \cup \{q\}).$$

First we assume that the query in round $t' + t$ (i.e., the query that follows the sequence) does not return v as a reply, or v stops being $\frac{1}{\Delta+1}$-heavy. We argue, informally speaking, that this query in round $t' + t$ amortizes the t queries prior to it thanks to the assumption $t \leq \deg(q)$. Because the lie counter of q increments in round $t' + t$,

$$\mu_{t'+t}(q) \leq \frac{1}{\Gamma}\mu_{t'}(q). \tag{8}$$

We have $\mu_{t'+t}(Q_1) \leq \frac{1}{\Gamma}\mu_{t'}(Q_1)$ by the formulation of Algorithm EDGE, and $\mu_{t'+t}(Q_2) \leq \mu_{t'}(Q_2)$. Then, $Q_1 \cup Q_2 = \overline{q}$ and $Q_1 \cap Q_2 = \emptyset$ imply $\mu_{t'}(Q_1) \leq \mu_{t'}(\overline{q}) - \mu_{t'}(Q_2)$ and hence

$$\mu_{t'+t}(Q_1) + \mu_{t'+t}(Q_2) \leq \frac{1}{\Gamma}\mu_{t'}(\overline{q}) + \frac{\Gamma-1}{\Gamma}\mu_{t'}(Q_2). \tag{9}$$

Due to the order according to which the edges $\{q, v_i\}$ are queried, we have

$$\mu_{t'}(Q_2) \leq \left(1 - \frac{t}{\deg(q)}\right)\mu_{t'}(\overline{q}) \leq \left(1 - \frac{t}{\Delta}\right)\mu_{t'}(\overline{q}). \tag{10}$$

Note that $\mu_{t'}(\overline{q}) \leq \frac{\Delta}{\Delta+1}\mu_{t'}$ since by assumption q is $\frac{1}{\Delta+1}$-heavy in round t'. Since $\mu_{t'+t} = \mu_{t'+t}(q) + \mu_{t'+t}(Q_1) + \mu_{t'+t}(Q_2)$, we get by (8), (9) and (10):

$$\mu_{t'+t} \leq \left(\frac{1}{\Gamma} + \frac{\Gamma-1}{\Gamma}\frac{\Delta-t}{\Delta+1}\right)\mu_{t'} = \left(1 - \frac{\Gamma-1}{\Gamma}\frac{t+1}{(\Delta+1)}\right)\mu_{t'} \leq \left(1 - \frac{\Gamma-1}{\Gamma(\Delta+1)}\right)^{t+1}\mu_{t'},$$

where the last inequality comes from $(1-x)^k \geq 1 - xk$, for $k \geq 1$ and $x < 1$.

Consider now a maximal sequence S of rounds in which q is $\frac{1}{\Delta+1}$-heavy and is not $\frac{1}{\Delta+1}$-heavy in the round that follows the sequence. Note that Algorithm EDGE cyclically queries the edges incident to q in S. Let $r'_1 \leq \cdots \leq r'_{b'}$ be all rounds in S having q as an

answer. Denote $X = S \setminus \{r'_1, \ldots, r'_{b'}\}$, the set of rounds in S in which q is not an answer. Let $a = \lceil b'/\deg(q) \rceil$. The lie counter of each vertex in $\overline{q}$ increases by at least $a-1$ and by at most a times by executing S – we point out that this crucial property follows from the fact that the queries in the segment are applied to the edges incident to q consecutively modulo $\deg(v)$. Since q is $\frac{1}{\Delta+1}$-heavy at the beginning of S and is not $\frac{1}{\Delta+1}$-heavy right after S, the lie counter of q increases by at least a as a result of S. Hence, $|X| \geq a$. Partition $r'_1, \ldots, r'_{b'}$ into a minimum number of segments of length at most $\deg(q)$ each, which leads to at most a segments. Thus, we can pair these segments with rounds in X. For each such pair of at most $\deg(q)+1$ rounds we apply the amortized analysis performed above. Note that this approach is valid since the amortized analysis is insensitive of the order of appearance of the queries in X and the queries in $S \setminus X$.

Finally, suppose that there is a series of queries at the end of the strategy (a suffix) performed to edges incident to a $\frac{1}{\Delta+1}$-heavy vertex q such that all replies point to q and q remains $\frac{1}{\Delta+1}$-heavy till the end of the strategy. Note that in such a case q is the target. The vertex q had the uniquely smallest lie counter just before those queries. This in particular implies that the lie counter is strictly smaller than L. We artificially add a sequence of *pseudo-queries*, each of which increments the lie counter of q until it reaches L. This implies that the suffix of the search strategy now consists of a reply (which comes from a regular query or a pseudo-query) which does not point to q. Thus, we use again the arguments from our amortized analysis: we can find a segment and pair with it the above mentioned query pointing away from q. ◀

Proof of Theorem 7

Proof. Similarly as in the case of vertex queries, the generic strategy in Algorithm EDGE for edge queries and its corresponding bound for a fixed number of lies can be used to provide strong bounds for linearly bounded and probabilistic error models.

Let $\Gamma = 1 + \frac{\Delta+1}{\Delta}\frac{\varepsilon}{1-\varepsilon} = \frac{1-r}{r} \cdot \frac{1}{\Delta}$. Denote $Q_{\min} = \frac{\ln n}{\ln(1+\frac{\Gamma-1}{\Gamma\Delta+1}) - r\ln\Gamma}$. We run Algorithm EDGE with bound $L = Q_{\min} r$ and parameter Γ set as just mentioned above. Then, by Theorem 6, the length of the strategy is at most $\frac{1}{\ln(1+\frac{\Gamma-1}{\Gamma\Delta+1})} \cdot \ln n + \frac{\ln\Gamma}{\ln(1+\frac{\Gamma-1}{\Gamma\Delta+1})} \cdot Q_{\min} r = Q_{\min} = L/r$. To conclude the proof, we bound

$$Q_{\min} = \frac{\ln n}{F(\varepsilon)/(\Delta+1) + F(-\varepsilon/\Delta)\cdot\Delta/(\Delta+1)} =$$

(where $F(x) \stackrel{\text{def}}{=} x + (1-x)\ln(1-x) = \sum_{i=2}^{\infty} \frac{x^i}{i(i-1)}$)

$$= \frac{\ln n}{\sum_{i=2}^{\infty} \frac{\varepsilon^i}{i(i-1)} \frac{\Delta^{i-1}+(-1)^i}{(\Delta+1)\Delta^{i-1}}} \leq \frac{\ln n}{\varepsilon^2/(2\Delta)} = 2\varepsilon^{-2}\Delta\ln n.$$ ◀

Proof of Theorem 8

Proof. For edge queries, we use a two step approach: first, we repeatedly ask queries to boost their error rate from $\sim 1/2$ to below $1/(\Delta+1)$, and then use the linearly bounded error strategy.

As a first step, we show that for $p_0 = \frac{1}{\Delta+1}(1-\varepsilon_0)$, there exists a strategy that locates the target with high probability using $\mathcal{O}(\Delta \log n/\varepsilon_0^2)$ edge queries. Indeed, assume without loss of generality that $\varepsilon_0 < 1/2$. We fix $\varepsilon_1 = \varepsilon_0/(1+\sqrt{\frac{3}{2}\frac{\Delta+1}{\Delta}\ln\delta^{-1}/\ln n})$, and use Theorem 7

with error rate $r_0 = \frac{1}{\Delta+1}(1-\varepsilon_1)$. By Theorem 7, we obtain that the strategy length is $Q = 2\varepsilon_1^{-2}\Delta \ln n = \mathcal{O}(\Delta\varepsilon_0^{-2}(\log n + \log \delta^{-1}))$. The expected number of lies is $\mathbb{E}[L] = p_0 \cdot Q$ and by the Chernoff bound,

$$\Pr[L \geq r_0 \cdot Q] \leq \exp\left(-\frac{1}{3}\left(\frac{r_0}{p_0} - 1\right)^2 \cdot p_0 \cdot Q\right) \leq \exp\left(-\frac{2}{3}\left(\frac{\varepsilon_0 - \varepsilon_1}{\varepsilon_1}\right)^2 \cdot \frac{\Delta}{\Delta+1} \ln n\right) = \delta.$$

We now observe that to achieve the error rate of $\frac{1}{2}(1-\varepsilon)$, we can boost the query error rate to be smaller by repeating the same query multiple times and taking the majority answer. By repeating each query $P = \mathcal{O}(\log(2\Delta+2) \cdot \varepsilon^{-2})$ times, we get the correct answer with probability $1 - p' = 1 - \frac{1}{2} \cdot \frac{1}{\Delta+1}$, and as shown already, we only need $\mathcal{O}(\Delta(\log n + \log \delta^{-1}))$ queries with the error rate p' to locate the target with probability at least $1-\delta$. Thus the claimed bound follows. ◀

As an immediate corollary we obtain a very simple strategy for noisy binary search in an integer range of complexity $\mathcal{O}(\varepsilon^{-2}(\log n + \log \delta^{-1}))$ matching [20].

B Application: Searching Unbounded Integer Ranges

Building on our generic strategies, we now obtain a general technique for searching an unbounded domain $\mathbb{N} = \{1, 2, \ldots\}$ with comparison queries. Here the measure of complexity is the dependency on the error rate (number of lies) and on N, the (initially unknown) position of the target. The main idea is to use Algorithms VERTEX and EDGE, tweaking the initial weight distribution. We fix the *initial* weight of an integer n to be $\mu_0(n) = n^{-2}$. The total initial weight then equals $\pi^2/6 = \Theta(1)$. We provide the following bounds.[3]

▶ **Corollary 11.** *There exists a strategy that finds an integer in an unbounded integer range ($\mathbb{N}$) using at most*

- $\frac{\log \frac{\pi^2}{6} + 2\log N + L \log \Gamma}{\log \frac{2\Gamma}{\Gamma+1}}$ *ternary queries, or*
- $\frac{\log \frac{\pi^2}{6} + 2\log N + L \log \Gamma}{\log \frac{3\Gamma}{2\Gamma+1}}$ *binary (comparison) queries,*

where N is the target, L is an upper bound on the number of (adversarial) lies and $\Gamma > 1$ is an arbitrarily selected coefficient.

Proof. We use Algorithm VERTEX for ternary queries; let the strategy length be Q. By the proof of Theorem 3, $\mu_Q \leq (\frac{2\Gamma}{\Gamma+1})^Q \cdot \frac{\pi^2}{6}$. The final weight is at least $\mu_Q \geq N^{-2} \cdot \Gamma^{-L}$, and the bound for ternary queries follows since the number of queries is at most $\log(\frac{\pi^2/6}{N^{-2}\Gamma^{-L}}) / \log \frac{2\Gamma}{\Gamma+1}$.

The bound for binary queries is obtained analogously from Theorem 6 (note that $\Delta = 2$) since we apply Algorithm EDGE for binary queries. ◀

Simply setting $\Gamma = 2$ yields an $\mathcal{O}(\log N + L)$ length strategy with comparison queries on unbounded integer domains with a fixed number of L lies.

We need to restate the linearly bounded error model in the case of unbounded domains since the Responder does not know a priori the length of the strategy. We define this error model as follows: whenever the Questioner finds the target and thus declares the search to be completed after t rounds, it is guaranteed that at most rt lies have occurred throughout the search.

[3] We note that the term *ternary* refers to a model in which each query selects an integer i and as a response receives information whether the target is smaller than i, equals i, or is greater than i.

▶ **Corollary 12.** *For the linearly bounded error model with an error rate r and an unbounded integer domain, there exists a strategy that finds the target integer N in:*

- $\mathcal{O}(\varepsilon^{-2}\log N)$ *ternary queries when* $r = \frac{1}{2}(1-\varepsilon)$*, or*
- $\mathcal{O}(\varepsilon^{-2}\log N)$ *binary queries when* $r = \frac{1}{3}(1-\varepsilon)$*.*

Proof. Consider ternary queries. We proceed analogously as in the proof of Theorem 1. We have that the initial weight is $\pi^2/6$. Run Algorithm VERTEX until there is a single n such that $\ell_n \leq r \cdot Q$. Any Q such that $Q \geq \ln(\pi^2/6)/\ln\frac{2\Gamma}{\Gamma+1} + 2\ln N/\ln\frac{2\Gamma}{\Gamma+1} + L\ln\Gamma/\ln\frac{2\Gamma}{\Gamma+1}$ is an upper bound on the length of the strategy. We thus get an upper bound

$$Q \leq 2\varepsilon^{-2}(2\ln N + \mathcal{O}(1)).$$

The binary case follows in an analogous manner. ◀

▶ **Corollary 13.** *In the probabilistic error model, the target integer N can be found in an unbounded integer range using $\mathcal{O}(\varepsilon^{-2}(\log N + \log\delta^{-1}))$ binary queries for $p = \frac{1}{2}(1-\varepsilon)$, correctly with probability at least $1-\delta$.*

Proof. Same proof strategy as for Theorem 8, with $\Delta = 2$, applies. ◀

Selection from Heaps, Row-Sorted Matrices, and $X + Y$ Using Soft Heaps

Haim Kaplan[1]
Blavatnik School of Computer Science, Tel Aviv University, Israel
haimk@post.tau.ac.il

László Kozma[2]
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands
lkozma@gmail.com

Or Zamir[3]
Blavatnik School of Computer Science, Tel Aviv University, Israel
orzamir@mail.tau.ac.il

Uri Zwick[4]
Blavatnik School of Computer Science, Tel Aviv University, Israel
zwick@tau.ac.il

Abstract

We use *soft heaps* to obtain simpler optimal algorithms for selecting the k-th smallest item, and the set of k smallest items, from a heap-ordered tree, from a collection of sorted lists, and from $X + Y$, where X and Y are two unsorted sets. Our results match, and in some ways extend and improve, classical results of Frederickson (1993) and Frederickson and Johnson (1982). In particular, for selecting the k-th smallest item, or the set of k smallest items, from a collection of m sorted lists we obtain a new optimal *"output-sensitive"* algorithm that performs only $O(m + \sum_{i=1}^{m} \log(k_i + 1))$ comparisons, where k_i is the number of items of the i-th list that belong to the overall set of k smallest items.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases selection, soft heap

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.5

1 Introduction

The input to the standard *selection problem* is a set of n items, drawn from a totally ordered domain, and an integer $1 \leq k \leq n$. The goal is to return the k-th smallest item in the set. A classical result of Blum et al. [1] says that the selection problem can be solved deterministically in $O(n)$ time, i.e., faster than *sorting* the set. The number of comparisons required for selection was reduced by Schönhage et al. [26] to $3n$, and by Dor and Zwick [8, 9] to about $2.95n$.

[1] Research supported by the Israel Science Foundation grant no. 1841-14 and by a grant from the Len Blavatnik and the Blavatnik Family foundation.
[2] Research supported by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11) and ERC grant no. 617951.
[3] Research supported by a grant from the Len Blavatnik and the Blavatnik Family foundation. Work partly done during a research visit to Copenhagen supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.
[4] Work partly done during a research visit to Copenhagen supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

In the *generalized selection* problem, we are also given a *partial order* P known to hold for the set of n input items. The goal is again to return the k-th smallest item. The corresponding *generalized sorting* problem was extensively studied. It was shown by Kahn and Saks [20] that the problem can be solved using only $O(\log e(P))$ comparisons, where $e(P)$ is the number of *linear extensions* of P. Thus, the information-theoretic lower bound is tight for generalized sorting. The algorithm of Kahn and Saks [20] performs only $O(\log e(P))$ comparisons, but may spend much more time on deciding which comparisons to perform. Kahn and Kim [19] and Cardinal et al. [4] gave algorithms that perform only $O(\log e(P))$ comparisons and run in polynomial time.

A moment's reflection shows that an algorithm that finds the k-th smallest item of a set, must also identify the set of k smallest items of the set.[5] Given a partial order P, let $s_k(P)$ be the number of subsets of size k that may possibly be the set of k smallest items in P. Then, $\log_2 s_k(P)$ is clearly a lower bound on the number of comparisons required to select the k-th smallest item, or the set of k smallest items. Unlike sorting, this information-theoretic lower bound for selection may be extremely weak. For example, the information-theoretic lower bound for selecting the *minimum* is only $\log_2 n$, while $n - 1$ comparisons are clearly needed (and are sufficient). To date, there is no characterization of pairs (P, k) for which the information-theoretic lower bound for selection is tight, nor an alternative general technique to obtain a tight lower bound.

Frederickson and Johnson [12, 13, 14] and Frederickson [11] studied the generalized selection problem for some interesting specific partial orders. Frederickson [11] considered the case in which the input items are items of a binary *min-heap*, i.e., they are arranged in a binary tree, with each item smaller than its two children. Frederickson [11] gave a complicated algorithm that finds the k-th smallest item using only $O(k)$ comparisons, matching the information-theoretic lower bound for this case. (Each subtree of size k of the heap, containing the root, can correspond to the set of k smallest items, and there are $\frac{1}{k+1}\binom{2k}{k}$, the k-th *Catalan* number, such subtrees.)

Frederickson and Johnson [12, 13, 14] considered three other interesting special cases. (i) The input items are in a collection of sorted lists, or equivalently they reside in a row-sorted matrix; (ii) The input items reside in a collection of matrices, where each matrix is both row- and column-sorted; (iii) The input items are $X + Y$, where X and Y are unsorted sets of items.[6] For each of these cases, they present a selection algorithm that matches the information-theoretic lower bound.

We note in passing that *sorting* $X + Y$ is a well studied problem. Fredman [15] showed that $X + Y$, where $|X| = |Y| = n$, can be sorted using only $O(n^2)$ comparisons, but it is not known how to do it in $O(n^2)$ time. (An intriguing situation!) Fredman [15] also showed that $\Omega(n^2)$ comparisons are required, if only comparisons between items in $X + Y$, i.e., comparisons of the form $x_i + y_j \leq x_k + y_\ell$, are allowed. Lambert [24] and Steiger and Streinu [27] gave algorithms that sort $X + Y$ in $O(n^2 \log n)$ time using only $O(n^2)$ comparisons. Kane et al. [21], in a breakthrough result, have shown recently that $X + Y$ can be sorted using only $O(n \log^2 n)$ comparisons of the form $(x_i + y_j) - (x_{i'} + y_{j'}) \leq (x_k + y_\ell) - (x_{k'} + y_{\ell'})$, but it is again not known how to implement their algorithm efficiently.

---

[5] The information gathered by a comparison-based algorithm corresponds to a partial order which can be represented by a DAG (Directed Acyclic Graph). Every *topological sort* of the DAG corresponds to a total order of the items consistent with the partial order. Suppose that e is claimed to be the k-th smallest item and suppose, for the sake of contradiction, that the set I of the items that are *incomparable* with e is non-empty. Then, there is a topological sort in which e is before all the items of I, and another topological sort in which e is after all the items of I, contradicting the fact that e is the k-th smallest item in all total orders consistent with the partial order.

[6] By $X + Y$ we mean the set of pairwise sums $\{x + y \mid x \in X, y \in Y\}$.

The *median* of $X+Y$, on the other hand, can be found in $O(n \log n)$ time, and $O(n \log n)$ comparisons of items in $X+Y$, as was already shown by Johnson and Mizoguchi [18] and Johnson and Kashdan [17]. The selection problem from $X+Y$ becomes more challenging when $k = o(n^2)$.

Frederickson [11] gives two applications for selection from a binary min-heap. The first is in an algorithm for listing the k smallest spanning trees of an input graph. The second is a certain resource allocation problem. Eppstein [10] uses the heap selection algorithm in his $O(m + n \log n + k)$ algorithm for generating the k shortest paths between a pair of vertices in a digraph. As pointed out by Frederickson and Johnson [13], selection from $X+Y$ can be used to compute the Hodges-Lehmann [16] estimator in statistics. Selection from a matrix with sorted rows solves the problem of "optimum discrete distribution of effort" with concave functions, i.e., the problem of maximizing $\sum_{i=1}^{m} f_i(k_i)$ subject to $\sum_{i=1}^{m} k_i = k$, where the f_i's are concave and the k_i's are non-negative integers. (See Koopman [23] and other references in [13].) Selection from a matrix with sorted rows is also used by Brodal et al. [3] and Bremner et al. [2].

The $O(k)$ heap selection algorithm of Frederickson [11] is fairly complicated. The naïve algorithm for the problem runs in $O(k \log k)$ time. Frederickson first improves this to $O(k \log \log k)$, then to $O(k3^{\log^* k})$, then to $O(k2^{\log^* k})$, and finally to $O(k)$.

Our first result is a very simple $O(k)$ heap selection algorithm obtained by running the naïve $O(k \log k)$ algorithm using an auxiliary *soft heap* instead of a standard heap. Soft heaps, discussed below, are fairly simple data structures whose implementation is not much more complicated than the implementation of standard heaps. Our overall $O(k)$ algorithm is thus simple and easy to implement and comprehend.

Relying on our simple $O(k)$ heap selection algorithm, we obtain simplified algorithms for selection from row-sorted matrices and from $X+Y$. Selecting the k-th item from a row-sorted matrix with m rows using our algorithms requires $O(m+k)$ time, if $k \leq 2m$, and $O(m \log \frac{k}{m})$ time, if $k \geq 2m$, matching the optimal results of Frederickson and Johnson [12]. Furthermore, we obtain a new optimal "output-sensitive" algorithm whose running time is $O(m + \sum_{i=1}^{m} \log(k_i + 1))$, where k_i is the number of items of the i-th row that belong to the set of k smallest items in the matrix. We also use our simple $O(k)$ heap selection algorithm to obtain simple optimal algorithms for selection from $X+Y$.

Soft heaps are "approximate" priority queues introduced by Chazelle [6]. They form a major building block in his deterministic $O(m\alpha(m,n))$-time algorithm for finding minimum spanning trees [5], which is currently the fastest known deterministic algorithm for the problem. Chazelle [6] also shows that soft heaps can be used to obtain a simple linear time (standard) selection algorithm. (See the next section.) Pettie and Ramachandran [25] use soft heaps to obtain an *optimal* deterministic minimum spanning tree algorithm with a yet unknown running time. A simplified implementation of soft heaps is given in Kaplan et al. [22].

All algorithms considered in the paper are *comparison-based*, i.e., the only operations they perform on the input items are pairwise comparisons. In the selection from $X+Y$ problem, the algorithms make pairwise comparisons in X, in Y and in $X+Y$. The number of comparisons performed by the algorithms presented in this paper dominates the total running time of the algorithms.

The rest of the paper is organized as follows. In Section 2 we review the definition of soft heaps. In Section 3 we describe our heap selection algorithms. In Section 3.1 we describe our basic algorithm for selection from *binary* min-heaps. In Sections 3.2 and 3.3 we extend the algorithm to d-ary heaps and then to general heap-ordered trees and forests. In Section 4 we

describe our algorithms for selection from row-sorted matrices. In Section 4.1 we describe a simple $O(m + k)$ algorithm which is optimal if $k = O(m)$. In Section 4.2 we build on the $O(m + k)$ algorithm to obtain an optimal $O(m \log \frac{k}{m})$ algorithm, for $k \geq 2m$. In Sections 4.3 and 4.4 we obtain new results that were not obtained by Frederickson and Johnson [12]. In Section 4.3 we obtain an $O(m + \sum_{i=1}^{m} \log n_i)$ algorithm, where $n_i \geq 1$ is the length of the i-th row of the matrix. In Section 4.4 we obtain the new $O(m + \sum_{i=1}^{m} \log(k_i + 1))$ optimal output-sensitive algorithm. In Section 5 we present our algorithms for selection from $X + Y$. In Section 5.1 we give a simple $O(m + n + k)$ algorithm, where $|X| = m$, $|Y| = n$. In Section 5.2 we give a simple $O(m \log \frac{k}{m})$ algorithm, for $m \geq n$ and $k \geq 6m$. We conclude in Section 6 with some remarks and open problems.

2 Soft heaps

Soft heaps, invented by Chazelle [6], support the following operations:

`soft-heap`(ε): Create and return a new, empty soft heap with *error parameter* ε.

`insert`(Q, e): Insert item e into soft heap Q.

`meld`(Q_1, Q_2): Return a soft heap containing all items in heaps Q_1 and Q_2, destroying Q_1 and Q_2.

`extract-min`(Q): Delete from the soft heap and return an item of minimum key in heap Q.

In Chazelle [6], `extract-min` operations are broken into `find-min` and `delete` operations. We only need combined `extract-min` operations. We also do not need `meld` operations in this paper.

The main difference between soft heaps and regular heaps is that soft heaps are allowed to *increase* the keys of some of the items in the heap by an arbitrary amount. Such items are said to become *corrupt*. The soft heap implementation chooses which items to corrupt, and by how much to increase their keys. The only constraint is that for a certain *error parameter* $0 \leq \varepsilon < 1$, the number of corrupt items *in* the heap is at most εI, where I is the number of *insertions* performed so far. The ability to corrupt items allows the implementation of soft heaps to use what Chazelle [6] calls the "data structures equivalent of car pooling" to reduce the amortized time per operation to $O(\log \frac{1}{\varepsilon})$, which is the best possible dependency on ε. In the implementation of Kaplan et al. [22], `extract-min` operations take $O(\log \frac{1}{\varepsilon})$ amortized time, while all other operations take $O(1)$ amortized time. (In the implementation of Chazelle [6], `insert` operations take $O(\log \frac{1}{\varepsilon})$ amortized time while the other operations take $O(1)$ time.)

An `extract-min` operation returns an item whose current, possibly corrupt, key is the smallest in the heap. Ties are broken arbitrarily. (Soft heaps usually give many items the same corrupt key, even if initially all keys are distinct.) Each item e thus has two keys associated with it: its original key $e.key$, and its current key in the soft heap $e.key'$, where $e.key \leq e.key'$. If $e.key < e.key'$, then e is corrupt. The current key of an item may increase several times.

At first sight, it may seem that the guarantees provided by soft heaps are extremely weak. The only bound available is on the number of corrupt items currently *in* the heap. In particular, *all* items extracted from the heap may be corrupt. Nonetheless, soft heaps prove to be an extremely useful data structure. In particular, they play a key role in the fastest known deterministic algorithm of Chazelle [5] for finding minimum spanning trees.

Soft heaps can also be used, as shown by Chazelle [6], to select an *approximate median* of n items. Initialize a soft heap with some error parameter $\varepsilon < \frac{1}{2}$. Insert the n items into the soft heap and then perform $(1-\varepsilon)\frac{n}{2}$ `extract-min` operations. Find the maximum item e, with respect to the original keys, among the extracted items. The rank of e is between $(1-\varepsilon)\frac{n}{2}$ and $(1+\varepsilon)\frac{n}{2}$. The rank is at least $(1-\varepsilon)\frac{n}{2}$ as e is the largest among $(1-\varepsilon)\frac{n}{2}$ items. The rank is at most $(1+\varepsilon)\frac{n}{2}$ as the items remaining in the soft heap that are smaller than e must be corrupt, so there are at most εn such items. For, say, $\varepsilon = \frac{1}{4}$, the running time of the algorithm is $O(n)$.

Using a linear time approximate median algorithm, we can easily obtain a linear time algorithm for selecting the k-th smallest item. We first compute the true rank r of the approximate median e. If $r = k$ we are done. If $r > k$, we throw away all items larger than e. Otherwise, we throw away all items smaller than e and replace k by $k - r$. We then continue recursively. In $O(n)$ time, we reduced n to at most $(\frac{1}{2} + \varepsilon)n$, so the total running time is $O(n)$.

In this paper, we show the usefulness of soft heaps in solving generalized selection problems. We obtain simpler algorithms than those known before, and some results that were not known before.

In Chazelle [6] and Kaplan et al. [22], soft heaps may corrupt items while performing any type of operation. It is easy, however, to slightly change the implementation of [22] such that corruptions only occur *following* `extract-min` operations. In particular, `insert` operations do not cause corruption, and an `extract-min` operation returns an item with a smallest current key at the *beginning* of the operation. These assumptions simplify algorithms that use soft heaps, and further simplify their analysis. The changes needed in the implementation of soft heaps to meet these assumptions are minimal. The operations `insert` (and `meld`) are simply implemented in a *lazy* way. The implementation of [22] already has the property that `extract-min` operations cause corruptions only *after* extracting an item with minimum current key.

We assume that an `extract-min` operation returns a pair (e, C), where e is the extracted item, and C is a list of items that became corrupt after the extraction of e, i.e., items that were not corrupt before the operation, but are corrupt after it. We also assume that $e.corrupt$ is a bit that says whether e is corrupt. (Note that $e.corrupt$ is simply a shorthand for $e.key < e.key'$.) It is again easy to change the implementation of [22] so that `extract-min` operations return a list C of newly corrupt items, without affecting the amortized running times of the various operations. (In particular, the amortized running time of an `extract-min` operation is still $O(\log \frac{1}{\varepsilon})$, independent of the length of C. As each item becomes corrupt only once, it is easy to charge the cost of adding an item to C to its insertion into the heap.)

We stress that the assumptions we make on soft heaps in this paper can be met by minor and straightforward modifications of the implementation of Kaplan et al. [22], as sketched above. No complexities are hidden here. We further believe that due to their usefulness, these assumptions will become the standard assumptions regarding soft heaps.

3 Selection from heap-ordered trees

In Section 3.1 we present our simple, soft heap-based, $O(k)$ algorithm for selecting the k-th smallest item, and the set of k smallest items from a binary min-heap. This algorithm is the cornerstone of this paper. For simplicity, we assume throughout this section that the input heap is infinite. In particular, each item e in the input heap has two children $e.left$ and $e.right$. (A non-existent child is represented by a dummy item with key $+\infty$.) In Section 3.2

```
Heap-Select(r):
  S ← ∅
  Q ← heap()
  insert(Q, r)
  for i ← 1 to k do
      e ← extract-min(Q)
      append(S, e)
      insert(Q, e.left)
      insert(Q, e.right)
  return S
```

Figure 1 Extracting the k smallest items from a binary min-heap with root r using a standard heap.

```
Soft-Select(r):
  S ← ∅
  Q ← soft-heap(1/4)
  insert(Q, r)
  append(S, r)
  for i ← 1 to k − 1 do
      (e, C) ← extract-min(Q)
      if not e.corrupt then
        C ← C ∪ {e}
      for e ∈ C do
          insert(Q, e.left)
          insert(Q, e.right)
          append(S, e.left)
          append(S, e.right)
  return select(S, k)
```

Figure 2 Extracting the k smallest items from a binary min-heap with root r using a soft heap.

we adapt the algorithm to work for d-ary heaps, for $d \geq 3$, using "on-the-fly ternarization via heapification". In Section 3.3 we extend the algorithm to work on any heap-ordered tree or forest. The results of Section 3.3 are new.

3.1 Selection from binary heaps

The naïve algorithm for selection from a binary min-heap is given in Figure 1. The root r of the input heap is inserted into an auxiliary heap (priority queue), denoted Q. The minimal item e is extracted from Q and appended to a list S. The two children of e (in the input heap), if they exist, are inserted into Q. This operation is repeated k times. After k iterations, the items in S are the k smallest items in the input heap, in sorted order. Overall, $2k + 1$ items are inserted into Q and k items are extracted, so the total running time is $O(k \log k)$, which is optimal if the k smallest items are to be reported in sorted order.

Frederickson [11] devised a very complicated algorithm that outputs the k smallest items, not necessarily in sorted order, in only $O(k)$ time, matching the information-theoretic lower bound. In Figure 2 we give our very simple algorithm `Soft-Select`(r) for the same task, which also performs only $O(k)$ comparisons and runs in optimal $O(k)$ time. Our algorithm is a simple modification of the naïve algorithm of Figure 1 with the auxiliary heap replaced by a soft heap. The resulting algorithm is much simpler than the algorithm of Frederickson [11].

Algorithm `Soft-Select`(r) begins by initializing a soft heap Q with error parameter $\varepsilon = 1/4$ and by inserting the root r of the input heap into it. Items inserted into the soft heap Q are also inserted into a list S. The algorithm then performs $k - 1$ iterations. In each iteration, the operation $(e, C) \leftarrow$ `extract-min` extracts an item e with the smallest (possibly corrupt) key currently in Q, and also returns the set of items C that become corrupt as a result of the removal of e from Q. If e is *not* corrupt, then it is added to C. Now, for each item $e \in C$, we insert its two children $e.left$ and $e.right$ into the soft heap Q and the list S.

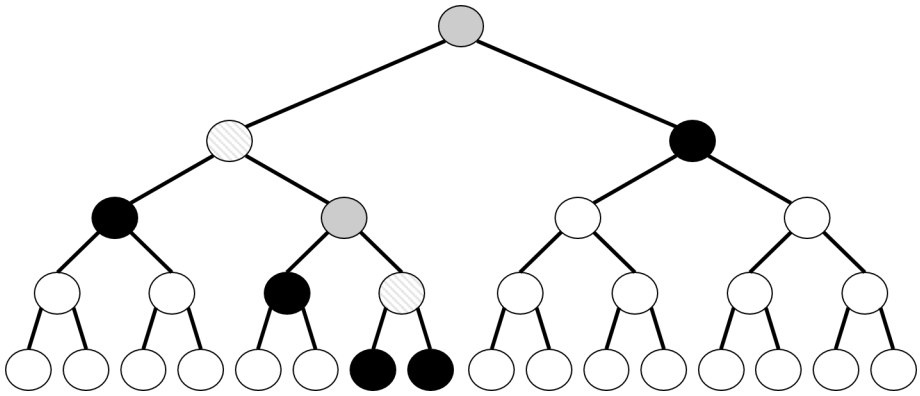


Figure 3 Types of items in the input heap. White nodes belong to A, i.e., were not inserted yet into the soft heap Q; black nodes belong to the barrier B; gray nodes belong to C, i.e., are corrupt; striped nodes belong to D, i.e., were already deleted.

Lemma 1 below claims that `Soft-Select`(r) inserts the k smallest items of the input heap into the soft heap Q. Lemma 2 claims that, overall, only $O(k)$ items are inserted into Q, and hence into S. Thus, the k smallest items in the input heap can be found by selecting the k smallest items in the list S using a standard selection algorithm.

▶ **Lemma 1.** *Algorithm* `Soft-Select`(r) *inserts the k smallest items from the input binary min-heap into the soft heap Q. (Some of them may subsequently be extracted from the heap.)*

Proof. At the beginning of an iteration of algorithm `Soft-Select`, let A be the set of items of the input binary heap that were not yet inserted into the soft heap Q; let B be the set of items that were inserted, not yet removed and are not corrupt; let C be the set of items that were inserted, not yet removed, and are *corrupt*; let D be the set of items that were inserted and already *deleted* from Q. We prove below, by easy induction, the following two invariants:

(a) All strict ancestors of items in B are in $C \cup D$.

(b) Each item in A has an ancestor in B.

Thus, the items in B form a *barrier* that separates the items of A, i.e., items that were not inserted yet into the heap, from the items of $C \cup D$, i.e., items that were inserted and are either corrupt or were already removed from the soft heap Q. For an example, see Figure 3.

Invariants (a) and (b) clearly hold at the beginning of the first iteration, when $B = \{r\}$ and $C \cup D$ is empty. Assume that (a) and (b) hold at the beginning of some iteration. Each iteration removes an item from the soft heap. The item removed is either a corrupt item from C, or an item (in fact the smallest item) on the barrier B. Following the extraction, some items on the barrier B become corrupt and move to C. The barrier is 'mended' by inserting to Q the children of items on B that were extracted or became corrupt. By our simplifying assumption, insertions do not corrupt items, so the newly inserted items belong to B and are thus part of the new barrier, reestablishing (a) and (b).

We now make the following two additional claims:

(c) The item extracted at each iteration is smaller than or equal to the smallest item on the barrier. (With respect to the original keys.)

(d) The smallest item on the barrier cannot decrease.

Claim (c) follows immediately from the definition of an `extract-min` operation and our assumption that corruption occurs only after an extraction. All the items on the barrier, and in particular the smallest item e on the barrier, are in the soft heap and are not corrupt.

Thus, the extracted item is either e, or a corrupt item f whose corrupt key is still smaller than e. As corruption can only increase keys, we have $f < e$.

Claim (d) clearly holds as items on the barrier at the end of an iteration were either on the barrier at the beginning of the iteration, or are children of items that were on the barrier at the beginning of the iteration.

Consider now the smallest item e on the barrier after $k-1$ iterations. As all extracted items are smaller than it, the rank of e is at least k. Furthermore, all items smaller than e must be in $C \cup D$, i.e., inserted at some stage into the heap. Indeed, let f be an item of A, i.e., an item not inserted into Q. By invariant (b), f has an ancestor f' on the barrier. By heap order and the assumption that e is the smallest item on the barrier we indeed get $e \leq f' < f$. Thus, the smallest k items were indeed inserted into the soft heap as claimed. ◀

The proof of Lemma 1 relies on our assumption that corruptions in the soft heap occur only after `extract-min` operations. A slight change in the algorithm is needed if `insert` operations may cause corruptions; we need to repeatedly add children of newly corrupt items until no new items become corrupt. (Lemma 2 below shows that this process must end if $\varepsilon < \frac{1}{2}$. The process may *not* end if $\varepsilon \geq \frac{1}{2}$.) The algorithm, without any change, remains correct, and in particular Lemma 1 holds, if `extract-min` operations are allowed to corrupt items before extracting an item of minimum (corrupt) key. The proof, however, becomes more complicated. (Claim (c), for example, does not hold in that case.)

▶ **Lemma 2.** *Algorithm* `Soft-Select`(r) *inserts only* $O(k)$ *items into the soft heap* Q*.*

Proof. Let I be the number of insertions made by `Soft-Select`(r), and let C be the number of items that become corrupt during the running of the algorithm. (Note that `Soft-Select`(r) clearly terminates.) Let $\varepsilon (= \frac{1}{4})$ be the error parameter of the soft heap. We have $I < 2k + 2C$, as each inserted item is either the root r, or a child of an item extracted during one of the $k-1$ iterations of the algorithm, and there are at most $2k-1$ such insertions, or a child of a corrupt item, and there are exactly $2C$ such insertions. We also have $C < k + \varepsilon I$, as by the definition of soft heaps, at the end of the process at most εI items in the soft heap may be corrupt, and as only $k-1$ (possibly corrupt) items were removed from the soft heap. Combining these two inequalities we get $C < k + \varepsilon(2k + 2C)$, and hence $(1 - 2\varepsilon)C < (1 + 2\varepsilon)k$. Thus, if $\varepsilon < \frac{1}{2}$ we get

$$C \;<\; \frac{1+2\varepsilon}{1-2\varepsilon}\,k \quad , \quad I \;<\; 2\left(1 + \frac{1+2\varepsilon}{1-2\varepsilon}\right)k\,.$$

The number of insertions I is therefore $O(k)$, as claimed. (For $\varepsilon = \frac{1}{4}$, $I < 8k$.) ◀

Combining the two lemmas we easily get:

▶ **Theorem 3.** *Algorithm* `Soft-Select`(r) *selects the* k *smallest items of a binary min-heap in* $O(k)$ *time.*

Proof. The correctness of the algorithm follows from Lemmas 1 and 2. Lemma 2 also implies that only $O(k)$ operations are performed on the soft heap. As $\varepsilon = 1/4$, each operation takes $O(1)$ amortized time. The total running time, and the number of comparisons, performed by the loop of `Soft-Select`(r) is thus $O(k)$. As the size of S is $O(k)$, the selection of the smallest k items from S also takes only $O(k)$ time. ◀

3.2 Selection from d-ary heaps

Frederickson [11] claims, in the last sentence of his paper, that his algorithm for binary min-heaps can be modified to yield an optimal $O(dk)$ algorithm for d-ary min-heaps, for any $d \geq 2$, but no details are given. (In a d-ary heap, each node has (at most) d children.)

We present two simple $O(dk)$ algorithms for selecting the k smallest items from a d-ary min-heap. The first is a simple modification of the algorithm for the binary case. The second in a simple reduction from the d-ary case to the binary case.

Algorithm `Soft-Select`(r) of Figure 2 can be easily adapted to work on d-ary heaps. We simply insert the d children of an extracted item, or an item that becomes corrupt, into the soft heap. If we again let I be the number of items inserted into the sort heap, and C be the number of items that become corrupt, we get $I < d(k + C)$ and $C < k + \varepsilon I$, and hence

$$C \;<\; \frac{1 + d\varepsilon}{1 - d\varepsilon}\, k \quad , \quad I \;<\; d\left(1 + \frac{1 + d\varepsilon}{1 - d\varepsilon}\right) k \;,$$

provided that $\varepsilon < \frac{1}{d}$, e.g., $\varepsilon = \frac{1}{2d}$. The algorithm then performs $O(dk)$ `insert` operations, each with an amortized cost of $O(1)$, and $k - 1$ `extract-min` operations, each with an amortized cost of $O(\log \frac{1}{\varepsilon}) = O(\log d)$. The total running time is therefore $O(dk)$. (Note that it is important here to use the soft heap implementation of [22], with an $O(1)$ amortized cost of `insert`.)

An alternative $O(dk)$ algorithm for d-ary heaps, for any $d \geq 2$, can be obtained by a simple reduction from d-ary heaps to 3-ary (or binary) heaps using a process that we call "on-the-fly ternarization via heapification". We use the well-known fact that an array of d items can be *heapified*, i.e., converted into a binary heap, in $O(d)$ time. (See Williams [28] or Cormen et al. [7].) We describe this alternative approach because we think it is interesting, and because we use it in the next section to obtain an algorithm for general heap-ordered trees, i.e., trees in which different nodes may have different degrees, and the degrees of the nodes are not necessarily bounded by a constant.

In a d-ary heap, each item e has (up to) d children $e.child[1], \ldots, e.child[d]$. We construct a *ternary* heap on the same set of items in the following way. We *heapify* the d children of e, i.e., construct a *binary* heap whose items are these d children. This gives each child f of e two new children $f.left$ and $f.right$. (Some of these new children are null.) We let $e.middle$ be the root of the heap composed of the children of e. Overall, this gives each item e in the original heap three new children $e.left$, $e.middle$ and $e.right$, some of which may be null. Note that e gets its new children $e.left$ and $e.right$ when it and its siblings are heapified. (The names *left*, *middle* and *right* are, of course, arbitrary.) For an example, see Figure 4.

This heapification process can be carried out on-the-fly while running `Soft-Select`(r) on the resulting ternary heap. The algorithm starts by inserting the root of the d-ary heap, which is also the root of its ternarized version, into the soft heap. When an item e is extracted from the soft heap, or becomes corrupt, we do not immediately insert its d original children into the soft heap. Instead, we *heapify* its d children, in $O(d)$ time. This assigns e its middle child $e.middle$. Item e already has its left and right children $e.left$ and $e.right$ defined. The three new children $e.left$, $e.middle$ and $e.right$ are now inserted into the soft heap. We call the resulting algorithm `Soft-Select-Heapify`(r).

▶ **Theorem 4.** *Algorithm* `Soft-Select-Heapify`(r) *selects the k smallest items from a d-ary heap with root r in $O(dk)$ time.*

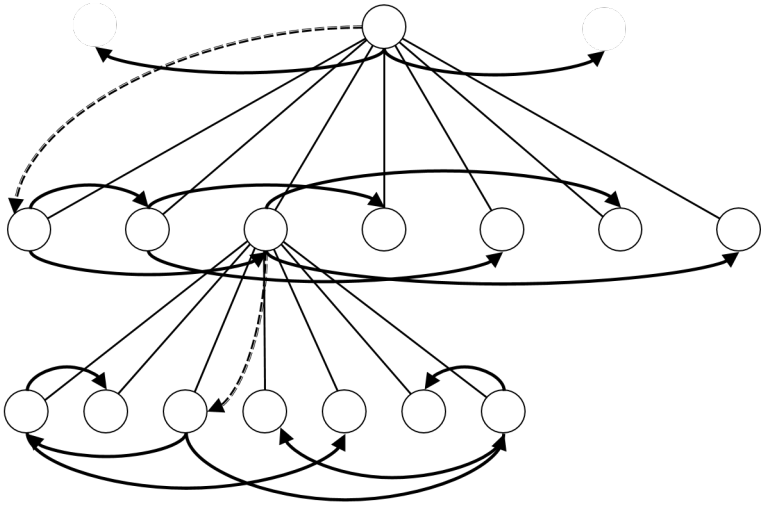


Figure 4 On-the-fly ternarization of a 7-ary heap. Thin lines represent the original 7-ary heap. Bold arrows represent new *left* and *right* children. Dashed arrows represent new *middle* children.

Proof. Algorithm `Soft-Select-Heapify`(r) essentially works on a ternary version of the input d-ary heap constructed on the fly. Simple adaptations of Lemmas 1 and 2 show that the total running time, excluding the heapifications' cost, is $O(k)$. As only $O(k)$ heapifications are performed, the cost of all heapifications is $O(dk)$, giving the total running time of the algorithm. ◀

It is also possible to *binarize* the input heap on the fly. We first ternarize the heap as above. We now convert the resulting ternary tree into a binary tree using the standard first child, next sibling representation. This converts the ternary heap into a binary heap, *if* the three children of each item are sorted. During the ternarization process, we can easily make sure that the three children of each item appear in sorted order, swapping children if necessary, so we can apply this final binarization step.

3.3 Selection from general heap-ordered trees

Algorithm `Soft-Select-Heapify`(r) works, of course, on arbitrary heap-ordered trees in which different nodes have different degrees. Algorithm `Soft-Select`(r), on the other hand, is not easily adapted to work on general heap-ordered trees, as it is unclear how to set the error parameter ε to obtain an optimal running time. To bound the running time of `Soft-Select-Heapify`(r) on an arbitrary heap-ordered tree, we introduce the following definition.

▶ **Definition 5** ($D(T, k)$)**.** Let T be a (possibly infinite) rooted tree and let $k \geq 1$. Let $D(T, k)$ be the maximum *sum of degrees* over all subtrees of T of size k rooted at the root of T. (The degrees summed are in T, not in the subtree.)

For example, if T_d is an infinite d-ary tree, then $D(T_d, k) = dk$, as the sum of degrees in each subtree of T_d containing k vertices is dk. For a more complicated example, let T be the infinite tree in which each node at level i has degree $i+2$, i.e., the root has two children, each of which has three children, etc. Then, $D(T, k) = \sum_{i=2}^{k+1} i = k(k+3)/2$, where the subtrees achieving the maximum are paths from the root. A simple adaptation of Theorem 4 gives:

▶ **Theorem 6.** *Let T be a heap-ordered tree with root r. Algorithm* `Soft-Select-Heapify`(r) *selects the k-th smallest item in T, and the set of k smallest items in T, in $O(D(T, 3k))$ time.*

Proof. We use the on-the-fly binarization and a soft heap with $\varepsilon = \frac{1}{6}$. The number of corrupt items is less than $2k$. The number of extracted items is less than k. Thus, the algorithm needs to heapify the children of less than $3k$ items that form a subtree T' of the original tree T. The sum of the degrees of these items is at most $D(T, 3k)$, thus the total time spent on the heapifications, which dominates the running time of the algorithm, is $O(D(T, 3k))$. We note that $D(T, 3k)$ can be replaced by $D(T, (2 + \delta)k)$, for any $\delta > 0$, by choosing ε small enough. ◀

▶ **Theorem 7.** *Let T be a heap-ordered tree and let $k \geq 1$. Any comparison-based algorithm for selecting the k-th smallest item in T must perform at least $D(T, k-1)-(k-1)$ comparisons on some inputs.*

Proof. Let T' be the subtree of T of size $k - 1$ that achieves the value $D(T, k - 1)$, i.e., the sum of the degrees of the nodes of T' is $D(T, k - 1)$. Suppose the $k - 1$ items of T' are the $k - 1$ smallest items in T. The nodes of T' have at least $D(T, k - 1) - (k - 2)$ children that are not in T'. The k-th smallest item is the minimum item among these items, and no information on the order of these items is implied by the heap order of the tree. Thus, finding the k-th smallest item in this case requires at least $D(T, k - 1) - (k - 1)$ comparisons. ◀

4 Selection from row-sorted matrices

In this section we present algorithms for selecting the k smallest items from a row-sorted matrix, or equivalently from a collection of sorted lists. Our results simplify and extend results of Frederickson and Johnson [12]. The algorithms presented in this section use our `Soft-Select` algorithm for selection from a binary min-heap presented in Section 3.1. (Frederickson's [11] algorithm could also be used, but the resulting algorithms would become much more complicated, in particular more complicated than the algorithms of Frederickson and Johnson [12].) In Section 4.1 we give an $O(m + k)$ algorithm, where m is the number of rows, which is optimal for $k = O(m)$. In Section 4.2 we give an $O(m \log \frac{k+m}{m})$ algorithm which is optimal for $k = \Omega(m)$. These results match results given by Frederickson and Johnson [12]. In Sections 4.3 and 4.4 we give two new algorithms that improve in some cases over the previous algorithms.

4.1 An $O(m + k)$ algorithm

A sorted list may be viewed as a heap-sorted path, i.e., a 1-ary heap. We can convert a collection of m sorted lists into a (degenerate) binary heap by building a binary tree whose leaves are the first items in the lists. The values of the $m - 1$ internal nodes in this tree are set to $-\infty$. Each item in a list will have one real child, its successor in the list, and a dummy child with value $+\infty$. To find the k smallest items in the lists, we simply find the $m + k - 1$ smallest items in the binary heap. This can be done in $O(m + k)$ time using algorithm `Soft-Select` of Section 3.1. More directly, we can use the following straightforward modification of algorithm `Soft-Select`. Insert the m first items in the lists into a soft heap. Perform $k - 1$ iterations in which an item with minimum (corrupt) key is extracted. Insert into the soft heap the child of the item extracted as well as the children of all the items that became corrupt following the `extract-min` operation.

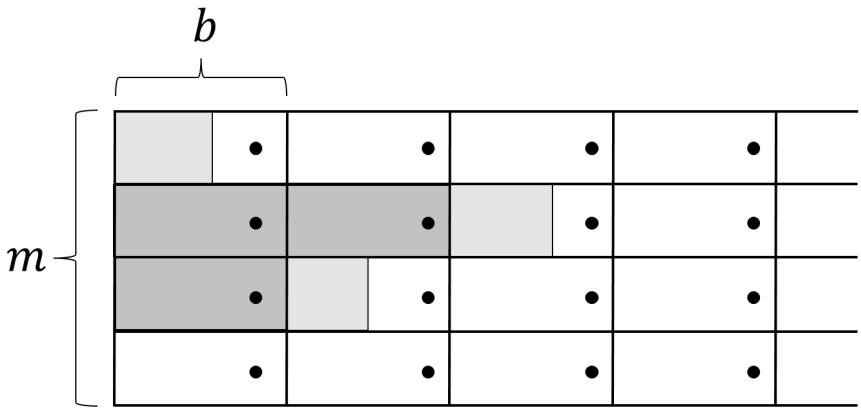


Figure 5 Partitioning the items in each row to blocks of size b. Block representatives are shown as small filled circles. The shaded regions contains the k smallest items. The darkly shaded region depicts blocks all of whose items are among the k smallest.

Alternatively, we can convert the m sorted lists into a heap-ordered tree $T_{m,1}$ by adding a root with value $-\infty$ that will have the m first items as its children. All other nodes in the tree will have degree 1. It is easy to see that $D(T_{m,1}, k) = m + k - 1$. By Theorem 6 we again get an $O(m + k)$ algorithm. We have thus presented three different proofs of the following theorem.

▶ **Theorem 8.** *Let A be a row-sorted matrix containing m rows. Then, the k-th smallest item in A, and the set of k smallest items in A, can be found in $O(m + k)$ time.*

We refer to the algorithm of Theorem 8 as $\texttt{Mat-Select}_1(A, k)$. The $O(m + k)$ running time of $\texttt{Mat-Select}_1(A, k)$ is asymptotically optimal if $k = O(m)$, as $\Omega(m)$ is clearly a lower bound; each selection algorithm must examine at least one item in each row of the input matrix.

4.2 An $O(m \log \frac{k}{m})$ algorithm, for $k \geq 2m$

We begin with a verbal description of the algorithm. Let A be the input matrix and let $k \geq 2m$. Partition each row of the matrix A into *blocks* of size $b = \lfloor \frac{k}{2m} \rfloor$. The *last* item in each block is the *representative* of the block. Consider the (yet unknown) distribution of the k smallest items among the m rows of the matrix. Let k_i be the number of items in the i-th row that are among the k smallest items in the whole matrix. These k_i items are clearly the first k_i items of the i-th row. They are partitioned into a number of full blocks, followed possibly by one partially filled block. (For an example, see Figure 5.) The number of items in partially filled blocks is at most $m \lfloor \frac{k}{2m} \rfloor$. Thus, the number of filled blocks is at least

$$\frac{k - m \lfloor \frac{k}{2m} \rfloor}{\lfloor \frac{k}{2m} \rfloor} \geq m .$$

Apply algorithm $\texttt{Mat-Select}_1$ to select the smallest m block representatives. This clearly takes only $O(m)$ time. (Algorithm $\texttt{Mat-Select}_1$ is applied on the implicitly represented matrix A' of block representatives.) All items in the m blocks whose representatives were selected are among the k smallest items of the matrix. The number of such items is $mb = m \lfloor \frac{k}{2m} \rfloor \geq \frac{k}{4}$, as $k \geq 2m$. These items can be removed from the matrix. All that remains is to select the $k - mb$ smallest remaining items using a recursive call to the algorithm. In each recursive

```
Mat-Select_2(A, k):
    m ← num-rows(A)
    if k ≤ 2m then
        return Mat-Select_1(A, k)
    else
        b ← ⌊k/(2m)⌋
        A' ← jump(A, b)
        K ← Mat-Select_1(A', m)
        A'' ← shift(A, bK)
        return bK + Mat-Select_2(A'', k − bm)
```

Figure 6 Selecting the k smallest items from a row-sorted matrix A. (Implicit handling of submatrices passed to recursive calls.)

```
Mat-Select_2(⟨A, c, D⟩, k):
    m ← num-rows(A)
    if k ≤ 2m then
        return Mat-Select_1(⟨A, c, D⟩, k)
    else
        b ← ⌊k/(2m)⌋
        K ← Mat-Select_1(⟨A, bc, D⟩, m)
        return bK + Mat-Select_2(⟨A, c, D + bcK⟩, k − bm)
```

Figure 7 Selecting the k smallest items from a row-sorted matrix A. (Explicit handling of submatrices passed to recursive calls.)

call (or iteration), the total work is $O(m)$. The number of items to be selected drops by a factor of at least 3/4. Thus after at most $\log_{4/3} \frac{k}{2m} = O(\log \frac{k}{m})$ iterations, k drops below $2m$ and then `Mat-Select`$_1$ is called to finish the job in $O(m)$ time.

Pseudo-code of the algorithm described above, which we call `Mat-Select`$_2(A, k)$ is given in Figure 6. The algorithm returns an array $K = (k_1, k_2, \ldots, k_m)$, where k_i is the number of items in the i-th row that are among the k smallest items of the matrix. The algorithm uses a function *num-rows*(A) that returns the number of rows of a given matrix, a function *jump*(A, b) that returns an (implicit) representation of a matrix A' such that $A'_{i,j} = A_{i,bj}$, for $i, j \geq 1$, and a function *shift*(A, K) that returns an (implicit) representation of a matrix A'' such that $A''_{i,j} = A_{i,j+k_i}$, for $i, j \geq 1$.

In Figure 7 we eliminate the use of *jump* and *shift* and make everything explicit. The input matrix is now represented by a triplet $\langle A, c, D\rangle$, where A is a matrix, $c \geq 1$ is an integer, and $D = (d_1, d_2, \ldots, d_m)$ is an array of non-negative integral *displacements*. `Mat-Select`$_2(\langle A, c, D\rangle, k)$ selects the k smallest items in the matrix A' such that $A'_{i,j} = A_{i,cj+d_i}$, for $i, j \geq 1$. To select the k smallest items in A itself, we simply call `Mat-Select`$_2(\langle A, 1, \mathbf{0}\rangle, k)$, where $\mathbf{0}$ represents an array of m zeros. The implementation of `Mat-Select`$_2(\langle A, c, D\rangle, k)$ in Figure 7 is recursive. It is easy to convert it into an equivalent iterative implementation.

▶ **Theorem 9.** *Let A be a row-sorted matrix containing m rows and let $k \geq 2m$. Algorithm* `Mat-Select`$_2(A, k)$ *selects the k smallest items in A in $O(m \log \frac{k}{m})$ time.*

Frederickson and Johnson [12] showed that the $O(m \log \frac{k}{m})$ running time of algorithm $\texttt{Mat-Select}_2(A, k)$ is optimal, when $k \geq 2m$. A simple proof of this claim can also be found in Section 4.5.

4.3 An $O(m + \sum_{i=1}^{m} \log n_i)$ algorithm

Assume now that the i-th row of A contains only n_i items. We assume that $n_i \geq 1$, as otherwise, we can simply remove the i-th row. We can run algorithms $\texttt{Mat-Select}_1$ and $\texttt{Mat-Select}_2$ of the previous sections by adding dummy $+\infty$ items at the end of each row, but this may be wasteful. We now show that a simple modification of $\texttt{Mat-Select}_2$, which we call $\texttt{Mat-Select}_3$, can solve the selection problem in $O(m + \sum_{i=1}^{m} \log n_i)$ time. We focus first on the number of comparisons performed by the new algorithm.

At the beginning of each iteration, $\texttt{Mat-Select}_2$ sets the block size to $b = \lfloor \frac{k}{2m} \rfloor$. If $n_i < b$, then the last item in the first block of the i-th row is $+\infty$. Assuming that $k \leq \sum_{i=1}^{m} n_i$, no representatives from the i-th row will be selected in the current iteration. There is therefore no point in considering the i-th row in the current iteration. Let m' be the number of *long* rows, i.e., rows for which $n_i \geq \lfloor \frac{k}{2m} \rfloor$. We want to reduce the running time of the iteration to $O(m')$ and still reduce k by some constant factor.

The total number of items in the short rows is less than $m \lfloor \frac{k}{2m} \rfloor \leq \frac{k}{2}$. The long rows thus contain at least $\frac{k}{2}$ of the k smallest items of the matrix. We can thus run an iteration of $\texttt{Mat-Select}_2$ on the long rows with $k' = \frac{k}{2}$. In other words, we adjust the block size to $b' = \left\lfloor \frac{k'}{2m'} \right\rfloor = \lfloor \frac{k}{4m'} \rfloor$ and use $\texttt{Mat-Select}_1$ to select the m' smallest representatives. This identifies $b'm' \geq \frac{k'}{4} \geq \frac{k}{8}$ items as belonging to the k smallest items in A. Thus, each iteration takes $O(m')$ time and reduces k by a factor of at least $\frac{7}{8}$.

In how many iterations did each row of the matrix participate? Let k_j be the number of items still to be selected at the beginning of iteration j. Let $b_j = \left\lfloor \frac{k_j}{2m} \right\rfloor$ be the threshold for long rows used in iteration j. As k_j drops exponentially, so does b_j. Thus, row i participates in at most $O(\log n_i)$ of the *last* iterations of the algorithm. The total number of comparisons performed is thus at most $O(m + \sum_{i=1}^{m} \log n_i)$, as claimed.

To show that the algorithm can also be implemented to run in $O(m + \sum_{i=1}^{m} \log n_i)$ time, we need to show that we can quickly identify the rows that are long enough to participate in each iteration. To do that, we sort $\lceil \log n_i \rceil$ using bucket sort. This takes only $O(m + \max_i \lceil \log n_i \rceil)$ time. When a row loses some of its items, it is easy to move it to the appropriate bucket in $O(1)$ time. In each iteration we may need to examine rows in one bucket that turn out not to be long enough, but this does not affect the total $O(m + \sum_{i=1}^{m} \log n_i)$ running time of the algorithm.

▶ **Theorem 10.** *Let A be a row-sorted matrix containing m rows, and let $N = (n_1, n_2, \ldots, n_m)$, where $n_i \geq 1$ be the number of items in the i-th row of the matrix, for $1 \leq i \leq m$. Let $k \leq \sum_{i=1}^{m} n_i$. Algorithm $\texttt{Mat-Select}_3(A, N, k)$ selects the k smallest items in A in $O(m + \sum_{i=1}^{m} \log n_i)$ time.*

In Section 4.5 below we show that the running time of $\texttt{Mat-Select}_3(A, N, k)$ is optimal for some values of $N = (n_1, n_2, \ldots, n_m)$ and k, e.g., if $k = \frac{1}{2} \sum_{i=1}^{m} n_i$, i.e., for median selection. The $O(m \log \frac{k}{m})$ running time of $\texttt{Mat-Select}_2(A, k)$ is sometimes better than the $O(m + \sum_{i=1}^{m} \log n_i)$ running time of $\texttt{Mat-Select}_3(A, N, k)$. We next describe an algorithm, $\texttt{Mat-Select}_4(A, k)$, which is always at least as fast as the three algorithms already presented, and sometimes faster.

4.4 An $O(m + \sum_{i=1}^{m} \log(k_i + 1))$ algorithm

As before, let k_i be the (yet unknown) number of items in the i-th row that belong to the smallest k items of the matrix. In this section we describe an algorithm for finding these k_i's that runs in $O(m + \sum_{i=1}^{m} \log(k_i + 1))$ time.

We partition each row this time into blocks of size $1, 2, 4, \ldots$. The representative of a block is again the last item in the block. Note that the first k_i items in row i reside in $\lfloor \log(k_i + 1) \rfloor$ complete blocks, plus one incomplete block, if $\log(k_i + 1)$ is not an integer. Thus $L = \sum_{i=1}^{m} \lfloor \log(k_i + 1) \rfloor$ is exactly the number of block representatives that belong to the k smallest items of the matrix.

Suppose that $\ell \geq L$ is an upper bound on the true value of L. We can run $\texttt{Mat-Select}_1$ to select the ℓ smallest block representatives in $O(m + \ell)$ time. If ℓ_i representatives were selected from row i, we let $n_i = 2^{\ell_i + 1} - 1$. We now run $\texttt{Mat-Select}_3$ which runs in $O(m + \sum_{i=1}^{m} \log n_i) = O(m + \sum_{i=1}^{m} (\ell_i + 1)) = O(m + \ell)$. Thus, if $\ell = O(L)$, the total running time is $O(m + \sum_{i=1}^{m} \log(k_i + 1))$, as promised.

How do we find a tight upper bound on $L = \sum_{i=1}^{m} \lfloor \log(k_i + 1) \rfloor$? We simply try $\ell = m, 2m, 4m, \ldots$, until we obtain a value of ℓ that is high enough. If $\ell < L$, i.e., ℓ is not large enough, we can discover it in one of two ways. Either $\sum_{i=1}^{m} n_i < k$, in which case ℓ is clearly too small. Otherwise, the algorithm returns an array of k_i values. We can check whether these values are the correct ones in $O(m)$ time. First compute $M = \max_{i=1}^{m} A_{i,k_i}$. Next check that $A_{i,k_i+1} > M$, for $1 \leq i \leq m$. As ℓ is doubled in each iteration, the cost of the last iteration dominates the total running time which is thus $O(m + 2L) = O(m + \sum_{i=1}^{m} \log(k_i + 1))$. We call the resulting algorithm $\texttt{Mat-Select}_4$.

▶ **Theorem 11.** *Let A be a row-sorted matrix containing m rows and let $k \geq 2m$. Algorithm $\texttt{Mat-Select}_4(A, k)$ selects the k smallest items in A in $O\left(m + \sum_{i=1}^{m} \log(k_i + 1)\right)$ time, where k_i is the number of items selected from row i.*

4.5 Lower bounds for selection from row-sorted matrices

We begin with a simple proof that the $O(m \log \frac{k}{m})$ algorithm is optimal for $k \geq 2m$.

▶ **Theorem 12.** *Any algorithm for selecting the k smallest items from a matrix with m sorted rows must perform at least $(m-1) \log \frac{m+k}{m}$ comparisons on some inputs.*

Proof. We use the information-theoretic lower bound. We need to lower bound $s_k(m)$, which is the number of m-tuples $(k_1, k_2, \ldots, k_m)$, where $0 \leq k_i$, for $1 \leq i \leq m$, and $\sum_{i=1}^{m} k_i = k$. It is easily seen that $s_k(m) = \binom{m+k-1}{m-1}$, as this is the number of ways to arrange k identical balls and $m - 1$ identical dividers in a row. We thus get a lower bound of

$$\log \binom{m+k-1}{m-1} \geq \log \left(\frac{m+k-1}{m-1} \right)^{m-1} = (m-1) \log \frac{m+k-1}{m-1} \geq (m-1) \log \frac{m+k}{m},$$

where we used the well-known relation $\binom{n}{k} > \left(\frac{n}{k}\right)^k$. ◀

We next show that our new $O(m + \sum_{i=1}^{m} \log n_i)$ algorithm is optimal, at least in some cases, e.g., when $k = \frac{1}{2} \sum_{i=1}^{m} n_i$ which corresponds to median selection.

▶ **Theorem 13.** *Any algorithm for selecting the $k = \frac{1}{2} \sum_{i=1}^{m} n_i$ smallest items from a row-sorted matrix with m rows of lengths $n_1, n_2, \ldots, n_m \geq 1$ must perform at least $\sum_{i=1}^{m} \log(n_i + 1) - \log\left(1 + \sum_{i=1}^{m} n_i\right)$ comparisons on some inputs.*

Proof. The number of possible solutions to the selection problem for all values of $0 \leq k \leq \sum_{i=1}^{m} n_i$ is $\prod_{i=1}^{m}(n_i + 1)$. (Each solution corresponds to a choice $0 \leq k_i \leq n_i$, for $i = 1, 2, \ldots, m$.) We prove below that the number of solutions is maximized for $k = \left\lfloor \frac{1}{2} \sum_{i=1}^{m} n_i \right\rfloor$ (and $k = \left\lceil \frac{1}{2} \sum_{i=1}^{m} n_i \right\rceil$). The number of possible solutions for this value of k is thus at least $(\prod_{i=1}^{m}(n_i + 1))/(1 + \sum_{i=1}^{m} n_i)$. Taking logarithm, we get the promised lower bound.

We next prove that the number of solutions is maximized when $k = \left\lfloor \frac{1}{2} \sum_{i=1}^{m} n_i \right\rfloor$. Let X_i be a uniform random variable on $\{0, 1, \ldots, n_i\}$, and let $Y = \sum_{i=1}^{m} X_i$. The number of solutions for a given value k is proportional to the probability that Y attains the value k. Let $Y_j = \sum_{i=1}^{j} X_i$. We prove by induction on j that the distribution of Y_j is maximized at $\mu_j = \frac{1}{2} \sum_{i=1}^{j} n_i$, is symmetric around μ_j, and is increasing up to μ_j and decreasing after μ_j. The base case is obvious as $Y_1 = X_1$ is a uniform distribution. The induction step follows from an easy calculation. Indeed, $Y_j = Y_{j-1} + X_j$, where X_j is uniform and Y_{j-1} has the required properties. The distribution of Y_j is the *convolution* of the distributions of Y_{j-1} and X_j, which corresponds to taking the *average* of $n_j + 1$ values of the distribution of Y_{j-1}. It follows easily that Y_j also has the required properties. ◀

We next compare the lower bound obtained, $\sum_{i=1}^{m} \log(n_i + 1) - \log\left(1 + \sum_{i=1}^{m} n_i\right)$, with the upper bound $O(m + \sum_{i=1}^{m} \log n_i)$. The subtracted term in the lower bound is dominated by the first term, i.e., $\log\left(1 + \sum_{i=1}^{m} n_i\right) \leq \frac{\log(m+1)}{m} \sum_{i=1}^{m} \log(n_i + 1)$, where equality holds only if $n_i = 1$, for every i. When the n_i's are large, the subtracted term becomes negligible. Also, as $n_i \geq 1$, we have $\sum_{i=1}^{m} \log(n_i + 1) \geq m$. Thus, the lower and upper bound are always within a constant multiplicative factor of each other.

The optimality of the $O(m + \sum_{i=1}^{m} \log n_i)$ algorithm also implies the optimality of our new "output-sensitive" $O(m + \sum_{i=1}^{m} \log(k_i + 1))$ algorithm. As $k_i \leq n_i$, an algorithm that performs less than $c(m + \sum_{i=1}^{m} \log(k_i + 1))$ comparisons on all inputs, for some small enough c, would contradict the lower bounds for the $O(m + \sum_{i=1}^{m} \log n_i)$ algorithm. We also note that an argument very similar to the one used in the proof of Theorem 13 shows that $\Omega(\sum_{i=1}^{m} \log(k_i + 1))$ comparisons are required even if the algorithm is given a multiplicative approximation of the k_i's.

5 Selection from $X + Y$

We are given two *unsorted* sets X and Y and would like to find the k-th smallest item, and the set of k smallest items, in the set $X + Y$. We assume that $|X| = m$, $|Y| = n$, where $m \geq n$.

5.1 An $O(m + n + k)$ algorithm

Heapify X and heapify Y, which takes $O(m + n)$ time. Let $x_1, \ldots, x_m$ be the heapified order of X, i.e., $x_i \leq x_{2i}, x_{2i+1}$, whenever the respective items exists. Similarly, let $y_1, \ldots, y_n$ be the heapified order of Y. Construct a heap of maximum degree 4 representing $X + Y$ as follows. The root is $x_1 + y_1$. Item $x_i + y_1$, for $i \geq 1$ has four children $x_{2i} + y_1, x_{2i+1} + y_1, x_i + y_2, x_i + y_3$. Item $x_i + y_j$, for $i \geq 1$, $j > 1$, has two children $x_i + y_{2j}, x_i + y_{2j+1}$, again when the respective items exist. (Basically, this is a heapified version of $X + y_1$, where each $x_i + y_1$ is the root of a heapified version of $x_i + Y$.) We can now apply algorithm `Soft-Select` on this heap, which takes only $O(k)$ time. We call the resulting algorithm `X+Y-Select`$_1(X, Y)$.

▶ **Theorem 14.** *Let X and Y be unordered sets of m and n items respectively. Then, algorithm* `X+Y-Select`$_1(X, Y)$ *finds the k-th smallest item, and the set of k smallest items in $X + Y$, in $O(m + n + k)$ time.*

We note that if X and Y are given to us as heaps, and in particular, if they are sorted, then the running time of algorithm $\texttt{X+Y-Select}_1(X, Y)$ becomes $O(k)$, as there is no need to heapify X and Y.

5.2 An $O(m \log \frac{k}{m})$ algorithm, for $k \geq 6m$, $m \geq n$

If Y is sorted, then $X + Y$ is a row-sorted matrix, and we can use algorithm $\texttt{Mat-Select}_2$ of Theorem 9. We can sort Y in $O(n \log n)$ time and get an $O(m \log \frac{k}{m} + n \log n)$ algorithm. The running time of this algorithm is $O(m \log \frac{k}{m})$ when $k \geq mn^{\varepsilon}$, for any fixed $\varepsilon > 0$. But, for certain values of m, n and k, e.g., $m = n$ and $k = mn^{o(1)}$, the cost of sorting is dominant. We show below that the sorting can always be avoided.

We first regress and describe an alternative $O(m \log \frac{k}{m})$ algorithm for selection from row-sorted matrices. The algorithm is somewhat more complicated than algorithm $\texttt{Mat-Select}_2$ given in Section 4.2. The advantage of the new algorithm is that much less assumptions are made about the order of the items in each row. A similar approach was used by Frederickson and Johnson [12] but we believe that our approach is simpler. In particular we rely on a simple partitioning lemma (Lemma 16 below) which is not used, explicitly or implicitly, in [12].

Instead of partitioning each row into blocks of equal size, as done by algorithm $\texttt{Mat-Select}_2$ of Section 4.2, we partition each row into exponentially increasing blocks, similar, but not identical, to the partition made by algorithm $\texttt{Mat-Select}_3$ of Section 4.3.

Let $b = \lfloor \frac{k}{3m} \rfloor$. Partition each row into blocks of size $b, b, 2b, 4b, \ldots, 2^j b, \ldots$. The representative of a block is again the last item in the block. We use algorithm $\texttt{Mat-Select}_1$ to select the m smallest representatives. This takes $O(m)$ time. Let $e_1 < e_2 < \ldots < e_m$ denote the m selected representatives in (the unknown) sorted order, and let $s_1, s_2, \ldots, s_m$ be the sizes of their blocks. We next use an $O(m)$ *weighted selection* algorithm (see, e.g., Cormen et al. [7], Problem 9.2, p. 225) to find the smallest ℓ such that $\frac{k}{6} \leq \sum_{j=1}^{\ell} s_j$ and the items $e_1, e_2, \ldots, e_\ell$, in some order. Such an $\ell \leq m$ must exist, as $m \lfloor \frac{k}{3m} \rfloor \geq m(\frac{k}{3m} - 1) = \frac{1}{3}(k - 3m) \geq \frac{k}{6}$, as $k \geq 6m$. Also note that $\frac{k}{6} \leq \sum_{j=1}^{\ell} s_j < \frac{k}{3}$, as the addition of each block at most doubles the total size, i.e., $\sum_{j=1}^{\ell} s_j < 2 \sum_{j=1}^{\ell-1} s_j$, for $\ell > 1$.

▶ **Lemma 15.** *All items of the blocks whose representatives are $e_1, e_2, \ldots, e_\ell$ are among the k smallest items in the matrix.*

Proof. Let S_k be the set of k smallest items of the matrix. Consider again the partition of S_k among the m rows of the matrix. Less than $mb = m \lfloor \frac{k}{3m} \rfloor \leq \frac{k}{3}$ of the items of S_k belong to rows that do not contain a full block of S_k items. Thus, at least $\frac{2k}{3}$ of the items of S_k are contained in rows that contain at least one full block of S_k items. The exponential increase in the size of the blocks ensures that in each such row, at least half of the items of S_k are contained in full blocks. Thus, at least $\frac{k}{3}$ of the items of S_k are contained in full blocks. In particular, if $e_1, e_2, \ldots, e_\ell$ are the smallest block representatives, and $\sum_{j=1}^{\ell} s_j \leq \frac{k}{3}$, then all the items in the blocks of $e_1, e_2, \ldots, e_\ell$ belong to S_k. ◀

We can thus remove all the items in the blocks of $e_1, e_2, \ldots, e_\ell$ from the matrix and proceed to find the $k - \sum_{j=1}^{\ell} s_j \leq \frac{5k}{6}$ smallest items of the remaining matrix. When k drops below $6m$, we use the algorithm of $\texttt{Mat-Select}_1$ of Section 4.1. The resulting algorithm performs $O(\log \frac{k}{m})$ iterations, each taking $O(m)$ time, so the total running time is $O(m \log \frac{k}{m})$. (This matches the running time of $\texttt{Mat-Select}_2$, using a somewhat more complicated algorithm.)

We make another small adaptation to the new $O(m \log \frac{k}{m})$ algorithm before returning to the selection from $X + Y$ problem. Instead of letting $b = \lfloor \frac{k}{3m} \rfloor$ and using blocks of size

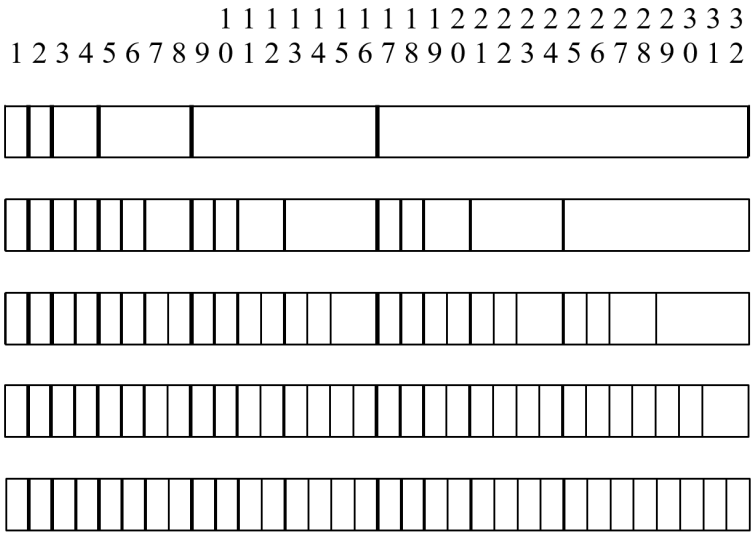


Figure 8 Partitions of Y for $n = 32$.

$b, b, 2b, 4b, \ldots$, we let $b' = 2^{\lfloor \log_2 b \rfloor}$, i.e., b' is the largest power of 2 which is at most b, and use blocks of size $b', b', 2b', 4b', \ldots$. All block sizes are now powers of 2. As the sizes of the blocks may be halved, we select the $2m$ smallest block representatives. The number of items removed from each row in each iteration is now also a power of 2.

Back to the $X + Y$ problem. The main advantage of the new algorithm is that we do not really need the items in each row to be sorted. All we need are the items of ranks $b, b, 2b, 4b, \ldots$, where $b = 2^\ell$ for some $\ell > 0$, in each row. In the $X + Y$ problem the rows, or what remains of them after a certain number of iterations, are related, so we can easily achieve this task.

At the beginning of the first iteration, we use repeated median selection to find the items of Y whose ranks are $1, 2, 4, \ldots$. This also partitions Y into blocks of size $1, 2, 4, \ldots$ such that items of each block are smaller than the items of the succeeding block. We also place the items of ranks $1, 2, 4, \ldots$ in their corresponding places in Y. This gives us enough information to run the first iteration of the matrix selection algorithm.

In each iteration, we refine the partition of Y. We apply repeated median selection on each block of size 2^ℓ in Y, breaking it into blocks of size $1, 1, 2, 4, \ldots, 2^{\ell-1}$. The total time needed is $O(n)$ per iteration, which we can easily afford. We assume for simplicity that $n = |Y|$ is a power of 2 and that all items in Y are distinct. We now have the following fun lemma:

▶ **Lemma 16.** *After i iterations of the above process, if $1 \le r \le n$ has at most i 1's in its binary representation, then $Y[r]$ is the item of rank r in Y, i.e., $Y[1 : r-1] < Y[r] < Y[r+1 : n]$. Additionally, if $r_1 < r_2$ both have at most i 1's in their binary representation, and r_2 is the smallest number larger than r_1 with this property, then $Y[1 : r_1] < Y[r_1+1 : r_2] < Y[r_2+1 : n]$, i.e., the items in $Y[r_1 + 1 : r_2]$ are all larger than the items in $Y[1 : r_1]$ and smaller than the items in $Y[r_2 + 1 : n]$.*

For example, if $n = 32$, then after the first iteration we have the partition

$$Y[1], Y[2], Y[3 : 4], Y[5 : 8], Y[9 : 16], Y[17 : 32] .$$

After the second iteration, we have the partition

$$Y[1], Y[2],\ \ Y[3], Y[4],\ Y[5], Y[6], Y[7:8],\ \ Y[9], Y[10], Y[11{:}12], Y[13:16],$$
$$Y[17], Y[18], Y[19:20], Y[21:24], Y[25:32]\ .$$

(Actually, blocks of size 2 are also sorted.) The partitions obtained for $n = 32$ in the first five iterations are also shown in Figure 8.

Proof. The claim clearly holds after the first iteration, as numbers with a single 1 in their binary representation are exactly powers of 2. Let $Y[r_1 + 1 : r_2]$ be a block of Y generated after i iterations. If r_1 has less than i 1's, then $r_2 = r_1 + 1$, so the block is trivial. Suppose, therefore, that r_1 has exactly i 1's in its representation and that $r_2 = r_1 + 2^\ell$. (ℓ is actually the index of the rightmost 1 in the representation of r_1, counting from 0.) In the $(i+1)$-st iteration, this block is broken into the blocks $Y[r_1 + 1], Y[r_1 + 2], Y[r_1 + 3 : r_1 + 4], \ldots, Y[r_1 + 2^{\ell-1} + 1 : r_1 + 2^\ell]$. As the numbers $r_1 + 2^j$, for $1 \le j < \ell$ are exactly the number between r_1 and r_2 with at most $i + 1$ 1's in their binary representation, this establishes the induction step. ◀

After i iterations of the modified matrix selection algorithm applied to an $X + Y$ instance, we have removed a certain number of items d_i from each row. The number of items removed from each row in each iteration is a power of 2. By induction, d_i has at most i 1's in its representation. In the $(i + 1)$-st iteration we set $b = 2^\ell$, for some $\ell \ge 1$ and need the items of rank $b, 2b, 4b, \ldots$ from what remains of each row. The items needed from the i-th row are exactly $X[i] + Y[d_i + 2^j b]$, for $j = 0, 1, \ldots$. The required items from Y are available, as $d_i + 2^j b$ has at most $i + 1$ 1s in its binary representation! We call the resulting algorithm $\texttt{X+Y-Select}_2(X, Y)$.

▶ **Theorem 17.** *Let X and Y be unordered sets of m and n items respectively, where $m \ge n$, and let $k \ge 6m$. Algorithm $\texttt{X+Y-Select}_2(X, Y)$ finds the k-th smallest item, and the set of k smallest items in $X + Y$, in $O(m \log \frac{k}{m})$ time.*

6 Concluding remarks

We used soft heaps to obtain a very simple $O(k)$ algorithm for selecting the k-th smallest item from a binary min-heap, greatly simplifying the previous $O(k)$ algorithm of Frederickson [11]. We used this simple heap selection algorithm to obtain simpler algorithms for selection from row-sorted matrices and from $X + Y$, simplifying results of Frederickson and Johnson [12]. The simplicity of our algorithms allowed us to go one step further and obtain some improved algorithms for these problems, in particular an $O(m + \sum_{i=1}^{m} \log(k_i + 1))$ "output-sensitive" algorithm for selection from row-sorted matrices.

Our results also demonstrate the usefulness of soft heaps outside the realm of minimum spanning tree algorithms. It would be nice to find further applications of soft heaps.

References

1 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time Bounds for Selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

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 Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro López-Ortiz. Online Sorted Range Reporting. In *Proc. of 20th ISAAC*, pages 173–182, 2009. doi:10.1007/978-3-642-10631-6_19.

4 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, 2013. doi:10.1007/s00493-013-2821-5.

5 Bernard Chazelle. A minimum spanning tree algorithm with Inverse-Ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000. doi:10.1145/355541.355562.

6 Bernard Chazelle. The soft heap: an approximate priority queue with optimal error rate. *J. ACM*, 47(6):1012–1027, 2000. doi:10.1145/355541.355554.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

8 Dorit Dor and Uri Zwick. Selecting the Median. *SIAM Journal on Computing*, 28(5):1722–1758, 1999. doi:10.1137/S0097539795288611.

9 Dorit Dor and Uri Zwick. Median Selection Requires $(2+\epsilon)n$ Comparisons. *SIAM Journal on Discrete Mathematics*, 14(3):312–325, 2001. doi:10.1137/S0895480199353895.

10 David Eppstein. Finding the k Shortest Paths. *SIAM J. Comput.*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

11 Greg N. Frederickson. An Optimal Algorithm for Selection in a Min-Heap. *Information and Computation*, 104(2):197–214, 1993. doi:10.1006/inco.1993.1030.

12 Greg N. Frederickson and Donald B. Johnson. The Complexity of Selection and Ranking in $X + Y$ and Matrices with Sorted Columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982. doi:10.1016/0022-0000(82)90048-4.

13 Greg N. Frederickson and Donald B. Johnson. Generalized Selection and Ranking: Sorted Matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984. doi:10.1137/0213002.

14 Greg N. Frederickson and Donald B. Johnson. Erratum: Generalized Selection and Ranking: Sorted Matrices. *SIAM Journal on Computing*, 19(1):205–206, 1990. doi:10.1137/0219013.

15 Michael L. Fredman. How Good is the Information Theory Bound in Sorting? *Theor. Comput. Sci.*, 1(4):355–361, 1976. doi:10.1016/0304-3975(76)90078-5.

16 Joseph L. Hodges Jr and Erich L. Lehmann. Estimates of location based on rank tests. *The Annals of Mathematical Statistics*, pages 598–611, 1963.

17 Donald B. Johnson and Samuel D. Kashdan. Lower Bounds for Selection in $X + Y$ and Other Multisets. *J. ACM*, 25(4):556–570, 1978. doi:10.1145/322092.322097.

18 Donald B. Johnson and Tetsuo Mizoguchi. Selecting the K-th element in $X + Y$ and $X_1 + X_2 + \ldots + X_m$. *SIAM J. Comput.*, 7(2):147–153, 1978. doi:10.1137/0207013.

19 Jeff Kahn and Jeong Han Kim. Entropy and Sorting. *J. Comput. Syst. Sci.*, 51(3):390–399, 1995. doi:10.1006/jcss.1995.1077.

20 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984. doi:10.1007/BF00565647.

21 Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-SUM and related problems. In *Proc. of 50th STOC*, pages 554–563, 2018. doi:10.1145/3188745.3188770.

22 Haim Kaplan, Robert Endre Tarjan, and Uri Zwick. Soft Heaps Simplified. *SIAM Journal on Computing*, 42(4):1660–1673, 2013. doi:10.1137/120880185.

23 Bernard O. Koopman. The optimum distribution of effort. *Journal of the Operations Research Society of America*, 1(2):52–63, 1953. doi:10.1287/opre.1.2.52.

24 Jean-Luc Lambert. Sorting the sums $(x_i + y_j)$ in $O(n^2)$ comparisons. *Theoretical Computer Science*, 103(1):137–141, 1992.

25 Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002. doi:10.1145/505241.505243.

26 Arnold Schönhage, Mike Paterson, and Nicholas Pippenger. Finding the Median. *J. Comput. Syst. Sci.*, 13(2):184–199, 1976. doi:10.1016/S0022-0000(76)80029-3.

27 William L. Steiger and Ileana Streinu. A Pseudo-Algorithmic Separation of Lines from Pseudo-Lines. *Inf. Process. Lett.*, 53(5):295–299, 1995. doi:10.1016/0020-0190(94)00201-9.

28 J.W.J. Williams. Algorithm 232: Heapsort. *cacm*, 7:347–348, 1964.

Approximating Optimal Transport With Linear Programs

Kent Quanrud[1]
University of Illinois, Urbana-Champaign, Illinois, USA
http://web.engr.illinois.edu/~quanrud2
quanrud2@illinois.edu

Abstract

In the regime of bounded transportation costs, additive approximations for the optimal transport problem are reduced (rather simply) to relative approximations for positive linear programs, resulting in faster additive approximation algorithms for optimal transport.

2012 ACM Subject Classification Theory of computation → Linear programming, Theory of computation → Packing and covering problems, Theory of computation → Parallel algorithms

Keywords and phrases optimal transport, fast approximations, linear programming

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.6

Acknowledgements We thank Jason Altschuler and Chandra Chekuri for insightful discussions and helpful feedback.

1 Introduction

For $\ell \in \mathbb{N}$, let $\Delta^\ell = \left\{p \in \mathbb{R}^\ell_{\geq 0} : \|p\|_1 = 1\right\}$ denote the convex set of probability distributions over $[\ell]$. In the **(discrete) optimal transport problem**, one is given two distributions $p \in \Delta^\ell$ and $q \in \Delta^k$ and a nonnegative matrix of **transportation costs** $C \in \mathbb{R}^{k\times\ell}_{\geq 0}$. The goal is to

$$\text{minimize } \sum_{i=1}^{k}\sum_{j=1}^{\ell} C_{ij}X_{ij}p_j \text{ over } X \in \mathbb{R}^{k\times\ell}_{\geq 0} \text{ s.t. } Xp = q \text{ and } X^t\mathbb{1} = \mathbb{1}. \tag{T}$$

We let (T) denote both the above optimization problem and its optimal value. (T) can be interpreted as the minimum cost of "transporting" a discrete distribution p to a target distribution q, where the cost of moving probability mass from one coordinate to another is given by C. (T) is sometimes called the *earth mover distance* between p and q, where one imagines p and q as each dividing the same amount of sand into various piles, and the goal is to rearrange the piles of sand of p into the piles of sand of q with minimum total effort.

Optimal transport (in much greater generality) is fundamental to applied mathematics [15, 16]. Computing (or approximating) the optimal transport matrix and its cost has many applications: we refer to recent work by Cuturi [8], Altschuler, Weed, and Rigollet [2] and Dvurechensky, Gasnikov, and Kroshnin [9] for further (and up-to-date) references.

Optimal transport is a linear program (abbr. LP) and can be solved exactly by linear program solvers. (T) can also be cast as a minimum cost flow problem, thereby solved combinatorially. The fastest exact algorithm runs in $\tilde{O}\left(k\ell\sqrt{k+\ell}\right)$ time via minimum cost flow [11]. Here and throughout $\tilde{O}(\cdot)$ hides polylogarithmic terms in k, ℓ.

[1] Supported in part by NSF grant CCF-1526799.

There is recent interest, sparked by Cuturi [8], in obtaining *additive approximations* to (T) with running times that are *nearly linear* in the size of the cost matrix C. For $\delta > 0$, a matrix X is a **δ-additive approximation** if it is a feasible solution to (T) with cost at most a δ additive factor more than the optimal transport cost (T). A "nearly linear" running time is one whose dependence on k and ℓ is of the form $O(k\ell \operatorname{polylog}(k, \ell))$; i.e., linear in the input size up to polylogarithmic factors. Cuturi highlights applications in machine learning with large, high-dimensional datasets, for which a faster approximation algorithm may be preferable to a slower exact algorithm.

The first nearly linear time additive approximation was obtained recently by Altschuler et al. [2]. Their result combines a reduction to matrix scaling observed by Cuturi and an improved analysis for a classical matrix scaling algorithm due to Sinkhorn and Knopp [14] as applied to this setting (see also [4]). The bound has a cubic dependency on $\|C\|_\infty/\delta$, where $\|C\|_\infty = \max_{i,j} C_{ij}$ is the maximum value of any coordinate in C and is considered a lower order term. One factor of $1/\delta$ can be removed by recent advances in matrix scaling [7] (per Altschuler et al. [2]). A tighter analysis by Dvurechensky et al. [9] of the Sinkhorn-Knopp approach decreases the dependency on $\|C\|_\infty/\delta$ to the following.

▶ **Theorem 1** ([9]). *A δ-additive approximation to (T) can be computed in $\tilde{O}\left(k\ell\left(\frac{\|C\|_\infty}{\delta}\right)^2\right)$ time.*

1.1 Results

The optimal transport cost can be approximated more efficiently as follows. Some of the results are parametrized by the quantity $\langle p, Cq\rangle$ instead of $\|C\|_\infty$. The quantity $\langle p, Cq\rangle$ is the average cost coefficient as sampled from the product distribution $p \times q$. Needless to say, the average cost $\langle p, Cq\rangle$ is at most the maximum cost $\|C\|_\infty$, and the relative difference may be arbitrarily large.

▶ **Theorem 2.** *One can compute a δ-additive approximate transportation matrix X from p to q sequentially in either*

1. $\tilde{O}\left(k\ell\left(\frac{\langle q, Cp\rangle}{\delta}\right)^2\right)$ *deterministic time or*
2. $\tilde{O}\left(k\ell\frac{\|C\|_\infty}{\delta}\right)$ *randomized time;*

or deterministically in parallel with

3. $\tilde{O}\left(\left(\frac{\langle q, Cp\rangle}{\delta}\right)^3\right)$ *depth and* $\tilde{O}\left(k\ell\left(\frac{\langle p, Cq\rangle}{\delta}\right)^2\right)$ *total work, or*
4. $\tilde{O}\left(\left(\frac{\|C\|_\infty}{\delta}\right)^2\right)$ *depth and* $\tilde{O}\left(k\ell\left(\frac{\|C\|_\infty}{\delta}\right)^2\right)$ *total work.*

The bounds are obtained rather simply by reducing to a variety of relative approximation algorithms for certain types of LPs. The reductions can be summarized briefly as follows.

A simple but important observation is that the transportation cost from p to q is bounded above by $\langle q, Cp\rangle$ (Lemma 3 below). Consequently, $(1 \pm \epsilon)$-multiplicative approximations to the value of (T), for $\epsilon = \delta/\langle q, Cp\rangle$, are δ-additive approximations as well.

The approximate LP solvers produce matrices X that certify the approximate value, but do not meet the constraints of (T) exactly. In particular, the approximations X transport $(1-\epsilon)$-fraction of the mass, leaving ϵ-fraction behind. The remaining ϵ-fraction of probability mass is then transported by a simple oblivious transportation scheme.

Here the algorithms diverge into two types, depending on how to model (T) as an LP. The first approach takes (T) as is, which is a "positive LP". Positive LPs are a subclass of LPs where all coefficients and variables are nonnegative. Positive LPs can be approximated faster than general LPs can be solved. Applied to (T), the approximation algorithms for positive LPs produce what we call "$(1-\epsilon)$-uniform transportation matrices", which not only transport all but an ϵ-fraction of the total mass, but transport all but an ϵ-fraction of each coordinate of p, and fill all but an ϵ-fraction of each coordinate of q. It is shown that $(1-\epsilon)$-uniformly approximate transportation matrix can be altered into exact transportation matrices with an additional cost of about $\epsilon\langle q, Cp\rangle$.

The second approach reformulates (T) as a "packing LP". Packing LPs are a subclass of positive LPs characterized by having only packing constraints. The advantage of packing LPs is that they can be approximated slightly faster than the broader class of positive LPs. However, the approximate transportation matrices X produced by the packing LP are not uniformly approximate in the sense discussed above. Consequently, there is a larger cost of about $\epsilon\|C\|_\infty$ to extend X to an exact transportation matrix.

1.2 Additional background

There is a burgeoning literature on parametrized regimes of optimal transport. The many parametrized settings are beyond the scope of this note, and we refer again to [2, 9] for further discussion.

An important special case of the optimal transport problem (T) is where C is a metric or, more generally, the shortest path metric of an undirected weighted graph. This setting is equivalent to *uncapacitated minimum cost flow*. Let m denote the number of edges and n the number of vertices of the underlying graph. Recently, Sherman [13] proved that a $(1+\epsilon)$-multiplicative approximation to (T) can be obtained in $\tilde{O}\left(m^{1+o(1)}/\epsilon^2\right)$ time. This translates to a δ-additive approximation in $\tilde{O}\left(m^{1+o(1)}(\langle q, Cp\rangle/\delta)^2\right)$ time. Remarkably, if the graph is sparse, then $\tilde{O}\left(m^{1+o(1)}\right)$ is much smaller than the explicit size of the shortest path metric, n^2 – let alone the time required to compute all pairs of shortest paths.

There are many applications where the cost matrix C is induced by some combinatorial or geometric context and may be specified more sparsely than as $O(k\ell)$ explicit coordinates. It is well known that some of the LP solvers used below as a black box, as well as other similar algorithms, can often be extended to handle such implicit matrices so long as one can provide certain simple oracles (e.g., [10, 17, 12, 5, 6]).

The running time (2) of Theorem 2 was obtained independently by Blanchet, Jambulapati, Kent, and Sidford [3], by a similar reduction to packing LPs. Blanchet et al. [3] also get the running time (2) via matrix scaling, more in the spirit of the preceding works [8, 2, 9].

1.3 Organization

The rest of this note is organized as follows. Section 2 outlines a simple and crude approximation algorithm for (T), which is used to repair approximate transportation matrices, and to upper bound (T). Section 3 applies positive LP solvers to approximate (T), and leads to

the running times in Theorem 2 that depend on $\langle q, Cp\rangle$ and not $\|C\|_\infty$. Section 4 applies packing LP solvers to a reformulation of (T). This approach leads to the remaining running times in Theorem 2 that all depend on $\|C\|_\infty$.

2 Oblivious transport

The high-level idea is to use approximate LP solvers to transport most of p to q, and then transport the remaining probability mass with a cruder approximation algorithm. The second step always uses the following oblivious transportation scheme. The upper bound obtained below also provides a frame of reference for comparing additive and relative approximation factors, and is useful for bounding a binary search for the optimal value.

▶ **Lemma 3.** *For a distribution $q \in \Delta^k$, consider the matrix $X \in \mathbb{R}^{k\times\ell}_{\geq 0}$ with each column set to q; i.e., $X_{ij} = q_i$ for all i, j. For any $p \in \Delta^\ell$, X is a transportation matrix from p to q, with total cost $\langle q, Cp\rangle$.*

Proof. Fix $p \in \Delta^\ell$. For any $i \in [m]$,

$$\langle e_i, Xp\rangle = \sum_{j=1}^{\ell} X_{ij}p_j = q_i \sum_{j=1}^{\ell} p_j \overset{(1)}{=} q_i$$

since (1) p is a distribution. For any $j \in [n]$, we have

$$\langle \mathbb{1}, Xe_j\rangle = \sum_{i=1}^{k} X_{ij} = \sum_{i=1}^{k} q_i \overset{(2)}{=} 1.$$

since (2) q is a distribution. Thus X is a transportation matrix from q to p. The transportation cost of X is

$$\sum_{i=1}^{k}\sum_{j=1}^{\ell} C_{ij}X_{ij}p_j = \sum_{i=1}^{k}\sum_{j=1}^{\ell} C_{ij}q_ip_j = \langle q, Cp\rangle,$$

as desired. ◀

3 Reduction to mixed packing and covering

Our first family of approximation algorithms, which obtain the bounds in Theorem 2 that are relative to $\langle q, Cp\rangle$, observe that optimal transport lies in the following class of LPs. A **mixed packing and covering program** is a problem of any of the forms

$$\{\text{find } x,\ \max\langle v, p\rangle,\ \text{or } \min\langle v, p\rangle\} \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \leq b \text{ and } Cx \geq d, \tag{PC}$$

where $A \in \mathbb{R}^{m_1\times n}_{\geq 0}$, $b \in \mathbb{R}^{m_1}_{\geq 0}$, $C \in \mathbb{R}^{m_2\times n}_{\geq 0}$, and $d \in \mathbb{R}^{m_2}_{\geq 0}$, and $v \in \mathbb{R}^n_{\geq 0}$ all have nonnegative coefficients. We let N denote the total number of nonzeroes in the input. For $\epsilon > 0$, an **ϵ-relative approximation** to (PC) is either (a) a certificate that (PC) is either infeasible, or (b) a nonnegative vector $x \in \mathbb{R}^n_{\geq 0}$ such that $Ax \leq (1+\epsilon)b$ and $Cx \geq (1-\epsilon)d$ and, when there is a linear objective and the linear program is feasible, within a $(1 \pm \epsilon)$-multiplicative factor of the optimal value. Relative approximations to positive LPs can be obtained with nearly-linear dependence on N, and polynomial dependency on $\frac{1}{\epsilon}$, as follows.

▶ **Lemma 4** ([17]). *Given an instance of (PC) and $\epsilon > 0$, one can compute a ϵ-relative approximation to (PC) in $\tilde{O}(N/\epsilon^2)$ deterministic time.*

▶ **Lemma 5** ([12]). *Given an instance of a mixed packing and covering problem (PC) and $\epsilon > 0$, one can compute a ϵ-relative approximation to (PC) deterministically in parallel in $\tilde{O}(1/\epsilon^3)$ depth and $\tilde{O}(N/\epsilon^2)$ total work.*

(T) is a minimization instance of mixed packing and covering that is always feasible. The role of nonnegative variables is played by the coordinates of the transportation matrix $X \in \mathbb{R}^{n\times n}_{\geq 0}$, with costs $p_j C_{ij}$ for each X_{ij}. The two equations $Xp = q$ and $X^t\mathbb{1} = \mathbb{1}$ each give rise to two sets of packing constraints, $Xp \leq q$ and $X^t\mathbb{1} \leq \mathbb{1}$, and two sets of covering constraints, $Xp \geq q$ and $X^t\mathbb{1} \geq \mathbb{1}$. We have $N = O(k\ell)$ nonzeroes, $m = 2(k+\ell)$ packing and convering constraints, and $n = k\ell$ variables.

An ϵ-relative approximation to (PC) is not necessarily a feasible solution to (T). To help characterize the difference, we define the following. For fixed $\epsilon > 0$ and two distributions $p \in \Delta^\ell$ and $q \in \Delta^k$, a **$(1-\epsilon)$-uniform transportation matrix** from p to q is a nonnegative matrix $X \in \mathbb{R}^{k\times\ell}_{\geq 0}$ with $(1-\epsilon)q \leq Xp \leq q$ and $(1-\epsilon)\mathbb{1} \leq X^t\mathbb{1} \leq \mathbb{1}$.

▶ **Lemma 6.** *Given an instance of the optimal transport problem (T) and $\epsilon > 0$, a $(1-\epsilon)$-uniform transport matrix with cost at most (T) can be computed*

1. *sequentially in $\tilde{O}\left(\frac{kl}{\epsilon^2}\right)$ time, and*
2. *in parallel in $\tilde{O}\left(\frac{1}{\epsilon^3}\right)$ depth and $\tilde{O}\left(\frac{kl}{\epsilon^2}\right)$ total work.*

Proof. By either Lemma 4 or Lemma 5, one can compute an ϵ-approximation X to (T) with the claimed efficiency. Then $(1-\epsilon)X$ is a $(1-\epsilon)^2$-uniform approximate transportation matrix. ◀

▶ **Lemma 7.** *Given a $(1-\epsilon)$-uniform approximate transportation matrix X, one can compute a transportation matrix U with cost at most an additive factor of $4\epsilon\langle q, Cp\rangle$ more than the cost of X, in linear time and work and with constant depth.*

Proof. We first scale down X slightly to a $(1-2\epsilon)$-uniform transportation matrix, and then augment the shrunken transportation matrix with the oblivious transportation scheme from Lemma 3. Clearly this can be implemented in linear time and work and in constant depth.

Let $Y = \left(1 - \frac{\epsilon}{1-\epsilon}\right)X$. Then Y is a $(1-2\epsilon)$-uniform approximate transportation matrix from p to q. Let $p' = \left(I - \operatorname{diag}(S^t\mathbb{1})\right)p$ and $q' = q - Sp$. Since Y is $(1-2\epsilon)$-uniform, we have $p' \leq 2\epsilon p$ and $q' \leq 2\epsilon q$. p' represents the probability mass not yet transported by Y, and q' represents the probability mass not yet filled by Y, and we have

$$\langle \mathbb{1}, p'\rangle = \langle \mathbb{1}, q'\rangle.$$

Let α denote this common value. Then

$$\alpha = \langle \mathbb{1}, q'\rangle = 1 - \langle \mathbb{1}, Yx\rangle \geq 1 - \langle \mathbb{1}, Xp\rangle + \frac{\epsilon}{1-\epsilon}\langle \mathbb{1}, Xp\rangle \overset{(3)}{\geq} \epsilon$$

because (3) X being $(1-\epsilon)$-uniform implies $1 - \epsilon \leq \langle Xp, \mathbb{1}\rangle \leq 1$. Let Z be the matrix where each column is q'/α; by Lemma 3, Z is a transportation matrix from p'/α to q'/α. Let $Z' = Z(I - \operatorname{diag}(Y^t\mathbb{1}))$. Then

$$(Y + Z')p = Yp + Zp' = Yx + q' = q,$$

and

$$\begin{aligned}(Y+Z')^t\mathbb{1} &= Y^t\mathbb{1} + (I - \operatorname{diag}(Y^t\mathbb{1}))Z^t\mathbb{1} = Y^t\mathbb{1} + (I - \operatorname{diag}(Y^t\mathbb{1}))\mathbb{1} \\ &= Y^t\mathbb{1} + \mathbb{1} - Y^t\mathbb{1} = \mathbb{1},\end{aligned}$$

so $Y + Z'$ is a transportation matrix from p to q. The cost of Y is less than the cost of X, and the cost of Z' is at most

$$\begin{aligned}\sum_{i,j} C_{ij}Z'_{ij}p_j &\overset{(4)}{=} \sum_{i,j} C_{ij}Z_{ij}p'_j \overset{(5)}{=} \frac{1}{\alpha}\sum_{i,j} C_{ij}q'_ip'_j \\ &= \frac{1}{\alpha}\langle q', Cp'\rangle \overset{(6)}{\leq} \frac{4\epsilon^2}{\alpha}\langle q, Cp\rangle \\ &\leq 4\epsilon\langle q, Cp\rangle\end{aligned}$$

by (4) definition of Z', (5) definition of Z, (6) $p' \leq 2\epsilon p$ and $q' \leq 2\epsilon q$, and (7) $\alpha \geq \epsilon$. ◀

▶ **Theorem 8.** *One can deterministically compute a δ-additive approximation to (T)*

1. *sequentially in $\tilde{O}\left(kl\left(\frac{\langle x, Cy\rangle}{\delta}\right)^2\right)$ time, and*
2. *in parallel in $\tilde{O}\left(\left(\frac{\langle x, Cy\rangle}{\delta}\right)^3\right)$ depth and $\tilde{O}\left(kl\left(\frac{\langle x, Cy\rangle}{\delta}\right)^2\right)$ total work.*

Proof. Given $\delta > 0$, let $\epsilon = \frac{\delta}{4\langle q, Cp\rangle}$. We apply Lemma 6 to generate a $(1-\epsilon)$-uniform transportation matrix X of cost at most (T) within the desired time/depth bounds. We then apply Lemma 7 to X to construct a transportation matrix U with cost at most $(\mathsf{T}) + 4\epsilon\langle q, Cp\rangle = (\mathsf{T}) + \delta$, as desired. ◀

4 Reduction to packing

A **(pure) packing LP** is a linear program of the form

$$\text{maximize } \langle c, p\rangle \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ over } Ax \leq b, \tag{P}$$

where $A \in \mathbb{R}^{m\times n}_{\geq 0}$, $b \in \mathbb{R}^m_{\geq 0}$, and $c \in \mathbb{R}^n_{\geq 0}$. For a fixed instance of (P), we let N denote the number of nonzeroes in A. For $\epsilon > 0$, a **$(1-\epsilon)$-relative approximation** to (P) is a point $x \in \mathbb{R}^n_{\geq 0}$ such that $Ax \leq b$ and $\langle c, x\rangle$ is at least $(1-\epsilon)$ times the optimal value of (P). $(1-\epsilon)$-relative approximations packing LPs can be obtained slightly faster than ϵ-relative approximations to more general positive linear programs, as follows.

▶ **Lemma 9** ([1]). *Given an instance of the pure packing problem (P), and $\epsilon > 0$, a $(1-\epsilon)$-multiplicative approximation to (P) can be computed in $\tilde{O}(N/\epsilon)$ randomized time.*

▶ **Lemma 10** ([12]). *Given an instance of the pure packing problem (P), and $\epsilon > 0$, a $(1-\epsilon)$-multiplicative approximation to (P) can be computed deterministically in parallel in $\tilde{O}(1/\epsilon^2)$ depth and $\tilde{O}(N/\epsilon^2)$ total work.*

Consider the following LP reformulation of (T), that is parametrized by a value λ that specifies a desired transportation cost.

$$\begin{aligned} &\max \langle \mathbb{1}, Xp \rangle \text{ over } X \in \mathbb{R}^{k\times \ell}_{\geq 0} \\ &\text{s.t. } \sum_{i=1}^{k} X_{ij} \leq 1 \text{ for all } j, \\ &\sum_{j=1}^{\ell} X_{ij} p_j \leq q_i \text{ for all } i \in [m], \\ &\sum_{i=1}^{k}\sum_{j=1}^{\ell} C_{ij} X_{ij} p_j \leq \lambda. \end{aligned} \tag{$\mathsf{TP}(\lambda)$}$$

The advantage of $(\mathsf{TP}(\lambda))$ compared to (T) is that $(\mathsf{TP}(\lambda))$ is a packing LP, which as observed above can be solved slightly faster than a mixed packing and covering LP. The packing problem $(\mathsf{TP}(\lambda))$ has $m = O(k+\ell)$ packing constraints, $n = k\ell$ variables, and $N = O(k\ell)$ nonzeroes.

$(1-\epsilon)$-approximations to $(\mathsf{TP}(\lambda))$ are not feasible solutions to (T) even for $\lambda = (\mathsf{T})$. To help characterize the difference, we define the following. For fixed $\epsilon > 0$ and two distributions $p \in \Delta^{\ell}$ and $q \in \Delta^{k}$, a **$(1-\epsilon)$-transportation matrix from p to q** is a nonnegative matrix $X \in \mathbb{R}^{k\times\ell}_{\geq 0}$ with $Xp \leq q$, $X^t \mathbb{1} \leq \mathbb{1}$, and $\langle \mathbb{1}, Xp \rangle \geq 1-\epsilon$.

▶ **Lemma 11.** *Consider an instance of the optimal transport problem (T), and let $\epsilon, \delta > 0$ be fixed parameters with $\epsilon \leq \dfrac{\delta}{\langle p, Cq\rangle}$. One can compute an $(1-\epsilon)$-transportation matrix from p to q with cost at most $(\mathsf{T}) + \delta$*

1. *sequentially in $\tilde{O}\left(\dfrac{kl}{\epsilon}\right)$ randomized time, and*
2. *in parallel in $\tilde{O}\left(\dfrac{1}{\epsilon^2}\right)$ depth and $\tilde{O}\left(\dfrac{kl}{\epsilon^2}\right)$ total work.*

Proof. For fixed λ, either $\lambda \leq (\mathsf{T})$, or either $(1-\epsilon)$-approximation algorithm from Lemma 9 or Lemma 10 returns $(1-\epsilon)$-transportation matrices X from p to q with cost at most λ. We wrap the $(1-\epsilon)$-relative approximation algorithms in a binary search for the smallest value, up to an additive factor of δ, that produces a $(1-\epsilon)$-transportation matrix from p to q. Since $\lambda = (\mathsf{T})$ is sufficient, such a search returns a value of $\lambda \leq (\mathsf{T}) + \delta$. By Lemma 3, the search can be bounded to the range $[0, \langle q, Cp\rangle]$. Thus the binary search needs at most $O\left(\log\left(\dfrac{\langle q, Cp\rangle}{\delta}\right)\right)$ iterations to identify such a value λ, for which we obtain the desired $(1-\epsilon)$-transportation matrix. We can assume that $\dfrac{\langle p, Cq\rangle}{\delta}$ is at most $\mathrm{poly}(k,\ell)$, since otherwise (T) can be solved exactly in $\mathrm{poly}(k,\ell) \leq \dfrac{\langle p, Cq\rangle}{\delta} \leq \dfrac{1}{\epsilon}$ time. ◀

▶ **Lemma 12.** *Let X be a $(1-\epsilon)$-transportation matrix from p to q. In $O(k\ell)$ time, one can extend X to a transportation matrix U with an additional cost of $\epsilon\|C\|_{\infty}$.*

Proof. We use the oblivious transportation scheme of Lemma 3 to transport the remaining ϵ-fraction of mass. It is straightforward to verify the additional transportation costs at most $\epsilon\|C\|_{\infty}$, as follows.

Let $p' = (I - \mathrm{diag}(X^t \mathbb{1}))p$ and $q' = q - Xp$. p' represents the probability mass not yet transported by X, and q' represents the probability mass not yet transported by Y. Let $\alpha = \langle \mathbb{1}, p' \rangle = \langle \mathbb{1}, q' \rangle$ denote the residual probability mass. Since X is a $(1-\epsilon)$-transportation matrix, we have

$$\alpha = \langle \mathbb{1}, q' \rangle = 1 - \langle \mathbb{1}, Xp \rangle \leq \epsilon. \tag{1}$$

Let Y be the matrix[2] where each column is set to q'/α; by Lemma 3, Y is a transportation matrix from p'/α to q'/α. Let $Y' = Y(I - \mathrm{diag}(X^t \mathbb{1}))p$. By the same calculations as in the proof of Lemma 7, $X + Y'$ is a transportation matrix from q to p. The cost of Y' is

$$\sum_{ij} C_{ij} Y'_{ij} p_j \overset{(7)}{=} \frac{1}{\alpha} \langle q', C'p' \rangle \overset{(8)}{\leq} \frac{\|q'\|_1 \|p'\|_1}{\alpha} \|C'\|_\infty = \alpha \|C'\|_\infty \overset{(9)}{\leq} \epsilon \|C'\|_\infty$$

by (8) the proof of Lemma 7, (9) Cauchy-Schwartz, and (10) the above inequality (1). ◀

▶ **Theorem 13.** *One can compute a δ-additive approximation to (T)*

1. *sequentially in $\tilde{O}\left(k\ell \frac{\|C\|_\infty}{\delta}\right)$ randomized time, and*
2. *in parallel with $\tilde{O}\left(\left(\frac{\|C\|_\infty}{\delta}\right)^2\right)$ depth and $\tilde{O}\left(k\ell\left(\frac{\|C\|_\infty}{\delta}\right)^2\right)$ total work.*

Proof. Let $\delta > 0$ be fixed. Let $\epsilon = \dfrac{\delta}{2\|C\|_\infty}$. By Lemma 11, we can compute a $(1-\epsilon)$-transportation matrix X with cost at most $(\mathsf{T}) + \dfrac{\delta}{2}$. By Lemma 12, we can extend X to a transportation matrix U with additional cost of at most $\epsilon\|C\|_\infty = \dfrac{\delta}{2}$, for a total cost at most $(\mathsf{T}) + \delta$. ◀

References

1. Zeyuan Allen Zhu and Lorenzo Orecchia. Nearly-Linear Time Positive LP Solver with Faster Convergence Rate. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 229–236, 2015.
2. Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1961–1971, 2017.
3. Jose Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. Towards Optimal Running Times for Optimal Transport. *CoRR*, abs/1810.07717, 2018. `arXiv:1810.07717`.
4. Deeparnab Chakrabarty and Sanjeev Khanna. Better and Simpler Error Analysis of the Sinkhorn-Knopp Algorithm for Matrix Scaling. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 4:1–4:11, 2018.
5. Chandra Chekuri and Kent Quanrud. Near-Linear Time Approximation Schemes for some Implicit Fractional Packing Problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 801–820, 2017.

[2] In fact, any transportation matrix from p' to q' will do.

6 Chandra Chekuri and Kent Quanrud. Randomized MWU for Positive LPs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 358–377, 2018.

7 Michael B. Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix Scaling and Balancing via Box Constrained Newton's Method and Interior Point Methods. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 902–913, 2017.

8 Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2292–2300, 2013.

9 Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational Optimal Transport: Complexity by Accelerated Gradient Descent Is Better Than by Sinkhorn's Algorithm. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1366–1375, 2018.

10 Christos Koufogiannakis and Neal E. Young. A Nearly Linear-Time PTAS for Explicit Fractional Packing and Covering Linear Programs. *Algorithmica*, 70(4):648–674, 2014. Preliminary version in FOCS 2007.

11 Yin Tat Lee and Aaron Sidford. Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014.

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

13 Jonah Sherman. Generalized Preconditioning and Undirected Minimum-Cost Flow. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 772–780, 2017.

14 Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

15 Cédric Villani. *Topics in Optimal Transportation*, volume 58 of *Graduate Studies in Mathematics*. American Mathematical Society, 2003.

16 Cédric Villani. *Optimal transport: old and new*, volume 338 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, Heidelberg, 2009.

17 Neal E. Young. Nearly Linear-Time Approximation Schemes for Mixed Packing/Covering and Facility-Location Linear Programs. *CoRR*, abs/1407.3015, 2014. `arXiv:1407.3015`.

LP Relaxation and Tree Packing for Minimum k-cuts

Chandra Chekuri
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 11786, USA
chekuri@illinois.edu

Kent Quanrud
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 11786, USA
quanrud2@illinois.edu

Chao Xu[1]
Yahoo! Research, New York, NY 10003, USA
chao.xu@oath.com
https://orcid.org/0000-0003-4417-3299

Abstract

Karger used spanning tree packings [14] to derive a near linear-time randomized algorithm for the global minimum cut problem as well as a bound on the number of approximate minimum cuts. This is a different approach from his well-known random contraction algorithm [13, 15]. Thorup developed a fast deterministic algorithm for the minimum k-cut problem via greedy *recursive* tree packings [29].

In this paper we revisit properties of an LP relaxation for k-cut proposed by Naor and Rabani [21], and analyzed in [3]. We show that the dual of the LP yields a tree packing, that when combined with an upper bound on the integrality gap for the LP, easily and transparently extends Karger's analysis for mincut to the k-cut problem. In addition to the simplicity of the algorithm and its analysis, this allows us to improve the running time of Thorup's algorithm by a factor of n. We also improve the bound on the number of α-approximate k-cuts. Second, we give a simple proof that the integrality gap of the LP is $2(1-1/n)$. Third, we show that an optimum solution to the LP relaxation, for all values of k, is fully determined by the principal sequence of partitions of the input graph. This allows us to relate the LP relaxation to the Lagrangean relaxation approach of Barahona [2] and Ravi and Sinha [24]; it also shows that the idealized recursive tree packing considered by Thorup gives an optimum dual solution to the LP. This work arose from an effort to understand and simplify the results of Thorup [29].

2012 ACM Subject Classification Theory of computation → Discrete optimization

Keywords and phrases k-cut, LP relaxation, tree packing

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.7

Funding Work on this paper supported in part by NSF grant CCF-1526799.

[1] This work was done while the author was at University of Illinois at Urbana-Champaign.

1 Introduction

The global minimum cut problem in graphs (MinCut) is well-known and extensively studied. Given an undirected graph $G = (V, E)$ with non-negative edge capacities $c : E \to \mathbb{R}_+$, the goal is to remove a minimum capacity set of edges such that the residual graph has at least two connected components. When all capacities are one, the mincut of a graph is its global edge-connectivity. The k-Cut problem is a natural generalization. Given a graph $G = (V, E)$ and an integer $k \geq 2$, the goal is to remove a minimum capacity set of edges such that the residual graph has at least k connected components. MinCut and k-Cut have been extensively studied in the literature. Initial algorithms for MinCut were based on a reduction to the s-t-mincut problem. However, it was realized later on that it can be solved more efficiently and directly. Currently the best deterministic algorithm for MinCut runs in $O(mn + n^2 \log n)$ time [27] and is based on the maximum adjacency ordering approach of Nagamochi and Ibaraki [19]. On the other hand, there is a near-linear time Monte Carlo randomized algorithm due to Karger [14]. Bridging the gap between the running times for the deterministic and randomized algorithms is a major open problem. In recent work [16, 12] obtained near-linear time deterministic algorithms for *simple* unweighted graphs.

The k-Cut problem is NP-Hard if k is part of the input [10], however, there is a polynomial-time algorithm for any fixed k. Such an algorithm was first devised by Goldschmidt and Hochbaum [10], and subsequently there have been several different algorithms improving the run-time. The randomized algorithm of Karger and Stein [15] runs in $\tilde{O}(n^{2(k-1)})$ time and outputs the optimum cut with high probability. The fastest deterministic algorithm, due to Thorup [29], runs in $\tilde{O}(mn^{2k-2})$ time [29]. Recent work of Gupta, Lee and Li [11] obtains a faster run-time of $\tilde{O}(k^{O(k)} n^{(2\omega/3+o(1))k})$ if the graph has small integer weights, where ω is the exponent in the run-time of matrix multiplication. It is also known that k-Cut is $W[1]$-hard when parameterized by k [6]; that is, we do not expect an algorithm with a run-time of $f(k)n^{O(1)}$. Several algorithms that yield a 2-approximation are known for k-Cut; Saran and Vazirani's algorithm based on repeated minimum-cut computations gives $(2 - 2/k)$-approximation [25]; the same bound can be achieved by removing the $(k - 1)$ smallest weight edges in a Gomory-Hu tree of the graph [25]. Nagamochi and Kamidoi showed that using the concept of extreme sets, a $(2 - 2/k)$-approximation can be found even faster [20]. Naor and Rabani developed an LP relaxation for k-Cut [21] and this yields a $2(1-1/n)$-approximation [3]. Ravi and Sinha [24] obtained another $2(1-1/n)$-approximation via a Lagrangean relaxation approach which was also considered independently by Barahona [2]. A factor of 2, for large k, is the best possible approximation under the Small Set Expansion hypothesis [18]. Recent work has obtained a 1.81 approximation in $2^{O(k^2)} n^{O(1)}$ time [11]; whether a PTAS can be obtained in $f(k)\text{poly}(n)$ time is an interesting open problem.

Motivation and contributions: The main motivation for this work was to simplify and understand Thorup's tree packing based algorithm for k-Cut. Karger's near-linear time algorithm and analysis for the MinCut problem [14] is based on the well-known theorem of Tutte and Nash-Williams (on the minmax relation for edge-disjoint trees in a graph). It is simple and elegant; the main complexity is in the improved running time which is achieved via a complex dynamic program. Karger also tightened the bound on the number of α-approximate minimum cuts in a graph (originally shown via his random contraction algorithm) via tree packings. In contrast to the case of mincut, the main structural result in Thorup's work on k-Cut is much less easy to understand and motivate. His proof consists

of two parts. He shows that an ideal tree packing obtained via a recursive decomposition of the graph, first outlined in [28], has the property that any optimum k-cut crosses some tree in the packing at most $2k-2$ times. The second part argues that a greedy tree packing with sufficiently many trees approximates the ideal tree packing arbitrarily well. The greedy tree packing is closely related to a multiplicative weight update method for solving a basic tree packing linear program, however, no explicit LP is used in Thorup's analysis. Thus, although Thorup's algorithm is very simple to describe (and implement), the analysis is somewhat opaque.

In this paper we make several contributions which connect Thorup's tree packing to the LP relaxation for k-Cut [21]. We outline the specific contributions below.

- We show that the dual of the LP for k-Cut gives a tree packing and one can use a simple analysis, very similar to that of Karger, to show that any optimum k-cut crosses some tree in the packing at most $(2k-3)$ times. Thorup proved a bound of $(2k-2)$ for his tree packing. This leads to a slightly faster algorithm than that of Thorup and also to an improved bound on the number of approximate k-cuts.
- We give a new and simple proof that the integrality gap of the LP for k-cut is upper bounded by $2(1-1/n)$. We note that the proof claimed in [21] was incorrect and the proof in [3] is indirect and technical.
- We show that the optimum solution of the k-cut LP, for all values of k, can be completely characterized by the principal sequence of partitions of the cut function of the given graph. This establishes the connection between the dual of the LP relaxation and the ideal recursive tree packing considered by Thorup. It also shows that the lower bound provided by the LP relaxation is equivalent to the Lagrangean relaxation lower bound considered by Barahona [2] and Ravi and Sinha [24].

Our results help unify and simplify the different approaches to k-cut via the LP relaxation and its dual. A key motivation for this paper is to simplify and improve the understanding of the tree packing approach. For this reason we take a leisurely path and reprove some of Karger's results for the sake of completeness, and to point out the similarity of our argument for k-Cut to the case of MinCut. Readers familiar with [14] may wish to skip Section 3.

Organization: Section 2 sets up some basic notation and definitions. Section 3 discusses Karger's approach for MinCut via tree packings with some connections to recent developments on approximately solving tree packings. Section 4 describes the tree packing obtained from the dual of the LP relaxation for k-Cut and how it can be used to extend Karger's approach to k-Cut. Section 5 gives a new proof that the LP integrality gap for k-Cut is $2(1-1/n)$. In Section 6 we show that the optimum LP solution for all values of k can be characterized by a recursive decomposition of the input graph.

2 Preliminaries

We use n and m to denote the number of nodes and edges in a given graph. For a graph $G=(V,E)$, let $\mathcal{T}(G)$ denote the set of spanning trees of G. For a graph $G=(V,E)$ with edge capacities $c: E \to \mathbb{R}_+$ the *fractional spanning tree packing number*, denoted by $\tau(G)$, is the optimum value of a simple linear program shown in Figure 1 whose variables are $y_T, T \in \mathcal{T}(G)$. The LP has an exponential number of variables but is still polynomial time solvable. There are several ways to see this and efficient strongly combinatorial algorithms are also known [8]. We also observe that there is an optimum solution to the LP whose support has at most m trees since the number of non-trivial constraints in the LP is at most m (one per each edge).

$$\begin{aligned} \max \sum_{T \in \mathcal{T}(G)} & y_T \\ \sum_{T \ni e} & y_T \leq c(e) \quad e \in E \\ & y_T \geq 0 \qquad T \in \mathcal{T}(G) \end{aligned}$$

Figure 1 LP relaxation defining $\tau(G)$.

There is a min-max formula for $\tau(G)$ which is a special case of the min-max formula for matroid base packing due to Tutte and Nash-Williams. To state this theorem we introduce some notation. For a partition $\mathcal{P}$ of the vertex set V let $E(\mathcal{P})$ denote the set of edges that cross the partition (that is, have end points in two different parts) and let $|\mathcal{P}|$ denote the number of parts of $\mathcal{P}$. A *k-cut* is $E(\mathcal{P})$ for some partition $\mathcal{P}$ such that $|\mathcal{P}| \geq k$. A *cut* is a 2-cut. The value of the minimum cut of G is denoted as $\lambda(G)$. It is not hard to see that for any partition $\mathcal{P}$ of the vertex set V, $\tau(G) \leq \frac{c(E(\mathcal{P}))}{|\mathcal{P}|-1}$ since every spanning tree of G contains at least $|\mathcal{P}| - 1$ edges from $E(\mathcal{P})$. The minimum over all partitions of the quantity, $\frac{c(E(\mathcal{P}))}{|\mathcal{P}|-1}$, is also referred to as the *strength* of G (denoted by $\sigma(G)$), and turns out to be equal to $\tau(G)$.

▶ **Theorem 1** (Tutte and Nash-Williams). *For any undirected edge capacitated graph G,*

$$\tau(G) = \min_{\mathcal{P}} \frac{c(E(\mathcal{P}))}{|\mathcal{P}| - 1}.$$

Tutte and Nash-Williams proved the integer packing version of the preceding theorem which is harder; they showed that the maximum number of edge-disjoint spanning trees in a graph G with integer capacities c is give by $\min_{\mathcal{P}} \lfloor \frac{c(E(\mathcal{P}))}{|\mathcal{P}|-1} \rfloor$. The theorem is in fact a special case of matroid base packing theorem and can also be derived via the matroid union theorem of Edmonds; we refer the reader to [26].

A useful and well-known corollary of the preceding theorem is given below.

▶ **Corollary 2.** *For any graph G, $\tau(G) \geq \frac{n}{2(n-1)} \cdot \lambda(G)$. If G is an unweighted graph then $\tau(G) \geq \frac{\lambda(G)+1}{2}$.*

Proof. Consider the partition $\mathcal{P}^*$ that achieves the minimum in the minmax formula. We have $c(E(\mathcal{P})) \geq |\mathcal{P}^*|\lambda(G)/2$ since the capacity of edges leaving each part of $\mathcal{P}^*$ is at least $\lambda(G)$ and an edge in $E(\mathcal{P}^*)$ crosses exactly two parts. Thus,

$$\tau(G) = \frac{c(E(\mathcal{P}^*))}{|\mathcal{P}^*| - 1} \geq \frac{|\mathcal{P}^*|\lambda(G)}{2(|\mathcal{P}^*| - 1)} \geq \frac{n}{2(n-1)}\lambda(G)$$

since $|\mathcal{P}^*| \leq n$. If G is unweighted graph then $|\mathcal{P}| \leq \lambda(G) + 1$ and hence $\tau(G) \geq \frac{\lambda(G)+1}{2}$ as desired. ◀

We say that a tree packing $y : \mathcal{T}(G) \to \mathbb{R}_+$ is $(1-\epsilon)$-approximate if $\sum_{T \in \mathcal{T}(G)} y_T \geq (1-\epsilon)\tau(G)$. Note that we typically want a compact tree packing that can either be explicitly specified via a small number of trees or even implicitly via a data structure representing a collection of trees. Approximate spanning tree packings have been obtained via greedy spanning tree packings which can be viewed as applying the multiplicative weight update method. Recently [4] obtained the following result.

▶ **Theorem 3** ([4]). *There is a deterministic algorithm that, given an edge-capacitated undirected graph on m edges and an $\epsilon \in (0, 1/2)$, runs in $O(m \log^3 n/\epsilon^2)$ time and outputs an implicit representation of a $(1-\epsilon)$-approximate tree packing.*

3 Tree packings and minimum cuts

We review some of Karger's observations and results connecting tree packings and minimum cuts [14] which follow relatively easily via Corollary 2. In this section, we restrict the definition of a cut to an edge set $E(\mathcal{P})$ for a partition $\mathcal{P} = \{S, V \setminus S\}$ with *exactly* two parts. Hence, we use $\delta(S)$ to uniquely identify a cut. However, as one can see later in section 4, the results will also hold for the more general definition of a cut, where $\mathcal{P}$ have *at least* two parts. We rephrase his results and arguments with a slightly different notation. Given a spanning tree T and a cut $A \subseteq E$, following Karger, we say that T h-respects A for some integer $h \geq 1$ if $|E(T) \cap A| \leq h$.

Karger proved that a constant fraction of trees (in the weighted sense) of an optimum packing 2-respect any fixed mincut. In fact this holds for a $(1-\epsilon)$-approximate tree packing for sufficiently small ϵ. The proof, as follows, is an easy consequence of Corollary 2 and an averaging argument. It is convenient to view a tree packing as a probability distribution. Let $p_T = y_T/\tau(G)$. We then have $\sum_T p_T = 1$ for an exact tree packing and for a $(1-\epsilon)$-packing we have $\sum_T p_T \in (1-\epsilon, 1]$. Let $\delta(S)$ be a fixed minimum cut whose capacity is $\lambda(G)$. Let $\ell_T = |E(T) \cap \delta(S)|$ be the number of edges of T that cross S. Let $q = \sum_{T:\ell_T \leq 2} p_T$ be the fraction of trees that 2-respect $\delta(S)$. Since each tree crosses S at least once we have,

$$\sum_T p_T \ell_T = \sum_{T:\ell_T \leq 2} p_T \ell_T + \sum_{T:\ell_T \geq 3} p_T \ell_T \geq q + 3(1-\epsilon-q).$$

Because y is a valid packing,

$$\tau(G) \sum_T p_T \ell_T = \sum_T y_T \ell_T \leq c(\delta(S)) = \lambda(G).$$

Putting the two inequalities together and using Corollary 2,

$$3(1-\epsilon) - 2q \leq \lambda(G)/\tau(G) \leq 2(n-1)/n$$

which implies that

$$q \geq \frac{3}{2}(1-\epsilon) - (1-1/n) = \frac{1}{2} + \frac{1}{n} - \frac{3\epsilon}{2}.$$

If $\epsilon = 0$ this implies that at least half the fraction of trees 2-respect any minimum cut. Let q' be the fraction of trees that 1-respect a minimum cut. One can do similar calculations as above to conclude that

$$q' \geq 2(1-\epsilon) - 2(1-1/n) \geq 2(\frac{1}{n} - \epsilon).$$

Thus, $q' > 0$ as long as $\epsilon < 1/n$. In an optimum packing there is always a tree in the support that 1-respects a mincut. The preceding argument can be generalized in a direct fashion to yield the following useful lemma on α-approximate cuts.

▶ **Lemma 4.** *Let y be a $(1-\epsilon)$-approximate tree packing. Let $\delta(S)$ be an α-approximate minimum cut (i.e., $c(\delta(S)) \leq \alpha\lambda(G)$) for some fixed $\alpha \geq 1$. For a fixed integer $h \geq 1$, let q_h denote the fraction of trees in the packing y that h-respect $\delta(S)$. Then*

$$q_h \geq (1-\epsilon)\left(1 + \frac{1}{h}\right) - \frac{2\alpha}{h}\left(1 - \frac{1}{n}.\right)$$

3.1 Number of approximate minimum cuts

Karger showed that the number of α-approximate minimum cuts is at most $O(n^{2\alpha})$ via his random contraction algorithm [13]. He improved the bound to $O(n^{\lfloor 2\alpha \rfloor})$ (for any fixed α) via tree packings in [14]. We review the latter argument.

For any cut $\delta(S)$, we associate the subset of edges of T that cross S, $E(T) \cap \delta(S)$. In the other direction, removing a set of edges $A \subseteq E(T)$ induces several components in $T - A$, which induces a unique cut in G where any two components of $T - A$ adjacent in T lie on opposite sides of the cut. This gives a bijection between cuts induced by edge removals in T, and cuts in the graph.

Fix $\alpha > 1$, and let $h = \lfloor 2\alpha \rfloor$. Let y be a fixed optimum tree packing supported by some $m' \leq m$ trees. For any α-approximate mincut $\delta(S)$, by Lemma 4 and some simplification, the fraction of trees in the packing y that h-respects $\delta(S)$ is at least

$$q_{h,\alpha} \equiv \frac{1}{\lfloor 2\alpha \rfloor} \left(1 - (2\alpha - \lfloor 2\alpha \rfloor)(1 - 1/n)\right).$$

Observing that $q_{h,\alpha} > 0$, an easy counting argument for approximate mincuts is the following. For each α-approximate mincut, there is at least one tree in the (support of the) packing y which crosses it at most h times. Hence each α-approximate cut can be mapped to a distinct combination of a tree in the packing and at most h edges from that tree. The total number of these combinations is $m'\binom{n-1}{h} = O(mn^h)$.

We can avoid the factor of m by leveraging the fact that $q_{h,\alpha}$ is a constant for every fixed α. We give an informal argument here. A tree can h-respect at most $h^h\binom{n-1}{h}$ distinct cuts, while each α-approximate minimum cut is h-respected by a (constant) $q_{h,\alpha}$-fraction of the tree packing. It follows by a packing argument that the total number of α-approximate mincuts is at most

$$\frac{h^h\binom{n-1}{h}}{q_{h,\alpha}} = O\left(n^h\right).$$

3.2 Algorithm for minimum cut via tree packings

Karger used tree packings to obtain a randomized near linear time algorithm for the global minimum cut. The algorithm is based on combining the following two steps.

- Given a graph G there is a randomized algorithm that outputs $O(\log n)$ trees in $\tilde{O}(m)$ time such that with high probability there is a global minimum cut that 2-respects one of the trees in the packing.
- There is a deterministic algorithm that given a graph G and a spanning tree T, in $\tilde{O}(m)$ time finds the cut of minimum capacity in G that 2-respects T. This is based on a clever dynamic programming algorithm that utilizes dynamic tree data structures.

Only the first step of the algorithm is randomized. Karger solves the first step as follows. Given a capacitated graph G and an $\epsilon > 0$, he sparsifies the graph G to obtain an unweighted skeleton graph H via random sampling such that (i) H has $O(n \log n/\epsilon^2)$ edges (ii) $\lambda(H) = \Theta(\log n/\epsilon^2)$ and (iii) a minimum cut of G corresponds to a $(1+\epsilon)$-approximate minimum cut of H in that the cuts induce the same vertex partition. Karger then uses greedy tree packing in H to obtain a $(1-\epsilon')$-tree packing in H with $O(\log n/\epsilon'^2)$ trees, and via Corollary 2 argues that one of the trees in the packing 2-respects a mincut of G; here ϵ and ϵ' are chosen to be sufficiently small but fixed constants.

We observe that Theorem 3 can be used in place of the sparsification step of Karger. The deterministic algorithm implied by the theorem can be used to find an implicit $(1-\epsilon)$-approximate tree packing in near linear time for any fixed $\epsilon > 0$. For sufficiently small but fixed ϵ, a constant fraction of the trees in the tree packing 2-respect any fixed minimum cut. Thus, if we sample a tree T from the tree packing, and then apply Karger's deterministic algorithm for finding the smallest cut that 2-respects T, we can find a minimum cut with constant probability. We can repeat the sampling $\Theta(\log n)$ times to obtain a high probability bound.

Karger raised the following question in his paper. Can the dynamic programming algorithm for finding the minimum cut that 2-respects a tree be made *dynamic*? That is, suppose T is altered via edge swaps to yield a tree $T' = T - e + e'$ where $e \in E(T)$ is removed and replaced by a new edge e'. Can the solution for T be updated quickly to obtain a solution for T'? Note that G is static, only the tree is changing. The tree packing from Theorem 3 finds an implicit packing via $\tilde{O}(m)$ edge swap operations from a starting tree T_0. Suppose there is a dynamic version of Karger's dynamic program that handles updates to the tree in amortized $g(n)$ time per update. This would yield a *deterministic* algorithm for the global mincut with a total time of $\tilde{O}(mg(n))$. We note that the best deterministic algorithm for capacitated graphs is $O(mn + n^2 \log n)$ [27]. This would be improved be any $g(n) = o(n)$.

4 Tree packings for k-cut via the LP relaxation

In this section we consider the k-Cut problem. Thorup [28] constructed a probability distribution over spanning trees which were obtained via a recursive greedy tree packing and showed that there is a tree T in the support of the distribution such that a minimum weight k-cut contains at most $2(k-1)$ edges of T. He then showed that greedy tree packing with $O(mk^3 \log n)$ trees closely approximates the ideal distribution. With this approach, he derived the currently fastest known deterministic algorithm to find the minimum k-Cut in $\tilde{O}(mn^{2k-2})$ time. This is only slightly slower than the randomized Monte Carlo algorithm of Karger and Stein [15] whose algorithm runs in $\tilde{O}(n^{2k-2})$ time. Thorup's algorithm is fairly simple. However, the proofs are somewhat complex since they rely on the recursive tree packing and its subtle properties. Arguing that the greedy tree packing approximates the recursive tree packing is also technical.

Here we consider a different tree packing for k-Cut that arises from the LP relaxation for k-Cut considered by Naor and Rabani [21]. This LP relaxation is shown in Figure 2. The variables are $x_e \in [0,1], e \in E$ which indicate whether an edge e is cut or not. There is a constraint for each spanning tree $T \in \mathcal{T}(G)$; at least $k-1$ edges from T need to be chosen in a valid k-cut. We note that for $k > 2$ the upper bound constraint $x_e \leq 1$ is necessary.

The dual of the LP is given in Figure 3. Naor and Rabani claimed an integrality gap of 2 for the k-Cut LP. Their proof was incomplete and a correct proof was given in [3] in the context of a more general problem called the Steiner k-Cut problem. Let $\lambda_k(G)$ denote the minimum k-cut capacity in G.

▶ **Theorem 5** ([3]). *The worst case integrality gap of the LP for k-Cut in Figure 2 is $2(1-1/n)$.*

▶ **Corollary 6.** *Let (y, z) be an optimum solution for the dual LP for k-Cut shown in Figure 3. Then*

$$(k-1)\sum_T y_T \geq \frac{n\lambda_k(G)}{2(n-1)} + z(E).$$

$$\begin{aligned} \min \quad & \sum_{e \in E)} c_e x_e \\ \text{s.t.} \quad & \sum_{e \in T} x_e \geq k-1 \text{ for all } T \in \mathcal{T}(G) \\ & x_e \leq 1 \text{ for all } e \in E \\ & x_e \geq 0 \text{ for all } e \in E \end{aligned}$$

Figure 2 An LP relaxation for the k-Cut problem from [21].

$$\begin{aligned} \max \quad & (k-1) \sum_{T \in \mathcal{T}(G)} y_T - \sum_{e \in E} z_e \\ \text{s.t.} \quad & \sum_{T \ni e} y_T \leq c_e + z_e \text{ for all } e \in E \\ & y_T \geq 0 \text{ for all } T \in \mathcal{T}(G) \end{aligned}$$

Figure 3 Dual of the LP relaxation from Figure 2.

Note that Corollary 2 is a special case of the preceding corollary.

▶ Remark. We note that the LP relaxation in Figure 2 assumes that G is connected. This is easy to ensure by adding dummy edges of zero cost to make G connected. However, it is useful to consider the general case when the number of connected components in G is h where we assume for simplicity that $h < k$ (if $h \geq k$ the problem is trivial). In this case we need to consider the maximal forests in G, each of which has exactly $n - h$ edges; to avoid notational overload we use $\mathcal{T}(G)$ to denote the set of maximal forests of G. The LP constraint now changes to

$$\sum_{e \in T} x_e \geq k - h \quad T \in \mathcal{T}(G).$$

Tree packing interpretation of the dual LP: The dual LP has two types of variables. For each edge e there is a variable z_e and for each spanning tree T there is a variable y_T. The dual seeks to add capacity $z : E \to \mathbb{R}_+$ to the original capacities c, and then find a maximum tree packing $y : \mathcal{T}(G) \to \mathbb{R}_+$ within the augmented capacities $c + z$. The objective is $(k-1)\sum_T y_T - \sum_{e \in E} z_e$. Note that for $k = 2$, there is an optimum solution with $z = 0$; this can be seen by the fact that for $k = 2$ the primal LP can omit the constraints $x_e \leq 1$ for $e \in E$. For $k > 2$ it may be advantageous to add capacity to some bottleneck edges (say from a minimum cut) to increase the tree packing value, which is multiplied by $(k-1)$.

Our goal is to show that one can transparently carry over the arguments for global minimum cut via tree packings to the k-Cut setting via (optimum) solutions y, z to the dual LP. Theorem 5 plays the role of Corollary 2. The key lemma below is analogous to Lemma 4.

▶ **Lemma 7.** *Let y, z be an optimum solution to the dual LP for k-Cut shown in Figure 3. Let $E' \subseteq E$ be any subset of edges such that $c(E') \leq \alpha \lambda_k(G)$ for some $\alpha \geq 1$. For each*

tree T, let $\ell_T = |E' \cap E(T)|$ denote the number of edges in both T and E'. For an integer $h \geq (k-1)$, let $q_h = \sum_{T:\ell_T \leq h} p_T$ denote the fraction of the trees in the packing induced by y, z that contain at most h edges from E'. Then

$$q_h \geq 1 - \frac{2\alpha(k-1)(1-\frac{1}{n})}{h+1}.$$

Proof. Let $\tau_k(G)$ denote $\sum_T y_T$ and let $p_T = y_T/\tau_k(G)$. Thus,

$$\sum_T p_T \ell_T = \sum_{T:\ell_T \leq h} p_T \ell_T + \sum_{T:\ell_T \geq (h+1)} p_T \ell_T \geq (h+1)(1-q_h).$$

Because y is a valid tree packing in capacities $c + z$,

$$\tau_k(G) \sum_T p_T \ell_T = \sum_T y_T \ell_T \leq c(E') + z(E') \leq \alpha\lambda_k(G) + z(E') \leq \alpha(\lambda_k(G) + z(E)).$$

In the second to last inequality uses the fact that $\alpha \geq 1$ and $z \geq 0$. Putting the preceding inequalities together, we have

$$(h+1)(1-q_h) \leq \frac{1}{\tau_k(G)} \alpha(\lambda_k(G) + z(E)).$$

We rearrange and simplify the inequality in Corollary 6 as

$$2\left(1 - \frac{1}{n}\right)(k-1) \sum_T y_T \geq \lambda_k(G) + 2(1-1/n)z(E) \geq \lambda_k(G) + z(E).$$

Plugging this inequality into the preceding one yields

$$(h+1)(1-q_h) \leq 2\alpha(k-1)(1-1/n),$$

which implies that

$$q_h \geq 1 - \frac{2\alpha(k-1)(1-\frac{1}{n})}{h+1}.$$ ◀

▶ Remark. Lemma 7 does not require A to be a k-cut. Ultimately we will only apply Lemma 7 to k-cuts.

▶ **Corollary 8.** *Let (y, z) be an optimum solution to the dual LP. For every optimum k-cut $A \subseteq E$ there is a tree T in the support of y such that $|E(T) \cap A| \leq 2k - 3$.*

Proof. We apply Lemma 7 with $h = 2k - 3$ and $\alpha = 1$ and observe that $q_h > 0$, which implies the desired statement. ◀

▶ **Corollary 9.** *Let (y, z) be a $(1-\epsilon)$-approximate solution to the dual LP where $\epsilon < \frac{1}{2k-1}$. For every optimum k-cut $A \subseteq E$ there is a tree T in the support of y such that $|E(T) \cap A| \leq 2k-2$.*

Proof. Let q_h be defined as in Lemma 7. If (y, z) is a $(1-\epsilon)$-approximate solution to the dual LP, we would have

$$(k-1) \sum_T y_T \geq (1-\epsilon)\frac{n\lambda_k(G)}{2(n-1)} + z(E) \geq (1-\epsilon)\frac{\lambda_k(G)}{2} + z(E) \tag{1}$$

in place of Corollary 6.

Examining the proof of Lemma 7, we see that optimality of (y, z) is not used in the proof except when invoking Corollary 6. Repeating the proof of Lemma 7, except using (1) instead of Corollary 6 and setting $\alpha = 1$, we obtain the bound

$$q_h \geq 1 - \frac{2(k-1)}{(h+1)(1-\epsilon)}.$$

We observe that $q_h > 0$ for $h = 2k - 2$ and $\epsilon < \frac{1}{2k-1}$. ◀

4.1 Number of approximate k-cuts

We now prove the following theorem.

▶ **Theorem 10.** *Let $G = (V, E)$ be an undirected edge-weighted graph and let k be a fixed integer. For $\alpha \geq 1$, the number of cuts with weight $\leq \alpha\lambda_k(G)$ is $O\left(n^{\lfloor 2\alpha(k-1)\rfloor}\right)$.*

Proof. Let $h = \lfloor 2\alpha(k-1)\rfloor$. By Lemma 7, there is a fixed value

$$q_h = 1 - \frac{2\alpha(k-1)\left(1-\frac{1}{n}\right)}{h+1} > 0$$

such that for any cut A with total weight $\leq \alpha\lambda_k(G)$, then at least a q_h-weighted fraction of trees in the tree packing y contains at most h edges of A.

For a given tree T, consider the number of distinct cuts that contain h or fewer edges in T. There are at most n^h subsets of the tree's edges of size at most h, and each selection induces $f(h)$ partitions of the components $\leq h+1$ into at least 2 parts for some $f(h) < h^h$. Thus there are at most $f(h)n^h$ cuts containing h or fewer edges from T for some fixed function f.

If there are (strictly) more than $f(h)n^h/q_h$ distinct cuts with weight at most $\alpha\lambda_k(G)$, then by the pigeonhole principle there exists a tree T in the packing that induces strictly more than $f(h)n^h$ different cuts with h or fewer edges – a contradiction. ◀

Like Lemma 7, Theorem 10 is not restricted to k-cuts. The primary application of Theorem 10 is to count the number of approximate minimum k-cuts, as follows.

▶ **Corollary 11.** *Let $G = (V, E)$ be an undirected edge-weighted graph and let k be a fixed integer. For $\alpha \geq 1$, the number of α-approximate minimum k-cuts is $O(n^{\lfloor 2\alpha(k-1)\rfloor})$.*

4.2 Enumerating all minimum k-cuts

We briefly describe how to enumerate all k-cuts via Lemma 7. The argument is basically the same as that of Karger and Thorup. First, we compute an optimum solution (y^*, z^*) to the dual LP. We can do this via the Ellipsoid method or other ways. Let $\beta(n, m)$ be the running time to find (y^*, z^*). Moreover, we find a basic feasible solution to the dual LP we are guarantees that the support of y has at most m distinct trees. Now Lemma 7 guarantees that for every minimum k-cut $A \subseteq E$ there is a tree T such that $y(T) > 0$ and T $(2k-3)$-respects A. Thus, to enumerate all minimum k-cuts the following procedure suffices. For each of the trees T in the optimum packing we enumerate all k-cuts induced by removing $h = 2k-3$ edges from T. With appropriate care and data structures (see [14] and [28]) this can be done for a single tree T in $\tilde{O}(n^{2k-3} + m)$ time. The total time over all m trees in the support of y is $\tilde{O}(mn^{2k-3})$ for $k > 2$. This gives the following theorem.

▶ **Theorem 12.** *For $k > 2$ all the minimum k-cuts of a graph with n nodes and m edges can be computed in time $\tilde{O}(mn^{2k-3} + \beta(m, n))$ time where $\beta(m, n)$ is the time to find an optimum solution to the LP for k-cut.*

We observe that Thorup's algorithm [28] runs in time $\tilde{O}(mn^{2k-2})$. Thorup uses greedy tree packing in place of solving the LP. The optimality of the LP solution was crucial in using the bound of $2k-3$ instead of $2k-2$. Thus, even though we obtain a slightly faster algorithm than Thorup, we need to find an optimum solution to the LP which can be done via the Ellipsoid method. The Ellipsoid method is not quite practical. Below we discuss a different approach.

In recent work Quanrud showed that a $(1-\epsilon)$-approximate solution to the dual LP can be computed in near-linear time. We state his theorem below.

▶ **Theorem 13** ([23]). *There is an algorithm that computes a $(1-\epsilon)$-approximate solution (y, z) the dual LP in $O(m \log^3 n/\epsilon^2)$ time.*

We observe that the preceding theorem guarantees $O(m \log^3 n/\epsilon^2)$ trees in the support of y and also implicity stores them in $O(m \log^3 n/\epsilon^2)$ space. If we choose $\epsilon < 1/(2k-1)$ then, via Corollary 9, for every minimum k-cut $A \subseteq E$ there is a tree T in the support of y that $(2k-2)$-respects A. This leads to an algorithm that in $\tilde{O}(mn^{2k-2})$ time enumerates all minimum k-cuts and recovers Thorup's running time. However, we note that the trees generated by the algorithm in the preceding theorem are implicit, and can be stored in small space. It may be possible to use this additional structure to match or improve the run-time achieved by Theorem 12.

▶ Remark. For unweighted graphs with $\tilde{O}(\frac{m}{n-k}\frac{1}{\epsilon^2})$ trees [23] guarantees a $(1-\epsilon)$-approximation. This improves the running time to $\tilde{O}(mn^{2k-3})$ for unweighted graphs.

We briefly discuss a potential approach to speed up the computation futher. Recall that Karger describes an algorithm that given a spanning tree T of a graph G finds the minimum cut that 2-respects T in $\tilde{O}(m)$ time, speeding up the easier $\tilde{O}(n^2)$ time algorithm. We can leverage this as follows. In the case of $k > 2$ we are given T and G and wish to find the minimum k-cut induced by the removal of at most t edges where t is either $2k-3$ or $2k-2$ depending on the tree packing we use. Suppose A is a set of $t-2$ edges of T. Removing them from T yields a forest with $t-1$ components. We can then apply Kargers algorithm in each of these components with an appropriate graph. This results in a running time of $\tilde{O}(mn^{t-2})$ per tree rather than $\tilde{O}(n^t)$. We can try to build on Karger's ideas to improve the running time to find the best 3-cut induced by removing at most 4 edges from T. We can then leverage this for larger values of k.

5 A new proof of the LP integrality gap for k-Cut

The proof of Theorem 5 in [3] is based on the primal-dual algorithm and analysis of Agarwal, Klein and Ravi [1], and Goemans and Williamson [9] for the Steiner tree problem. For this reason the proof is technical and indirect. Further, the proof from [3] is described for the Steiner k-cut problem which has additional complexity. Here we give a different and intuitive proof for k-Cut. Unlike the proof in [3], the proof here relies on optimality properties of the LP solution and hence is less useful algorithmically. We note that [23] uses Theorem 13 and a fast implementation of the algorithmic proof in [3] to obtain a near-linear time $(2+\epsilon)$-approximation for k-Cut.

Let $G = (V, E)$ be a graph with non-negative edge capacities $c_e, e \in E$. We let $\deg(v) = \sum_{e \in \delta(v)} c_e$ denote the capacitated degree of node v. We will assume without loss of generality that $V = \{v_1, v_2, \ldots, v_n\}$ and that the nodes are sorted in increasing order of degrees, that is, $\deg(v_1) \leq \deg(v_2) \leq \ldots \leq \deg(v_n)$. We observe that $\deg(v_1) + \deg(v_2) + \ldots + \deg(v_{k-1})$ is an upper bound on the value of an optimum k-Cut; removing all the edges incident to $v_1, v_2, \ldots, v_{k-1}$ gives a feasible solution in which the components are the $k-1$ isolated vertices $\{v_1\}, \{v_2\}, \ldots, \{v_{k-1}\}$, and a component consisting of the remaining nodes of the graph.

The key lemma is the following which proves the integrality gap in a special case.

▶ **Lemma 14.** *Let G be a connected graph and let x^* be an* optimum *solution to the k-Cut LP such that $x^*(e) \in (0,1)$ for each $e \in E$ (in other words x^* is fully fractional). Then $\sum_{i=1}^{k-1} \deg(v_i) \leq 2(1-1/n) \sum_e c_e x^*_e$.*

Proof. Let (y^*, z^*) be any fixed optimum solution to the dual LP. Complementary slackness gives the following two properties:

- $z^*(e) = 0$ for each $e \in E$, for if $z^*(e) > 0$ we would have $x^*(e) = 1$.
- for each $e \in E$, $\sum_{T \ni e} y^*_T = c_e$ since $x^*(e) > 0$.

From the second property above, and the fact that each spanning tree has exactly $(n-1)$ edges, we conclude that

$$(n-1)\sum_T y^*_T = \sum_{e \in E} c_e. \tag{2}$$

Since the degrees are sorted,

$$\sum_{i=1}^{k-1} \deg(v_i) \leq \frac{k-1}{n} \sum_{i=1}^{n} \deg(v_i) = 2\frac{k-1}{n} \sum_e c_e. \tag{3}$$

Putting the two preceding inequalities together,

$$\sum_{i=1}^{k-1} \deg(v_i) \leq 2(1-\frac{1}{n})(k-1)\sum_T y^*_T = 2(1-\frac{1}{n})\sum_e c_e x^*_e,$$

where, the last equality is based on strong duality and the fact that $z^* = 0$. ◀

The preceding lemma can be easily generalized to the case when G has h connected components following the remark in the preceding section on the k-Cut LP. This gives us the following.

▶ **Corollary 15.** *Let G be a graph with h connected components and let x^* be an* optimum *solution to the k-Cut LP such that $x^*(e) \in (0,1)$ for each $e \in E$. Then $\sum_{i=1}^{k-h} \deg(v_i) \leq 2(1-1/n)\sum_e c_e x^*_e$.*

Now we consider the general case when the optimum solution x^* to the k-Cut LP is not necessarily fully fractional as needed in Lemma 14. The following claim is easy.

▶ Claim 5.1. Let $x^*(e) = 0$ where $e = uv$. Let G' be the graph obtained from G by contracting u and v into a single node. Then there is a feasible solution x' to the k-Cut LP in G' of the same cost as that of x^*. Moreover a feasible k-cut in G' is a feasible k-cut in G of the same cost.

Using the preceding claim we can assume without loss of generality that $x^*(e) > 0$ for each $e \in E$. Let $F = \{e \in E \mid x^*(e) = 1\}$. Since the LP solution x^* paid for the full cost of the edges in F, we can recurse on $G' = G - F$ and the fractional solution x' obtained by restricting x^* to $E \setminus F$. If G' is connected then x' is an optimum solution the k-Cut LP on G', and is fully fractional, and we can apply Lemma 14. However, G' can be disconnected. Let h be the number of connected components in G'. If $h \geq k$ we are done since A is a feasible k-cut and $c(A) \leq \sum_e c_e x^*_e$. The interesting case is when $2 \leq h < k$. In this case we apply Corollary 15 based on the following claim which is intuitive and whose formal proof we omit.

▶ Claim 5.2. Let x' be the restriction of x^* to $E \setminus F$. Then for any maximal forest T in G' we have $\sum_{e \in T} x'(e) \geq k - h$. Moreover, x' is an optimum solution to the k-Cut LP in G'.

From Corollary 15 we can find $E' \subset E \setminus F$ such that $G' - E'$ induces a k-cut in G' such that

$$c(E') \leq 2\left(1 - \frac{1}{n}\right) \sum_{e \in E \setminus F} c_e x'_e = 2\left(1 - \frac{1}{n}\right) \sum_{e \in E \setminus F} c_e x^*_e.$$

Therefore $F \cup E'$ is a k-cut in G and we have that

$$c(F \cup E') = c(F) + c(E') \leq \sum_{e \in F} c_e x^*_e + 2(1 - \frac{1}{n}) \sum_{e \in E \setminus F} c_e x^*_e \leq 2(1 - \frac{1}{n}) \sum_{e \in E} c_e x^*_e.$$

This finishes the proof. Note that the proof also gives a very simple rounding algorithm assuming we have an optimum solution x^* for the LP. Contract all edges with $x^*(e) = 0$, remove all edges e with $x^*(e) = 1$, and use Corollary 14 in the residual graph to find the $(k - h)$ smallest degrees vertices.

An LP-based proof of Theorem 1: The preceding proof idea also yields a proof of Theorem 1, which we sketch here. We are not sure whether this argument has been considered previously. Recall that $\tau(G)$ is the optimum solution value to the tree packing LP, which corresponds to the dual of the k-Cut LP when $k = 2$. When $k = 2$, as we remarked, the LP does not require the upper bound constraints $x(e) \leq 1$ which implies that the dual tree packing LP does not have the z variables. Following Lemma 14 we consider a fully fractional optimum solution x^* to the k-Cut LP with $k = 2$ and an optimum dual solution y^* to the dual tree packing LP. We have

$$(n - 1) \sum_T y^*_T = (n - 1)\tau(G) = \sum_{e \in E} c_e.$$

Consider the partition $\mathcal{P}$ consisting of all the singleton vertices; all edges cross this partition, hence $c(\mathcal{P}) = \sum_e c_e$ and $|\mathcal{P}| = n$. Since $\tau(G) = \frac{\sum_e c_e}{n-1} = \frac{c(\mathcal{P})}{|\mathcal{P}|-1}$ it must be the case that $\frac{\sum_e c_e}{n-1}$ is the network strength which equals the tree packing value. When x^* is not fully fractional we can contract edges with $x^*_e = 0$ and apply the preceding argument. A similar argument can be used to prove the corresponding min-max relation for the fractional packing of bases of a matroid.

6 Characterizing the optimum LP solution

We have seen that the dual of the LP relaxation for k-Cut yields a tree packing that can be used in place of Thorup's recursive tree packing. In this section we show that the two are the same by characterizing the optimum LP solution for a given graph through a recursive partitioning procedure. This yields a nested sequence of partitions of the vertex set of the graph. This sequence is called the principal sequence of partitions of a graph and is better understood in the more general context of submodular functions [22]. We refer the reader to Fujishige's article for more on this topic [7], and to [5, 17] for algorithmic aspects in the setting of graphs. We also connect the LP relaxation with the Lagrangean relaxation approach for k-Cut considered by Barahona [2] and Ravi and Sinha [24]. Their approach is also built upon the principal sequence of partitions. In order to keep the discussion simple we mainly follow the notation and approach of [24].

Given $G = (V, E)$ and an edge set $A \subseteq E$ let $\kappa(A)$ denote the number of connected components in $G - A$. Recall that the strength of a capacitated graph G, denoted by $\sigma(G)$ is defined as $\min_{A \subseteq E} \frac{c(A)}{\kappa(A)-1}$. The k-Cut problem can be phrased as $\min_{A:\kappa(A) \geq k} c(A)$. However, the constraint that $\kappa(A) \geq k$ is not straightforward. It is, however, not hard to show that $\kappa(A)$ is a supermodular set function over the ground set E. A Lagrangean relaxation approach was considered in [2, 24]. To set this up we define, for any fixed edge set A, a function $g_A : \mathbb{R}_+ \to \mathbb{R}$ as $g_A(b) = c(A) - b(\kappa(A) - 1)$. We then obtain the function

$g : \mathbb{R}_+ \to \mathbb{R}$ where $g(b) = \min_{A \subseteq E} c(A) - b(\kappa(A) - 1)$. The quantity $g(b)$ is the attack value of the graph for parameter b and was considered by Cunningham [5] in his algorithm to compute the strength of the graph.

Then, as noted in [2, 24],

$$\min_{A:\kappa(A)\geq k} c(A) \geq \max_{b\geq 0} \min_{A\subseteq E} c(A) + b(k - \kappa(A)) = \max_{b\geq 0} g(b) + b(k-1).$$

Thus $g'(b) = g(b) + b(k-1)$ provides a lower bound on the optimum solution value. [24] describes structural properties of the function g, several of which are explicit or implicit in [5]. We state them below.

- The functions g and g' are continuous, concave and piecewise linear and have no more than $n-1$ breakpoints. The function g is non-increasing in b.
- Under a non-degeneracy assumption on the graph, which is easy to ensure, the following holds. If b is not a breakpoint then there is a unique edge set A such that $g_A(b) = g(b)$. If b is a breakpoint then there are exactly two edge sets A, B such that $g_A(b) = g_B(b)$.
- If b_0 is a breakpoint of g' induced by edge sets A and B with $\kappa(A) > \kappa(B)$ then $B \subset A$. In particular $A \setminus B$ is contained in some connected component of $G' = (V, E \setminus B)$.
- Let b_0 be a breakpoint of g' induced by edge set A. Then the next breakpoint is induced by the edge set which is the solution to the strength problem on the smallest strength component of $G' = (V, E \setminus A)$.

The above properties show that the breakpoints induce a sequence of partitions of V which are refinements. Alternatively we consider the sequence of edge sets $A_1, A_2, \ldots,$ obtained by the following algorithm. We will assume that G is connected. Let $A_0 = \emptyset$. Given A_i we obtain $A_{i+1} \supseteq A_i$ as follows. Let $G_i = (V, E \setminus A_{i-1})$. If G_i has no edges we stop. Otherwise let C_{i+1} be the minimum strength connected component of G_i and B_{i+1} be a maximal minimum strength edge set of C_{i+1}. We define $A_{i+1} = A_i \cup B_{i+1}$, and τ_i to be the strength of the component C_{i+1}. That is, $\tau_i = \frac{c(A_i) - c(A_{i-1})}{\kappa(A_i) - \kappa(A_{i-1})}$. The process stops when $A_h = E$. Let $\mathcal{P}_i$ denote the partition of V induced by A_i. Note that $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by replacing C_{i+1} by a minimum strength partition of C_{i+1}, thus $\mathcal{P}_{i+1}$ is a refinement of $\mathcal{P}_i$ and $\mathcal{P}_h$ consists of singleton nodes. Note that Thorup's ideal tree packing is also based on the same recursive decomposition. Let the ith breakpoint of g' to be b_i. It was shown that $b_i = \tau_i$ is precisely the ith breakpoint of the function g' [24].

Ravi and Sinha obtained a 2-approximation for k-Cut as follows. Given the preceding decomposition of G they consider the smallest j such that $|\mathcal{P}_j| \geq k$. If $|\mathcal{P}_j| = k$ they output it and can argue that it is an optimum solution. Otherwise they do the following. Recall $\mathcal{P}_j$ is obtained from $\mathcal{P}_{j-1}$ by replacing the component C_j in $G - A_{j-1}$ by a minimum strength decomposition of C_j. Let $k' = k - |\mathcal{P}_{j-1}|$. Consider the minimum strength partition of C_j and let $H_1, H_2, \ldots, H_{k'}$ be the connected components of the partition with the smallest shores. Output the cut $A_{j-1} \cup (\cup_{\ell=1}^{k'} \delta(H_\ell))$.

6.1 An optimum LP solution from the decomposition

Given k, as before let j be the smallest index such that $\kappa(A_j) \geq k$. We consider the following solution to the LP:

- $x(e) = 1$ for each $e \in A_{j-1}$.
- $x(e) = \alpha$ for each $e \in A_j \setminus A_{j-1}$, where $\alpha = \frac{k - \kappa(A_{j-1})}{\kappa(A_j) - \kappa(A_{j-1})}$.
- $x(e) = 0$ for each $e \in E \setminus A_j$.

▶ **Lemma 16.** *The solution x is feasible and has objective value*

$$c(A_{j-1}) + \alpha c(B_j) = c(A_{j-1}) + (k - \kappa(A_{j-1}))\, b_j.$$

Proof. Let T be any spanning tree. We want to show that $\sum_{e\in T} x(e) \geq k-1$. For each j, let $\kappa_j = \kappa(A_j)$, and let $\ell_j = |T \cap A_j|$. Then T has ℓ_{j-1} edges of value $x(e) = 1$, and $\ell_j - \ell_{j-1}$ edges of value α. We have

$$\begin{aligned}\sum_{e\in T} x(e) &= \ell_{j-1} + (\ell_j - \ell_{j-1})\alpha \\ &\geq \kappa_{j-1} - 1 + (\kappa_j - \kappa_{j-1})\alpha = k-1,\end{aligned}$$

where we observe that the RHS of the first line is decreasing in both ℓ_j and ℓ_{j-1}, $\ell_j \geq \kappa_j - 1$, and $\ell_{j-1} \geq \kappa_{j-1} - 1$. To calculate the objective value, we have

$$\sum_{e\in E} x(e) = \sum_{e\in A_{j-1}} c(e) + \sum_{e\in A_j \setminus A_{j-1}} \alpha c(e) = c(A_{j-1}) + \alpha c(B_j)$$ ◀

The harder part is:

▶ **Lemma 17.** *The solution x attains the optimum value to the LP relaxation.*

Proof. We prove the claim by constructing a dual solution of equal value. See Figure 3 for the dual LP.

Recall the definitions of $\mathcal{P}_i$, A_i, B_i, and C_i from above. For each i, let $\kappa_i = \kappa(A_i) = |\mathcal{P}_i|$ be the number of components in the ith partition. The sequence $b_1 < b_2 < \cdots < b_j$ is the breakpoints for g', which is also the strengths of the components $C_1, C_2, \ldots, C_j$. Let Q_i be the partition of C_i corresponding to B_i. An ideal tree packing, following [29], is a convex combination of trees $p : \mathcal{T}(G) \to [0,1]$ s.t. $\sum_T p_T = 1$ with the following properties.

1. For each i, every tree T supported by p induces a tree in the graph $G/\mathcal{P}_i$ obtained by contracting each component of $\mathcal{P}_i$.
2. For each i and each edge $e \in B_i$, p induces $\sum_{T\ni e} p_T = c(e)/b_i$ on e.

Every graph has an ideal tree packing, and (for example) can be constructed recursively as follows. For each C_i, we write each B_i as a sum of b_i (units of fractional) trees in C_i/Q_i (which holds because B_i is a minimum strength cut), and scale it down to a distribution p' of trees in C_i/Q_i with $\sum_{T\ni e} p'_T = c(e)/b_i$ on each edge in B_i. An ideal tree packing now corresponds to the distribution where we take the union of one sampled spanning tree from (the distribution of) each C_i/Q_i.

Let $p : \mathcal{T}(G) \to [0,1]$ be an ideal tree packing. To construct our dual solution, we define nonnegative edge potentials $z(e) \geq 0$ and a tree packing $y(t) \geq 0$ (packing into $c + z$) s.t.

$$\begin{aligned} y_T &= b_j p(T) \qquad \text{for all } T \in \mathcal{T}(G), \\ c(e) + z(e) &= \begin{cases} \frac{b_j}{b_i} c(e) & \text{for all } e \in B_i \text{ for } i < j \\ c(e) & \text{otherwise.} \end{cases}\end{aligned}$$

We first claim that (y, z) is feasible in the dual LP; that is, y is a feasible tree packing w/r/t the augmented capacities $c + z$. Observe that for any edge e, y uses capacity b_j times the capacity by p. We need to show the capacity used by y along any edge e is at most $c(e) + z(e)$. We have two cases.

1. If $e \in B_i$ for some $i < j$, then p uses capacity $\frac{c(e)}{b_i}$. In turn, y uses capacity $\frac{b_j}{b_i} c(e)$. By choice of $z(e)$, we have $c(e) + z(e) = \frac{b_j}{b_i} c(e)$, as desired.

2. If $e \in E \setminus A_{j-1}$, then p uses capacity at most $\frac{c(e)}{b_j}$. In turn, y uses capacity at most $\frac{b_j}{b_i}c(e)$. But $b_i \leq b_j$, so the capacity used by y is $\leq c(e)$.

We now analyze the objective value of our dual solution. We first observe that since each tree supported by y is a tree in $G/\mathcal{P}_j$, we have

$$\begin{aligned}(k-1)\sum_T y_T &= \frac{k-1}{\kappa_j - 1}\sum_T y_T|T \cap A_j| = \frac{k-1}{\kappa_j - 1}\sum_{e\in A_j}\sum_{T\ni e} y_T \\ &= \frac{k-1}{\kappa_j - 1}b_j \sum_{i\leq j}\frac{1}{b_i}\sum_{e\in B_i} c(e) = \frac{k-1}{\kappa_j - 1}b_j \sum_{i\leq j}\kappa_i - \kappa_{i-1} \\ &= \frac{k-1}{\kappa_j - 1}b_j(\kappa_j - 1) = (k-1)b_j.\end{aligned}$$

On the other hand, when subtracting out the augmented capacities, we have

$$\begin{aligned}\sum_{e\in E} z(e) &= \sum_{i<j}\sum_{e\in B_i}\left(\frac{b_j}{b_i} - 1\right)c(e) = b_j\left(\sum_{i<j}\frac{1}{b_i}\sum_{e\in B_i} c(e)\right) - \sum_{e\in A_{j-1}} c(e) \\ &= b_j\sum_{i<j}(\kappa_i - \kappa_{i-1}) - \sum_{e\in A_{j-1}} c(e) = b_j\,(\kappa_{j-1} - 1) - \sum_{e\in A_{j-1}} c(e)\end{aligned}$$

Thus the total objective value of our solution, as a function of b_j, is

$$(k-1)\sum_T y_T - \sum_{e\in E} z(e) = (k - \kappa_{j-1})\,b_j + \sum_{e\in A_{j-1}} c(e),$$

as desired. ◀

▶ Remark. One can also verify the optimality of (x, y, z) in the proof above by complimentary slackness conditions. Recall that x and (y, z) satisfy the complimentary slackness conditions if

1. $z_e > 0$ only if $x_e = 1$.
2. $y_T > 0$ only if $\sum_{e\in T} x_e = k - 1$.
3. $x_e > 0$ only if $\sum_{T\ni e} y_e = c_e + z_e$.

We address these individually.

1. $z_e > 0$ only if $e \in B_i$ for some $i < j$. In this case, $e \in A_{j-1}$ so $x_e = 1$.
2. If $y_T > 0$ then T is in the support of the ideal tree packing. In particular, T contains exactly $\kappa_j - 1$ edges from A_j and $\kappa_j - 1$ edges from A_{j-1}, so we have

$$\sum_{e\in T} x_e = \sum_{e\in T\cap A_{j-1}} 1 + \sum_{e\in T\cap(A_j\setminus A_{j-1})}\frac{k - \kappa_{j-1}}{\kappa_j - \kappa_{j-1}} = \kappa_{j-1} - 1 + k - \kappa_{j-1} = k - 1,$$

   as desired.
3. If $x_e > 0$, then $e \in B_i$ for some $i \leq j$, so y uses $\frac{b_j}{b_i}c(e) = c(e) + z(e)$ units of capacity of e, as desired.

6.2 Implications of the characterization

We now outline some implications of the preceding characterization of the optimum LP solution.

Ravi and Sinha showed that Lagrangian relaxation lower bound is no weaker than the one provided by LP relaxation. Here we show that they are equivalent.

▶ **Theorem 18.** *The Lagrangian relaxation value is the same as the LP value.*

Proof. Let $\kappa_i = \kappa(A_i) = |\mathcal{P}_i|$ be the number of components after removing A_i. If $\kappa_j = k$ for some j, then the Lagrangian relaxation value is the min k-cut value. Hence it matches the LP value. Otherwise, assume $\kappa_{j-1} < k < \kappa_j$. Since g' is concave, continuous and piecewise linear, one can see that the function g' maximizes at one of the breakpoint, and it is precisely b_j. Indeed, $\kappa_j > k$, so $b_{j+1}(k - \kappa_j) < 0$ is a negative slope. We have

$$g'(b_j) = c(A_{j-1}) - b_j(\kappa_{j-1} - 1) + b_j(k-1) = c(A_{j-1}) + (k - \kappa_{j-1})b_j.$$

This is precisely the value of the LP in Lemma 16. ◀

The preceding also gives yet another proof that the integrality gap of the LP is $2(1 - 1/n)$.

Second, as we saw, for any value of k, an optimum dual solution to the k-Cut LP can be derived from the ideal tree packing [28, 29]. The last issue is the connection between greedy tree packing and the dual LP. At the high-level it is tempting to conjecture that greedy tree packing is essentially approximating the dual LP via the standard MWU approach. Proving the conjecture formally may require a fair amount of technical work and we leave it for future work. We believe that some insights obtained in [23] could be useful in this context; [23] recasts the LP relaxation for k-Cut into a pure covering LP, and the dual as a pure packing LP that packs forests instead of trees.

References

1. Ajit Agrawal, Philip Klein, and R Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.
2. Francisco Barahona. On the k-cut problem. *Operations Research Letters*, 26(3):99–105, 2000.
3. Chandra Chekuri, Sudipto Guha, and Joseph Naor. The Steiner k-cut problem. *SIAM Journal on Discrete Mathematics*, 20(1):261–271, 2006.
4. Chandra Chekuri and Kent Quanrud. Near-linear time approximation schemes for some implicit fractional packing problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 801–820. SIAM, 2017.
5. William H Cunningham. Optimal attack and reinforcement of a network. *Journal of the ACM (JACM)*, 32(3):549–561, 1985.
6. Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto-Rodriguez, and Frances A. Rosamond. Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems. *Electr. Notes Theor. Comput. Sci.*, 78:209–222, 2003.
7. Satoru Fujishige. Theory of principal partitions revisited. In *Research Trends in Combinatorial Optimization*, pages 127–162. Springer, 2009.
8. Harold N. Gabow and K. S. Manu. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming*, 82(1):83–109, June 1998.
9. Michel X Goemans and David P Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.
10. O. Goldschmidt and D.S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Mathematics of Operations Research*, pages 24–37, 1994.
11. Anupam Gupta, Euiwoong Lee, and Jason Li. Faster Exact and Approximate Algorithms for k-Cut. In *Proceedings of IEEE FOCS*, 2018.

12 Monika Henzinger, Satish Rao, and Di Wang. Local Flow Partitioning for Faster Edge Connectivity. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1919–1938. SIAM, 2017.

13 David R Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, February 1995.

14 David R. Karger. Minimum Cuts in Near-linear Time. *J. ACM*, 47(1):46–76, January 2000. doi:10.1145/331605.331608.

15 David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

16 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic Global Minimum Cut of a Simple Graph in Near-Linear Time. 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 665–674. ACM, 2015.

17 Vladimir Kolmogorov. A faster algorithm for computing the principal sequence of partitions of a graph. *Algorithmica*, 56(4):394–412, 2010.

18 Pasin Manurangsi. Inapproximability of Maximum Edge Biclique, Maximum Balanced Biclique and Minimum k-Cut from the Small Set Expansion Hypothesis. In *Proc. of ICALP*, volume 80 of *LIPIcs*, pages 79:1–79:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

19 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1-6):583–596, 1992. doi:10.1007/BF01758778.

20 Hiroshi Nagamochi and Yoko Kamidoi. Minimum cost subpartitions in graphs. *Information Processing Letters*, 102(2):79–84, 2007. doi:10.1016/j.ipl.2006.11.011.

21 J Naor and Yuval Rabani. Tree Packing and Approximating k-Cuts. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, volume 103, page 26. SIAM, 2001.

22 H. Narayanan. The principal lattice of partitions of a submodular function. *Linear Algebra and its Applications*, 144:179–216, 1991.

23 Kent Quanrud. Fast and Deterministic Approximations for k-Cut. *CoRR*, abs/1807.07143, 2018. arXiv:1807.07143.

24 R Ravi and Amitabh Sinha. Approximating k-cuts using network strength as a lagrangean relaxation. *European Journal of Operational Research*, 186(1):77–90, 2008.

25 Huzur Saran and Vijay V. Vazirani. Finding k-Cuts Within Twice the Optimal. *SIAM J. Comput.*, 24(1):101–108, February 1995.

26 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

27 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, July 1997. doi:10.1145/263867.263872.

28 Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.

29 Mikkel Thorup. Minimum k-way cuts via deterministic greedy tree packing. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 159–166. ACM, 2008.

On Primal-Dual Circle Representations

Stefan Felsner
Institut für Mathematik, Technische Universität Berlin, Germany
felsner@math.tu-berlin.de
https://orcid.org/0000-0002-6150-1998

Günter Rote
Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany
rote@inf.fu-berlin.de
https://orcid.org/0000-0002-0351-5945

Abstract

The Koebe-Andreev-Thurston Circle Packing Theorem states that every triangulated planar graph has a contact representation by circles. The theorem has been generalized in various ways. The most prominent generalization assures the existence of a primal-dual circle representation for every 3-connected planar graph. We present a simple and elegant elementary proof of this result.

2012 ACM Subject Classification Human-centered computing → Graph drawings, Mathematics of computing → Graph algorithms

Keywords and phrases Disk packing, planar graphs, contact representation

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.8

Acknowledgements We thank Manfred Scheucher for implementing the algorithm and helping with the figures.

1 Introduction

For a 3-connected plane graph $G = (V, E)$ with face set F, a spherical *primal-dual disk representation* of G consists of two families of disks $(C_x : x \in V)$ and $(D_y : y \in F)$ on the sphere $\mathbb{S}^2$ with the following properties (see Figure 1).

- **(i)** The vertex-disks C_x have pairwise disjoint interiors.
- **(ii)** The face-disks D_y have pairwise disjoint interiors.

Moreover, for every edge $xx' \in E$ with dual edge yy' (i. e., y and y' are the two faces separated by xx'), the following holds:

- **(iii)** Circles C_x and $C_{x'}$ touch at a point p.
- **(iv)** Circles D_y and $D_{y'}$ touch at the same point p.
- **(v)** The common tangent of C_x and $C_{x'}$ in the point p is perpendicular to the common tangent of D_y and $D_{y'}$ in p.

▶ **Theorem 1.** *Every 3-connected plane graph G admits a primal-dual disk representation on the sphere. This representation is unique up to Möbius transformations.*

Given a primal-dual disk representation of a graph G, we can use stereographic projection to obtain a primal-dual circle representation in the plane. (In the plane, we stick to the more common terminology of *circle* packings, because a circle defines a unique disk; on the sphere, we have to specify which of the two parts bounded by a circle we mean, and therefore we speak of *disk* packings.) Changing the center of the stereographic projection leads to

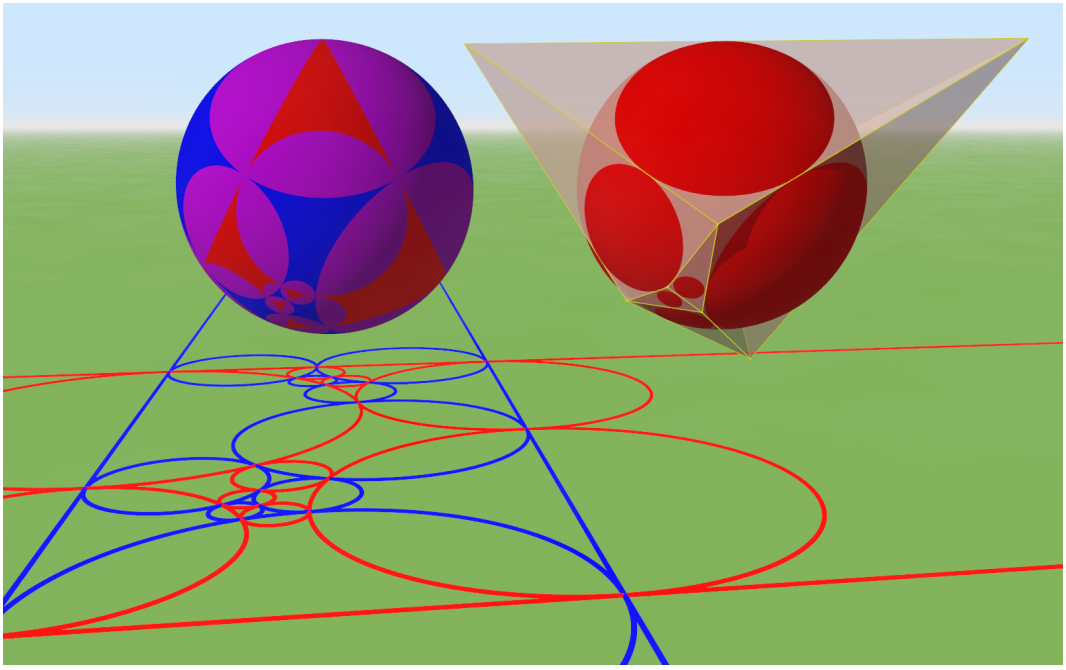


Figure 1 On the left, a primal-dual disk representation on the sphere, and its stereographic projection to the plane. The intersections of red primal disks with blue dual disks appear in purple. On the right, planes through the boundaries of the red disks define a polytope whose edges "cage" the sphere. The edge skeleton of this polytope is the dual graph (the touching graph of the blue disks).

different primal-dual circle representation in the plane. Figure 2 shows three primal-dual circle representations of K_4 in the plane where the projection center has been chosen as the center of one of the disks, the center of a digon formed by a primal-dual pair of intersecting disks, and the common point of four circles.

As a special case of the previous theorem we obtain the classical circle packing theorem:

▶ **Theorem 2.** *Every plane graph G admits a circle packing representation, i.e., it is the contact graph of a set of nonoverlapping disks in the plane.*

Our proof of Theorem 1 is constructive, in a sense: It computes a primal-dual circle representation in the plane by a limiting process. For the simplicity of the proof we choose the version where four of the circles are lines and all the other centers of circles are in the rectangle formed by these lines, see the right picture of Figure 2 and the larger example in Figure 3. The theorem then follows using an inverse stereographic projection.

Our proof combines ideas from an unpublished manuscript of Pulleyblank and Rote, from Brightwell and Scheinerman [6] and from Mohar [25]. All these proofs are based on an algorithm for iteratively improving estimates of the circle radii, whose idea goes back to Thurston [37, Section 13.7]. A distinguishing feature of our approach is the symmetric treatment of the primal and the dual family of circles. Four of the radii are already fixed at ∞, and this helps to reduce the graph-theoretic argument in the proof of convergence to a simple statement about the number of edges of a plane bipartite graph (Lemma 4) and a connectivity argument. The core of the proof requires only 1.5 pages and four chains of equations and inequalities. The layout of the "kites" obtained from the limits of the radii is based on an auxiliary result of independent interest (Lemma 5): when polygonal shapes are

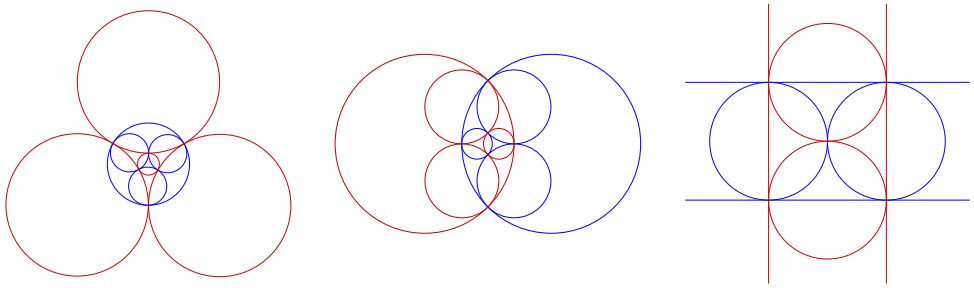


Figure 2 If we project the symmetric primal-dual disk representation of K_4 on $\mathbb{S}^2$ to the plane by stereographic projection, we get different primal-dual circle representations, depending on the center of projection. In the right picture we see that circles may degenerate to lines.

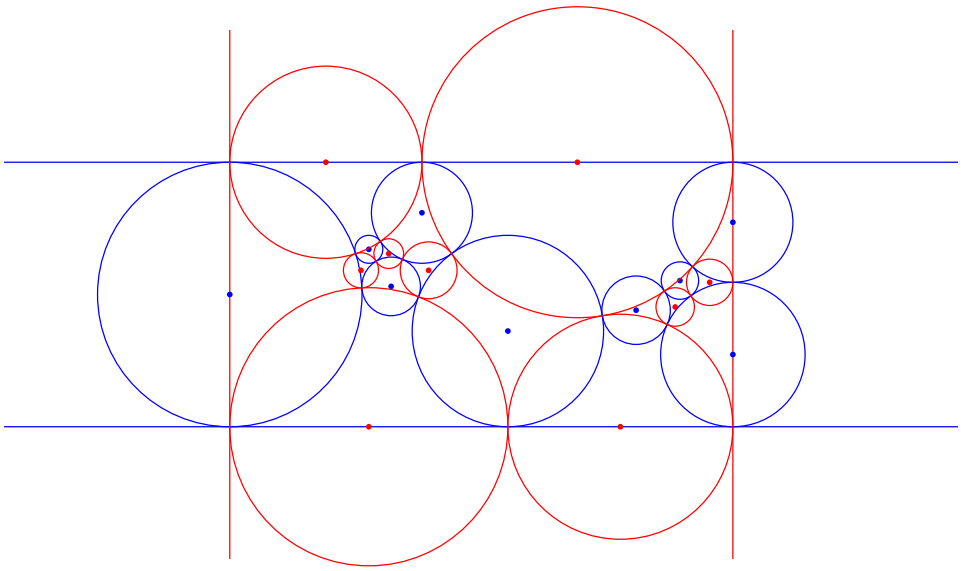


Figure 3 A larger example. Figure 1 includes a spacial image of this circle representation from a viewpoint on the left side and above the plane of Figure 3.

glued together along edges, local consistency conditions are sufficient to guarantee that these shapes form an overlap-free tiling.

Our simple and elementary proof of Koebe's Theorem, respectively its primal-dual version, is suited for a presentation in a class on Graph Theory, Discrete Geometry, Computational Geometry, or Graph Drawing.

In the next section we give a rather comprehensive account of the history of the theorem and mention some of its applications. The proof of the theorem is given in Section 3. Section 4 is devoted to the proof of Lemma 5.

2 History and Applications of the Theorem

In graph theory the study of circle contact representations can be traced back to the 1970's and 1980's; the term "coin representation" was used there. Wegner [39] and Jackson and Ringel [20] conjectured that every plane graph has a circle representation. The problem was popularized by Ringel [28], who also included it in a textbook from 1990 [19]. In a note written in 1991 [31], Sachs mentions that he found a proof of the circle packing theorem

which was based on conformal mappings. This eventually lead him to the discovery that the theorem had been proved by Koebe as early as 1936 [21].

Thurston, in the context of the study of 3-manifolds, proved that any triangulation of the sphere has an associated "circle packing" which is unique up to Möbius transformations [37, Sections 13.6–7]. Thurston noted that this result was already present in earlier work of Andreev [2]. Nowadays the result is commonly referred to as the *Koebe-Andreev-Thurston Circle Packing Theorem*. At a conference talk in 1985, Thurston suggested connections between circle packings and the Riemann Mapping Theorem. A precise version was obtained by Rudin and Sullivan [29]. This line lead to the study of discrete analytic functions and other aspects of discrete differential geometry, see to [35, 36, 5] for more on the topic.

In the early 1990's new proofs of the circle packing theorem where found. Colin de Verdière [7] gave an existential proof based on 'invariance of domain'; this proof can also be found in [27, Chapter 8] and in the primal-dual setting in an early draft of a book manuscript by Lovász's [22]. Colin de Verdière [8] gave another proof, which is based on the minimization of a convex function, and he extended circle packings to more general surfaces. Pulleyblank and Rote (unpublished) and Brightwell and Scheinerman [6] gave proofs of the primal-dual version (Theorem 1) based on an iterative algorithm, similar to the proof given in this note. Mohar [24] strengthened the result and proposed an iterative approach that obtains an ε-approximation for the radii and centers in time polynomial in the size of the graph and $\log(1/\varepsilon)$.

Primal-dual circle representations yield *simultaneous orthogonal drawings* of G and its dual G^*, i. e., straight-line drawings of G and G^* such that the outer vertex of G^* is at infinity and each pair of dual edges is orthogonal. The existence of such drawings was conjectured by Tutte [38]. In fact, it follows from Tuttes "spider-web" embedding method via the Maxwell-Cremona correspondence, which produces a convex piecewise linear surface in $\mathbb{R}^3$ that vertically projects onto the drawing of G. Polarity will then yield a straight-line embedding of G^* with edges orthogonal to edges of G, see [26] or [30, Section 5]. However, unlike the embeddings implied by the circle theorem, primal-dual edge pairs in this embedding may not intersect.

Another consequence of primal-dual circle representations is known as the *Cage Theorem*. It says that every 3-connected planar graph is the skeleton of a convex 3-polytope such that every *edge* of the polytope is tangent to a given sphere. This strengthening of the Steinitz Theorem is easily derived from Theorem 1, see Figure 1. The Cage Theorem was generalized by Schramm [32], who showed that the sphere that is caged can be replaced by any smooth strictly convex body.

A stunning generalization of the Circle Packing Theorem is the *Monster Packing Theorem* of Schramm [34]. The statement (slightly simplified) is as follows: if each vertex v of a planar triangulation G has a prescribed convex prototype P_v, then there is a contact representation of G where each vertex is represented by a nonnegative homothet of its prototype. Some of these homothets may degenerate to points, but when the prototypes have a smooth boundary, such degeneracies are excluded. Contact representations of planar graphs with other shapes than circles have received quite some attention over the years, for example with triangles [9, 17, 1], rectangles and squares [13, 33], and pentagons and k-gons [16, 15].

The Circle Packing Theorem has been used to prove *separator theorem*s. In particular, every planar graph with n vertices can be partitioned into components with at most $n/2$ vertices by removing $O(\sqrt{n})$ vertices. The approach was pioneered by Miller and Thurston and generalized to arbitrary dimensions by Miller, Teng, Thurston, and Vavasis [23]. The planar case is reviewed in [27, Chapter 8]. A slightly simpler proof was given by Har-Peled [18].

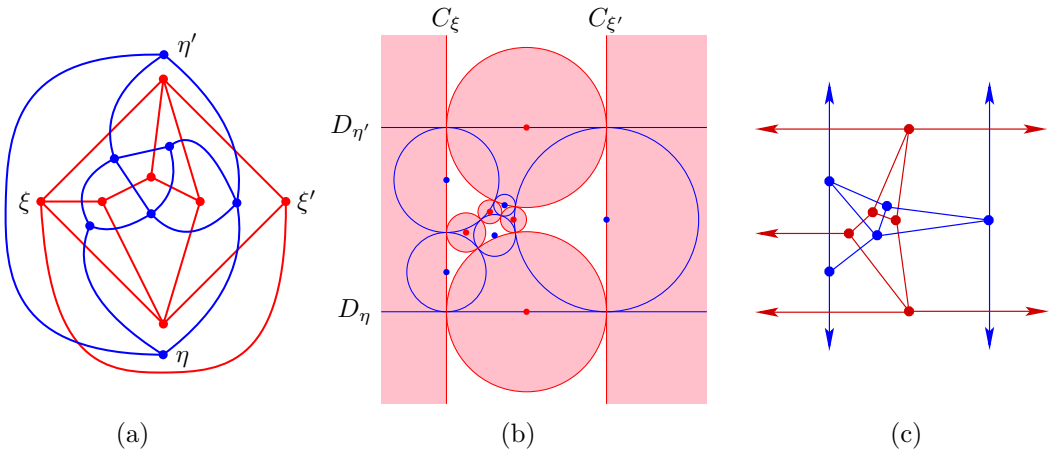


Figure 4 (a) A graph and its dual. (b) A cross-centered primal-dual circle packing for this graph. The areas of the primal disks are shaded. The two vertices ξ and ξ' are represented by "degenerate disks": disjoint halfplanes bounded by C_ξ and $C_{\xi'}$, which "touch at infinity". (c) The straight-line drawing of the two graphs induced by the circle packing; the centers ξ and ξ' of the degenerate disks lie infinitely far away to the left and to the right. The edge $\xi\xi'$ is not represented at all. The same holds for η and η' and the dual edge between them.

Bern and Eppstein [4, 11] relate circle packings to mesh generation techniques.

Not surprisingly, the theorem also has applications in Graph Drawing. Eppstein [12] used circle representations to prove that every planar graph with maximum degree 3 has a *Lombardi drawing*: a drawing in which the edges are drawn as circular arcs, meeting at equal angles at each vertex. Felsner, Igamberdiev, Kindermann, Klemz, Mchedlidze, and Scheucher [14] used circle representations to show that 3-connected planar graphs have planar *strongly monotone drawings*, i. e., straight-line drawings such that for any two vertices u, v there is a path which is monotone with respect to the connecting line of u and v.

3 Primal-Dual Circle Representation: The Proof

Let $G = (V, E)$ be a 3-connected plane graph with face set F and let $\xi\xi'$ be an edge in E with dual edge $\eta\eta'$, i. e., η and η' are the two faces on the sides of $\xi\xi'$. A *cross-centered primal-dual circle representation* of G with *central cross* $\xi\xi'$, $\eta\eta'$ consists of two vertical lines C_ξ and $C_{\xi'}$, two horizontal lines D_η and $D_{\eta'}$, and two families of circles $(C_x : x \in V \setminus \{\xi, \xi'\})$ and $(D_y : y \in F \setminus \{\eta, \eta'\})$ with the following five properties, sees Figures 3 and 4b for examples:

- **(i)** The vertex-circles C_x have pairwise disjoint interiors and are contained in the vertical strip between C_ξ and $C_{\xi'}$.
- **(ii)** The face-circles D_y have pairwise disjoint interiors and are contained in the horizontal strip between D_η and $D_{\eta'}$.

Moreover, for every edge $xx' \in E$ with $xx' \neq \xi\xi'$ and with dual edge yy' (i. e., y and y' are the two faces separated by xx'), the following holds:

- **(iii)** C_x and $C_{x'}$ are tangent at a point p with common tangent line $t_{xx'}$.
- **(iv)** D_y and $D_{y'}$ are tangent at the same point p with common tangent line $t_{yy'}$.
- **(v)** The lines $t_{xx'}$ and $t_{yy'}$ are orthogonal.

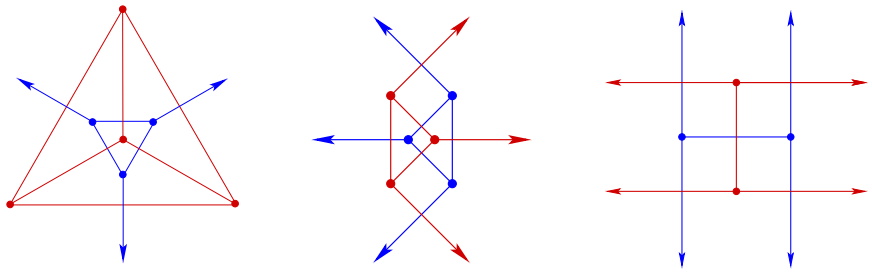


Figure 5 Three primal-dual straight-line drawings of K_4. They correspond to the primal-dual circle representations of Figure 2.

▶ **Theorem 3.** *Every 3-connected plane graph G admits a cross-centered primal-dual circle representation. Moreover, for a given central cross $\xi\xi', \eta\eta'$, this representation is unique up to scaling, translation, and horizontal or vertical reflections.*

Theorem 1 follows from Theorem 3 via inverse stereographic projection.

We give first an outline of the proof. A primal-dual circle representation of G induces a straight-line drawing of G and a straight-line drawing of the dual. Superimposing the two drawings yields a plane drawing whose faces are special quadrangles called kites, see Figures 6 and 7. After guessing radii for the circles, the shapes of the kites are determined. It is then checked whether the angles of kites meeting at a vertex sum up to 2π. If at some vertex the angle sum differs from 2π, the radii are changed to correct the situation. The process is designed to make the radii converge and to make the sum of angles meet the intended value at each vertex. The second part of the proof consists of showing that the kites corresponding to the final radii can be laid out to form a tessellation, thus giving the centers of a primal-dual circle representation of G.

Proof of Theorem 3. Given a cross-centered primal-dual circle representation of G we can use the centers of the circles C_x to obtain a planar straight-line drawing of G, see Figure 4c. Edges containing ξ or ξ' are represented by horizontal rays to the left and right respectively. The edge $\xi\xi'$ is missing. Similarly, the centers of the circles D_y yield a planar straight-line drawing of G^* with edges containing one of η and η' being represented by vertical rays.

For example, from the primal-dual circle representations of K_4 of Figure 2, we obtain plane straight-line drawings of K_4 and its dual that are displayed in Figure 5. The rightmost of these drawings corresponds to a cross-centered primal-dual circle representation of K_4.

3.1 Kites

If we overlay the drawings of G and G^*, we get a partition of the plane into *kites*: quadrilaterals with right angles at two opposite vertices and a line of symmetry through the other two vertices. Figure 6 shows an example, and Figure 7 shows a generic kite. In the cross-centered case, there are *degenerate kites*: rectangular strips that are unbounded in one direction. They have a vertex with a 180°-angle in the midpoint of the only bounded edge. In addition, we have four quadrants, which can be regarded as *exceptional kites*. The bounded kites fill a rectangle between C_ξ, $C_{\xi'}$, D_η, and $D_{\eta'}$.

The kites are in bijection with the incident pairs (x, y), where x is a primal vertex and y is a dual vertex. Since the involved circles or lines intersect orthogonally, the kite of x and y is completely determined by the radii r_x of C_x and r_y of D_y. (In the case of a line the radius

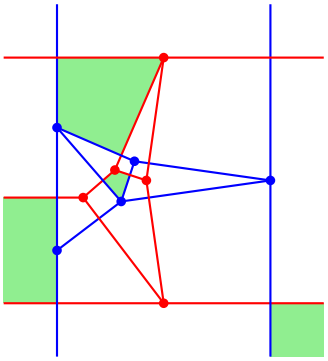


Figure 6 The tessellation of the plane into kites obtained from the example in Figure 4c. Four kites are shaded, among them a degenerate kite (semi-infinite strip) and an exceptional kite (quadrant).

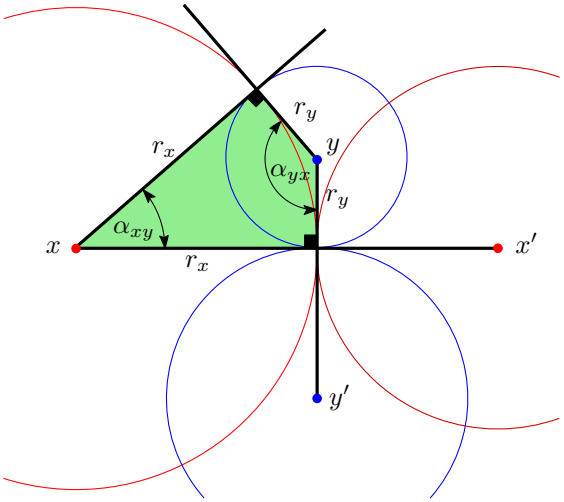


Figure 7 The kite corresponding to the incident vertex-face pair x,y.

is ∞.) For bounded kites, the angles at x and y are given by

$$\alpha_{xy} = 2\arctan\frac{r_y}{r_x} \quad \text{and} \quad \alpha_{yx} = 2\arctan\frac{r_x}{r_y}. \tag{1}$$

We extend these formulas to degenerate kites by taking the limits:

$$\alpha_{uw} = \begin{cases} 0, & \text{if } r_w \neq \infty \text{ and } r_u = \infty \\ \pi, & \text{if } r_w = \infty \text{ and } r_u \neq \infty \end{cases} \tag{2}$$

Then we have

$$\alpha_{uw} + \alpha_{wu} = \pi$$

for all pairs (u, w) forming a bounded or degenerate kite. We don't define the angles for the four exceptional kites because this would involve the undetermined expression $\frac{\infty}{\infty}$.

3.2 The Angle Graph

The number and combinatorial structure of the kites is captured by the angle graph. The *angle graph*, or *vertex-face incidence graph*, of a plane graph $G = (V, E)$ is the graph $G^\diamond = (U, K)$ whose node set $U = V \cup F$ represents both the vertices and faces of G, see Figure 8b. Its edges xy are the pairs with $x \in V$ and $y \in F$ that are incident in G, i. e., x is a vertex on the boundary of the face y. These edges are in bijection with the kites. The graph $G^\diamond$ is plane and bipartite. Its faces corresponds to the edges of G, and they are 4-gons, i. e., $G^\diamond$ is a quadrangulation. We choose the face $f_o = \xi\eta\xi'\eta'$ containing the four elements of the central cross as the outer face of $G^\diamond$. We denote its nodes by $U_o = \{\xi, \xi', \eta, \eta'\}$ and the remaining nodes by $U_{\text{in}} = U \setminus U_o$. We denote the four edges of the outer face by $K_o = \{\xi\eta, \eta\xi', \xi'\eta', \eta'\xi\}$.

An important property of the angle graph is that it cannot have a separating 4-cycle: If the nodes $xyx'y'$ with $x, x' \in V$ and $y, y' \in F$ would form some separating 4-cycle in $G^\diamond$, then x, x' would be a separating vertex pair in G, contradicting the 3-connectedness assumption for G.

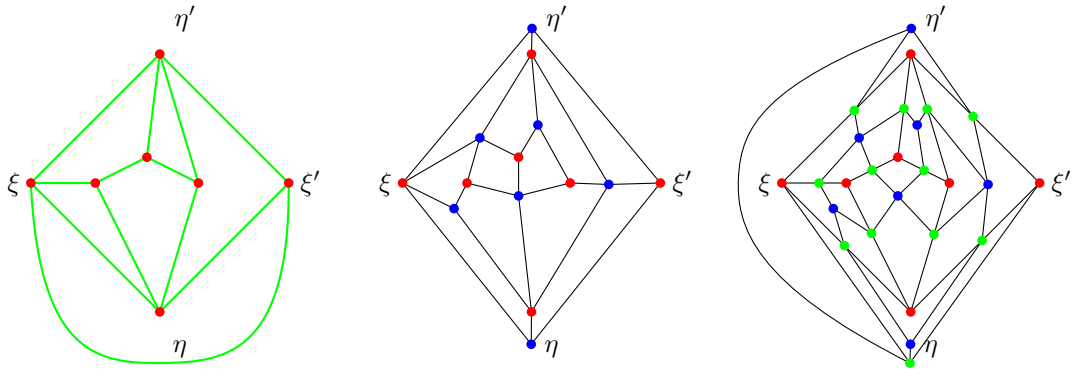


Figure 8 (a) The plane graph G from Figure 4a, (b) its angle graph $G^\diamond$, (c) the primal-dual completion $(G^\diamond)^\diamond$. The faces of this graph represent the kites, including the unbounded and exceptional kites.

We will need the following well-known basic fact about bipartite plane graphs, which is a consequence of Euler's formula. For completeness, we include the detailed proof in Appendix A.

▶ **Lemma 4.** *A simple bipartite plane graph with $|S| \geq 4$ nodes has at most $|E| \leq 2|S| - 4$ edges, with equality if and only if the graph is connected and every face is a quadrilateral with four distinct vertices.*

In particular, $G^\diamond$ contains $|K| = 2|U| - 4$ edges.

3.3 Angle Sums

We now come to the core of the argument. A hypothetical primal-dual circle representation of G contains a point for each $u \in U_{\text{in}}$. This point is fully surrounded by its incident kites. Hence, for every $u \in U_{\text{in}}$ we have:

$$\sum_{w\colon uw \in K} \alpha_{uw} = 2\pi \tag{3}$$

We now look at an arbitrary assignment $r\colon U_{\text{in}} \to \mathbb{R}_{>0}$ of radii. Additionally, we define $r_u = \infty$ for each $u \in U_o$. We can then form the corresponding kites and compute the angles according to (1) and (2). In particular, by (2), the degenerate kites have the correct angles:

$$\alpha_{uw} = \pi \text{ and } \alpha_{wu} = 0 \text{ whenever } u \in U_{\text{in}} \text{ and } w \in U_o. \tag{4}$$

Denote the angle sum at $u \in U_{\text{in}}$ by $\alpha_u = \alpha_u(r) = \sum_{w\colon uw \in K} \alpha_{uw}$. We want to find radii r such that $\alpha_u(r)$ becomes equal to the *target angle* 2π for all $u \in U_{\text{in}}$ in order to fulfill (3). Later we will show that a collection of radii with this property induces a primal-dual circle representation.

We first show that any choice of radii attains the correct target angles *on average*:

$$\sum_{u \in U_{\text{in}}} \big(\alpha_u(r) - 2\pi\big) = 0 \tag{5}$$

This follows from the following computation:

$$\sum_{u\in U_{\text{in}}} \alpha_u(r) = \sum_{\substack{uw\in K\\ u,w\in U_{\text{in}}}} (\alpha_{uw}+\alpha_{wu}) + \sum_{\substack{uw\in K\\ u\in U_{\text{in}},w\in U_o}} \alpha_{uw} = \sum_{\substack{uw\in K\\ u,w\in U_{\text{in}}}} \pi + \sum_{\substack{uw\in K\\ u\in U_{\text{in}},w\in U_o}} \pi = \pi|K\setminus K_o|$$
$$= \pi(|K|-4) = \pi(2|U|-8) = 2\pi(|U|-4) = 2\pi|U_{\text{in}}|$$

As a consequence, whenever $\alpha_u(r)\neq 2\pi$ for some u, the following two sets are both nonempty:

$$U_- = \{u\in U_{\text{in}}\colon \alpha_u(r)<2\pi\} \qquad \text{and} \qquad U_+ = \{u\in U_{\text{in}}\colon \alpha_u(r)>2\pi\}$$

If we increase the radius r_u of a node $u\in U_+$, leaving all remaining radii fixed, we observe from (1) that for every incident edge $uw\in K$, the angle α_{uw} decreases strictly to 0 as $r_u\to\infty$, with the possible exception of a single neighbor $w\in U_o$ with fixed angle $\alpha_{uw}=\pi$ according to (4). Hence, we can increase r_u to the unique value where $\alpha_u(r)=2\pi$.

3.4 Iteration and Convergence

The workhorse of the proof is the following infinite iteration.

repeat forever:
  for each $u\in U_{\text{in}}$:
    if $u\in U_+$ **then** increase r_u to reduce $\alpha_u(r)$ to 2π (6)

We will show that, for an arbitrary positive starting assignment, the radii converge to some limiting assignment $\hat{r}$, and this will imply that $\alpha_u(\hat{r})=2\pi$ for all $u\in U_{\text{in}}$.

Since radii can never decrease and every bounded monotone sequence is convergent, it is enough to show that the set of "divergent" nodes $D=\{u\in U : \lim r_u=\infty\}$ contains no other nodes than the four nodes of U_o. (The nodes $u\in U_o$ have $r_u=\infty$ fixed and are included in D by definition.)

The increase of r_u decreases the angle sum α_u, but not below 2π. It increases the angles at adjacent nodes, and it may hence cause some $w\in U_-$ to move to U_+. A transition from U_+ to U_-, however, is impossible. It follows that some node u_0 must belong to U_- indefinitely unless the iteration comes to a halt with $U_-=U_+=\emptyset$. Thus, as a consequence of the built-in behavior of the iteration, (a) U_- is disjoint from D from some time on, and (b) D is a proper subset of U.

Let us look at the subgraph $G^\diamond[D]$ of $G^\diamond$ induced by the divergent nodes. In order to apply Lemma 4, we will show that $G^\diamond[D]$ has at least $2|D|-4$ edges.

First, we wait for U_- to become disjoint from D. From that point onwards,

$$\sum_{u\in D\setminus U_o} \alpha_u(r) \;\geq\; \sum_{u\in D\setminus U_o} 2\pi \;=\; 2\pi|D\setminus U_o| = (2|D|-8)\pi. \tag{7}$$

On the other hand, if $u\in D$ and $w\in U\setminus D$, then α_{uw} converges to 0 according to (1). Thus, in addition to (7), the inequality $\alpha_{uw}\leq 1/|U|^2$ will eventually hold for each such edge. Bounding these edges separately from the others, we get the following inequality at this point of the iteration:

$$\sum_{u\in D\setminus U_o} \alpha_u(r) \;\leq\; |U|^2\cdot\frac{1}{|U|^2} + \sum_{\substack{\text{kite with } x,y\in D\\ x\notin U_o \text{ or } y\notin U_o}} (\alpha_{xy}+\alpha_{yx}) \;=\; 1+\sum_{\substack{xy \text{ edge of } G^\diamond[D]\\ xy\notin K_o}} \pi$$
$$= \; 1+(|E(G^\diamond[D])|-4)\pi, \tag{8}$$

where $E(G^\diamond[D])$ is the edge set of $G^\diamond[D]$. Comparing (7) and (8) gives $|E(G^\diamond[D])| \geq 2|D| - 4 - 1/\pi$ and therefore $|E(G^\diamond[D])| \geq 2|D| - 4$.

Since $U_o \subseteq D$ by definition and thus $|D| \geq 4$, we can apply Lemma 4. We conclude that $G^\diamond[D]$ is connected and its faces are simple 4-cycles.

The outer face of $G^\diamond[D]$ is the quadrilateral f_o formed by U_o. Our goal is to show that $D = U_o$ and $G^\diamond[D]$ consists just of the single 4-cycle f_o. Since $G^\diamond[D]$ is a proper subgraph of $G^\diamond$, $G^\diamond[D]$ has some face f that is not a face of $G^\diamond$. This face f is an inner face of $G^\diamond[D]$ because the outer face of $G^\diamond[D]$ agrees with f_o. Suppose for contradiction that f does not coincide with the interior face bounded by f_o. Then it would form a separating 4-cycle in $G^\diamond$: it would contain nodes of U both in its interior (because it is not a face of $G^\diamond$) and in its exterior (because some nodes of f_o lie there). Since separating 4-cycles are excluded, we have shown that $D = U_o$.

This means that all radii r_u for $u \in U_{\text{in}}$ converge to some limits, which we denote by $\hat{r}_u$. It follows that all angles α_{uw} and all angle sums $\alpha_u(r)$ converge as well, and by the working of the iteration (6), their limits $\alpha_u(\hat{r})$ are bounded by $\alpha_u(\hat{r}) \leq 2\pi$. Since $\sum_{u \in U_{\text{in}}} (\alpha_u(\hat{r}) - 2\pi) = 0$ by (5), we must have $\alpha_u(\hat{r}) = 2\pi$ for all $u \in U_{\text{in}}$.

3.5 Uniqueness

We show that the radii are unique up to scaling. Let r and r' be two vectors of radii such that $\alpha_r(u) = \alpha_{r'}(u) = 2\pi$ for all $u \in U_{\text{in}}$. Scaling allows to assume that $r_{u_0} = r'_{u_0}$ for some $u_0 \in U_{\text{in}}$. Consider the set $S = \{u \in U_{\text{in}} : r_u > r'_u\}$ and observe that $u_0 \in \bar{S} = U_{\text{in}} \setminus S$.

$$|S| \cdot 2\pi = \sum_{u \in S} \alpha_u(r) = \sum_{\substack{uw \in K \\ u,w \in S}} \big(\alpha_{uw}(r) + \alpha_{uw}(r)\big) + \sum_{\substack{uw \in K \\ u \in S, w \in U_o}} \alpha_{uw}(r) + \sum_{\substack{uw \in K \\ u \in S, w \in \bar{S}}} \alpha_{uw}(r)$$
$$= \sum_{\substack{uw \in K \\ u,w \in S}} 2\pi + \sum_{\substack{uw \in K \\ u \in S, w \in U_o}} \pi + \sum_{\substack{uw \in K \\ u \in S, w \in \bar{S}}} \alpha_{uw}(r)$$

Thus, the last sum has a constant value, independent of the radii r. However, if we change the radii from r to r', then, by (1), every term $\alpha_{uw}(r)$ in the last sum increases, because $r_u > r'_u$ and $r_w \leq r'_w$. This means that the set of edges over which the sum is taken must be empty. In other words, if $w \in \bar{S}$, then every neighbor $u \in U_{\text{in}}$ of w must also belong to $\bar{S}$. Since $u_0 \in \bar{S}$ and $G^\diamond[U_{\text{in}}]$ is connected, S must be empty. By a symmetric argument, the set $S' = \{u \in U_{\text{in}} : r'_u > r_u\}$ is empty as well, and this proves uniqueness of the radii r up to scaling.

The radii determine shape and size of the kites. Below we show that the kites can be laid out to form a tessellation of the plane. The line C_ξ is vertical, hence, the tessellation is unique up to scaling, translation, and horizontal or vertical reflection. Since the tessellation determines the circles, uniqueness carries over to the cross-centered primal-dual circle representation with fixed central cross.

3.6 Laying out the Kites

We now show that the kites defined by the limiting radii $\hat{r}$ can be laid out in the plane with the intended side-to-side contacts. Figure 9 illustrates this task. We will use Lemma 5 below, which warrants the existence of such a layout if certain local matching conditions are fulfilled. We invite the reader to skip forward and read the statement of Lemma 5 in Section 4. We apply this lemma to the graph H of the vertices and edges of the *bounded* kites, see Figures 6 and 9. This graph is a subgraph of the *primal-dual completion* of $G = (V, E)$ (which, by the

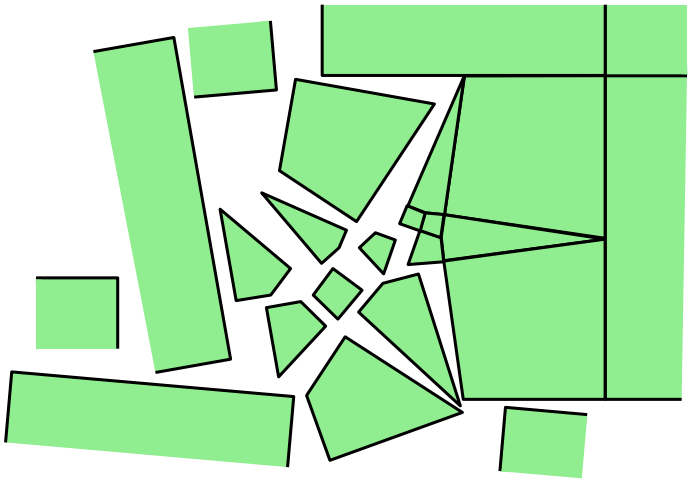


Figure 9 Laying out the kites.

way, is nothing but the angle graph $(G^\diamond)^\diamond$ of the angle graph of G, see Figure 8c). The nodes of H are vertices, faces and edges of G. Specifically, $V_H = (V \setminus \{\xi, \xi'\}) \cup (F \setminus \{\eta, \eta'\}) \cup (E \setminus \{\xi\xi'\})$, and the edges of H are the pairs $(z, e) \in ((V \setminus \{\xi, \xi'\}) \cup (F \setminus \{\eta, \eta'\})) \times (E \setminus \{\xi\xi'\})$ with z incident to the edge $e \in E$ in G. Each bounded face of H is a quadrilateral representing a bounded kite. It contains one node from V, one node from F, and two nodes from E.

The 3-connectivity of G and of G^* easily implies that H is 2-connected, as required for Lemma 5. (In fact, the first proof of Lemma 5 shows that connectedness of H is sufficient, provided that the outer face is a simple cycle.) We know that two adjacent kites fit together locally because they have the same edge lengths by construction: these lengths are defined by the same radius r_u. This is condition (iii) of Lemma 5.

Moreover, as we have shown, the kites around a vertex $u \in U_{\text{in}}$ form a complete angle of $\alpha_u(\hat{r}) = 2\pi$. Every right angle of a kite, if it is an interior node of H, is complemented by the right angles of three other kites to again form a complete angle of 2π. This is condition (i) of Lemma 5.

The vertices of H incident to the outer face of H are either points where one or two right angles of kites meet, forming an angle of 90° or 180°, or they are nodes $u \in U_{\text{in}}$ which are adjacent in $G^\diamond$ to some node $w \in U_o$, forming an angle $\alpha_{uw} = \pi$ by (4). Since this angle is not part of H, the incident angles in H around u sum up to $\alpha_u(\hat{r}) - \alpha_{uw} = 2\pi - \pi = \pi$. In summary, the angle sums of nodes incident to the outer face of H are either 90° or 180°, and thus condition (ii) for Lemma 5 is fulfilled, and moreover, the layout of the bounded kites must form a rectangle R. The unbounded kites can be attached edge by edge along the boundary of R. This yields the claimed cover of the whole plane.

3.7 Constructing the Circle Representation

Finally, we derive a cross-centered primal-dual circle representation from the layout of the kites. The kites induce a straight-line drawing of G and a straight-line drawing of the dual G^* with the edges incident to one node of U_o being rays and edges induced by U_o omitted. For every primal-dual pair xx', yy' of edges the point p where xx' and yy' meet is a right angle in each of the four involved kites. This implies (v).

For a node $u \in U_{\text{in}}$, consider the set of kites containing u. These kites can be put together in the cyclic order given by the rotation of u in $G^\diamond$ to form a polygon P_u surrounding u,

because $\alpha_u(\hat{r}) = 2\pi$. By the geometry of the kites, all edges incident to $u \in V \cap U_{\text{in}}$ have the same length $\hat{r}_u$, and the circle C_u of radius $\hat{r}_u$ centered at u is inscribed in P_u and touches P_u at the common corners of neighboring kites. For $u \in \{\xi, \xi'\}$, the polygon P_u obtained by gluing the corresponding unbounded kites is a halfplane and the vertical line C_u goes through the right-angle corners of the involved kites. From the incidences of the kites, and since the polygons P_u for $u \in V$ are pairwise disjoint, we obtain that the family $(C_x : x \in V)$ satisfies (i) and (iii).

Dually, the polygons P_u corresponding to $u \in F$ also tile the plane, and the family $(D_y : y \in F)$ satisfies Properties (ii) and (iv). This concludes the proof of Theorem 3. ◀

4 Tiling a Convex Polygon

The following lemma says that certain local consistency conditions around each vertex and along each edge are sufficient to guarantee a global nonoverlapping layout of faces with prescribed shapes.

▶ **Lemma 5.** *Let H be a 2-connected plane graph (possibly drawn with curved edges). For each bounded face f of H, a simple polygon P_f is given whose corners are labeled with the vertices from the boundary of f in the same cyclic order. Denote the corner of P_f labeled with v by p_{fv} and the angle of P_f at this corner by β_{fv}. For each vertex v, let F_v denote the set of incident bounded faces. We assume the following conditions:*

- **(i)** $\sum_{f \in F_v} \beta_{fv} = 2\pi$ *for every inner vertex v.*
- **(ii)** $\sum_{f \in F_v} \beta_{fv} \leq \pi$ *for every vertex v on the outer face.*
- **(iii)** $\|p_{fv} - p_{fw}\| = \|p_{gv} - p_{gw}\|$ *for every inner edge vw of H with incident faces f and g.*

Then there is a crossing-free straight-line drawing of H in which every bounded face f can be obtained from P_f by a rigid motion, i. e., translation and rotation.

Lemma 5 or similar statements have been used explicitly or implicitly in other situations, beyond the context of circle packings. For example in [16, 15] it is used in the context of contact representations with pentagons and k-gons. In fact, our second proof of Lemma 5 slightly generalizes a proof from [16].

We give two proofs of Lemma 5, a more geometric one and a more combinatorial one.

Proof 1. We proceed by induction on the number of interior vertices. The tool that we need is that every simple polygon can be subdivided into convex pieces, or if we want, into triangles, by inserting diagonals between its vertices.

To make the induction go through, we have to strengthen the assumption of the lemma and require that each polygon P_f is convex. This can be achieved by inserting diagonals and subdividing it into convex pieces. On the other hand, we don't require H to be 2-connected, and we even allow H to have multiple edges. (Showing beforehand that separating vertices or multiple edges cannot actually occur would be more tedious.) We do however maintain the requirement that H is connected and that the outer face is a simple cycle.

The inductive step proceeds as follows: If there is an interior vertex w, we take the k faces $f_1, \ldots, f_k$ incident to w and place the corresponding polygons $P_{f_1}, \ldots, P_{f_k}$ successively around the origin, see Figure 10. By condition i, they completely surround the origin. By convexity of the faces, each face is confined within its own sector, disjoint from the other sectors. Thus the faces don't overlap, and their union forms a simple polygon P that contains w in its interior. (It is star-shaped around w.) We triangulate P geometrically. We remove w from the graph H, and we insert the appropriate new edges into H, replacing the faces $f_1, \ldots, f_k$ by the new triangular faces, with the triangles as the corresponding polygons.

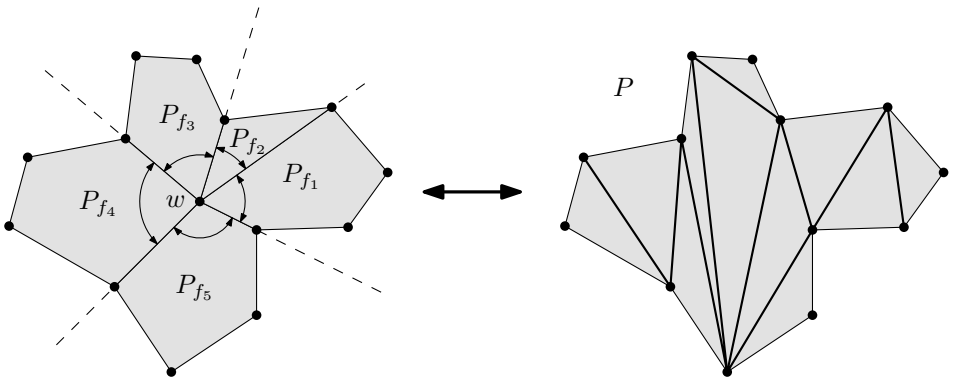


Figure 10 Triangulating the union of the faces surrounding a vertex w.

(Here is the point where multiple edges could conceivably be created.) When performing this replacement, by construction, the angle sum $\sum_{f \in F_v} \beta_{fv}$ remains the same around every vertex $v \neq w$ (conditions i and ii), and the new polygons have matching edge lengths, both among themselves and with the previously existing faces (condition iii). The resulting graph H' has one interior vertex less, and it is still connected, because the boundary vertices of P, which include all neighbors of w, are connected through the boundary edges of P. By induction, its faces can be laid out in the plane without overlap and with adjacent faces touching along their common edges. The triangular faces that were added form a polygon that is congruent to P. Cutting the polygon P into the faces $P_{f_1}, \ldots, P_{f_k}$ from which it was originally formed, we obtain the position for w and a drawing of the original graph H.

In the base case of the induction, there are no interior vertices. We simply merge adjacent polygons pairwise along their common interior edge. By condition ii, the two new resulting interior angles are $\leq \pi$. Hence the polygon resulting from each merge is again a convex polygon. In the end, we have a single convex polygonal face, and there is nothing left to prove. ◀

Proof 2. The proof proceeds in four steps. (A) In the first step, we define positions for every *face* Let H^* be the dual graph of H without the vertex corresponding to the outer face of H. Let S be a spanning tree of H^*. Then by (iii) we can glue the polygons P_f of all bounded faces f of H together along the edges of S. This determines a unique position for every polygon, up to a global motion.

(B) Since a vertex belongs to several faces, this layout might prescribe several inconsistent positions for the same *vertex*. In the second step, we that such contradictory constraints do not arise, and each vertex has a unique position. For the edges of S we already know that the polygons of the two incident faces touch in such a way that corners corresponding to the same vertex coincide. For the edges of the complement $\bar{S}$ of S we still need to show this. The set $\bar{S}$, considered as a subset of the edges of H, forms a forest. Let v be a leaf of this forest that is an inner vertex of H, and let e be the edge of $\bar{S}$ incident to v. Then for all incident edges $e' \neq e$ of v we already know that the polygons of the two incident faces of e touch in the right way. But then also the two polygons of the two incident faces of e touch in the right way because v fulfills property (i). Since the set of edges we still have to check remains a forest, we can iterate this process until all inner edges of H are checked. After gluing all the polygons P_f, every vertex v has an unambiguous position.

(C) Let P_o be the cycle formed by the boundary edges of H in this drawing. As the third step, we will show that P_o forms a convex polygon. We know from property (ii) that when we traverse P_0 with the interior on its left, we make only left turns, but it is conceivable

that P_0 makes several loops and intersects itself. We show that this is not the case. Let $H = (V, E)$ with face set F, and let V_o be the set of outer vertices of H. Denoting $d_o = |V_o|$, we claim that

$$\sum_{v \in V_o} \sum_{f \in F_v} \beta_{fv} = (d_o - 2)\pi. \tag{9}$$

To see this, we express the angle sum B over all polygons P_f in two different ways. Property (i) gives

$$B = \sum_{v \in V \setminus V_o} \sum_{f \in F_v} \beta_{fv} + \sum_{v \in V_o} \sum_{f \in F_v} \beta_{fv} = (|V| - d_o)2\pi + \sum_{v \in V_o} \sum_{f \in F_v} \beta_{fv}. \tag{10}$$

On the other hand, let us denote the degree of each bounded face f by d_f. Then the angle sum of P_f is $(d_f - 2)\pi$. Summing this over all bounded faces gives

$$B = \sum_f (d_f - 2)\pi = \big[(2|E| - d_o) - 2(|F| - 1)\big]\pi = (|E| - |F| + 2 - d_o)2\pi + (d_o - 2)\pi. \tag{11}$$

Comparing the right-hand sides of (10) and (11), Euler's Formula gives the claim (9).

Thus, the sum of angles at the outer vertices has just the right value for a d_o-gon. Hence, the image P_o of the boundary edges is a convex polygon and therefore nonintersecting.

(D) We finally prove that the glued polygons P_f tile the interior of P_o without holes or overlap. Since we will refer to this argument later, we formulate it as a separate lemma:

▶ **Lemma 6.** *Let H be a 2-connected plane graph (possibly drawn with curved edges). Let H' be a straight-line drawing of H in the plane (possibly with crossings), with the following properties:*

(a) *For each bounded or unbounded face f of H, the edges of the face cycle in H' form a simple polygon P_f.*

(b) *For each inner edge e with incident faces f and f', the interior regions of P_f and $P_{f'}$ lie on different sides of e.*

Then H' contains no crossings.

In our case, the assumptions of this lemma are fulfilled: For the bounded faces, Property (a) holds by assumption, and for the unbounded face, it has just been established in Step (C). Property (b) has been established in Step (B). Thus, our second proof of Lemma 5 is complete once we prove Lemma 6:

Proof of Lemma 6. Let P_o denote the outer boundary, which is a simple polygon by assumption (a). We prove that the polygons P_f tile the interior of P_o without holes or overlap, using a covering number argument. Consider a point p on the plane which does not lie on an edge of one of the polygons. We can move this point to infinity along a straight ray which avoids all polygon vertices. We keep track of the number $X(p)$ of polygons in which p is contained. Whenever we cross an edge e of some polygon, we leave one polygon and enter another polygon, keeping $X(p)$ constant, unless e is an edge of P_o. In the last case, $X(p)$ changes by ± 1 in the correct way. This argument remains valid if we cross several edges simultanously (but we are about to show that this situation never occurs). Since $X(p) = 0$ when p is far away outside all polygons, it follows that all points p, except those on polygon boundaries, have $X(p) = 1$ if they lie inside P_o, and $X(p) = 0$ if they lie outside P_o. Consequently, the union of the polygons P_f is the polygon bounded by P_o, and the polygons cover it without overlap. ◀

◀

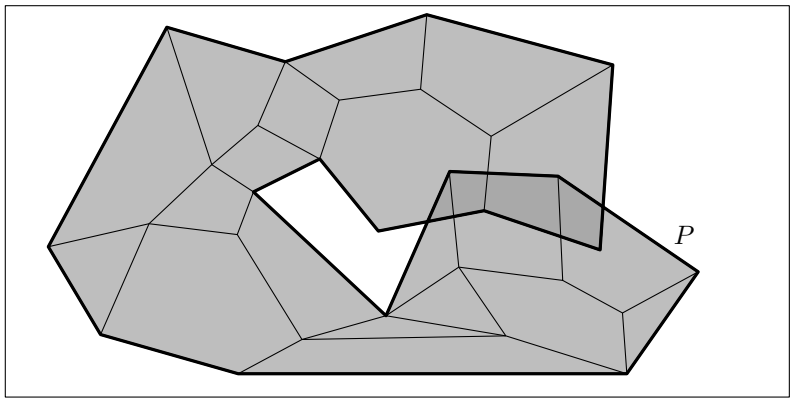


Figure 11 After placing a few faces, the outer cycle P might cross itself.

Our second proof of Lemma 5 generalizes to shapes P_f with curved edges: The matching condition (iii) of the lemma must then be strengthened in an appropriate way. The angle conditions corresponding to (i) and (ii) are not so straightforward to formulate, depending on the generality of the allowed boundaries, and an additional constraint is required to guarantee an overall convex shape (or at least a shape without self-overlap).

4.1 Comparison with Other Proofs

4.1.1 Lemma 5

We are aware of only one other proof of a statement like Lemma 5 in the literature: Brightwell and Scheinerman [6] (who did not formulate it as a separate lemma) gave a proof that is similar in spirit to our second proof. They successively place the polygons in some appropriate order, such that the boundary P of the placed polygons is always a simple cycle in the graph. In this way, what done in two separate Steps (A) and (B) in our proof, the placement of the faces and ensuring the consistency of the vertex positions, is achieved together. Step (C) is not necessary in their case, because the outer face is a triangle, and therefore it is automatically non-intersecting. The same statement actually applies to the application of Lemma 5 in the proof of our main theorem in Section 3.6, because the outer face is a rectangle in this case.

Step (D) is omitted in [6]. However, some argument like Lemma 6 is necessary, as illustrated by a hypothetical situation in Figure 11: The shaded faces have already been drawn, in a locally consistent way. While the outer boundary P forms a simple cycle in the graph, it self-intersects in the plane. It is conceivable that such a boundary can be completed with the remaining polygons to a locally consistent where the outer boundary becomes, say, the rectangular outline. It requires a proof that this cannot occur.

The existence of an appropriate face order for the face placement is assumed without justification in [6]. It is not hard to show that such an order can be chosen greedily: A proper subset of bounded faces enclosed by a simple cycle P can always be extended by an additional face f, so that $f \cup P$ is a connected curve, and the boundary remains a simple cycle.

(Alternatively, one can choose an edge st on the outer face and use a "bipolar orientation" (or an "s-t-numbering"), which is known to exist for any 2-connected graph. This results in an acyclic orientation of the dual graph, and any linear extension of this acyclic orientation is a suitable face order. We are grateful to Therese Biedl (private communication) for this observation.)

4.1.2 Lemma 6

Instead of Condition (b) of Lemma 6, we can stipulate that all faces are oriented consistently:

(b′) *Every bounded face cycle of H is oriented in the same way in H and H'.*

Up to reflecting the drawing and reversing the orientation of *every* face, this is equivalent to Condition (b): Condition (b′) clearly implies Condition (b). On the other hand, Condition (b) implies that adjacent bounded faces must be oriented consistently. (When the interior of the face is on the left, they must both be ordered clockwise or both counterclockwise.) Since the graph is 2-connected, the dual graph of the bounded faces is connected, and hence all bounded faces have to be ordered consistently.

A slightly different condition has been used by Devillers, Liotta, Preparata and Tamassia [10, Lemma 16]. Their lemma states that the following condition, in conjunction with Property (a), is sufficient to guarantee a non-crossing drawing:

(b″) *The cyclic order of the edges around every vertex is the same in H and H'.*

In contrast to Lemma 6, where the outer face of the initial drawing H is fixed and has to remain unchanged in H', this variation gives up the a-priori distinction between inner faces and the outer face. From the cyclic order in (b″), one can infer the face structure by walking around each face boundary, keeping the area of the face always to the left. The area of the face might turn out to be the inner (bounded) or the outer (unbounded) region bounded by the face cycle, depending on the orientation (counterclockwise or clockwise).

Our proof of Lemma 6 can be adapted to this situation: The regions P_f denote the (bounded or unbounded) face areas, and the goal is to show that these regions tile the whole plane, i. e., $X(p) = 1$ everywhere. To prove this, one has to establish that there is exactly one unbounded face. This can be shown by an account of the angle sums, like in Step (C) of our second proof of Lemma 5.

Lemma 16 of Devillers et al. [10] is stated for connected graphs and not just 2-connected graphs. In this case, face cycles are no longer simple polygons. The proof in [10] is sketchy, as acknowledged by one of the authors (private communication), and we could not fill all gaps. It is fortunate that Lemma 6 offers an alternate approach.

Di Battista and Vismara [3, Lemma 4.5] have previously proved another variation of the lemma where all interior faces are triangles. In this special case, condition (a) becomes trivial for the interior faces. Instead of condition (a), the only requirement in addition to (b″) is that and the boundary of the outer face turns only in one direction (cf. condition (ii) of Lemma 5 and the discussion in Step (C) of our second proof of Lemma 5). Their proof is by induction on the number of interior vertices, and the main argument proceeds by retriangulating the hole that is left after removing an interior vertex, like our first proof of Lemma 5.

References

1 Nieke Aerts and Stefan Felsner. Straight Line Triangle Representations. *Discr. and Comput. Geom.*, 57:257–280, 2017. doi:10.1007/s00454-016-9850-y.

2 E. M. Andreev. Convex polyhedra in Lobačevskiĭ spaces. *Mat. Sb. (N.S.)*, 81 (123):445–478, 1970. English: Math. USSR, Sb. 10, 413–440 (1971).

3 Giuseppe Di Battista and Luca Vismara. Angles of planar triangular graphs. *SIAM J. Discrete Math.*, 9(3):349–359, August 1996.

4 Marshall W. Bern and David Eppstein. Quadrilateral Meshing by Circle Packing. *Int. J. Comput. Geometry Appl.*, 10:347–360, 2000.

5 Alexander I. Bobenko and Boris A. Springborn. Variational principles for circle patterns and Koebe's theorem. *Trans. Amer. Math. Soc.*, 356:659–689, 2004.

6 G. R. Brightwell and E. R. Scheinerman. Representations of Planar Graphs. *SIAM J. Discr. Math.*, 6(2):214–229, 1993.

7 Yves Colin de Verdière. Empilements de cercles: convergence d'une méthode de point fixe. *Forum Math.*, 1:395–402, 1989.

8 Yves Colin de Verdière. Un principe variationnel pour les empilements de cercles. *Invent. Math.*, 104:655–669, 1991.

9 Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. On Triangle Contact Graphs. *Comb., Probab. and Comput.*, 3(02):233–246, 1994.

10 Olivier Devillers, Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Checking the convexity of polytopes and the planarity of subdivisions. *Comput. Geom.*, 11(3-4):187–208, 1998. doi:10.1016/S0925-7721(98)00039-X.

11 David Eppstein. Diamond-Kite Meshes: Adaptive Quadrilateral Meshing and Orthogonal Circle Packing. In *Proc. Mesh. Roundtable*, pages 261–277. Springer, 2012.

12 David Eppstein. A Möbius-invariant power diagram and its applications to soap bubbles and planar Lombardi drawing. *Discrete Comput. Geom.*, 52:515–550, 2014.

13 Stefan Felsner. Rectangle and Square Representations of Planar Graphs. In J. Pach, editor, *Thirty Essays in Geometric Graph Theory*, pages 213–248. Springer, 2013. doi:10.1007/978-1-4614-0110-0_12.

14 Stefan Felsner, Alexander Igamberdiev, Philipp Kindermann, Boris Klemz, Tamara Mchedlidze, and Manfred Scheucher. Strongly Monotone Drawings of Planar Graphs. In *Proc. 32nd Intern. Symp. Comput. Geom. (SoCG 2016)*, volume 51 of *LIPIcs*, pages 37:1–15, 2016.

15 Stefan Felsner, Hendrik Schrezenmaier, and Raphael Steiner. Equiangular polygon contact representations, 2017. page.math.tu-berlin.de/~felsner/Paper/kgons.pdf.

16 Stefan Felsner, Hendrik Schrezenmaier, and Raphael Steiner. Pentagon Contact Representations. *Electronic J. Combin.*, 25(3):article #P3.39, 38 pp., 2018.

17 Daniel Gonçalves, Benjamin Lévêque, and Alexandre Pinlou. Triangle Contact Representations and Duality. *Discr. and Comput. Geom.*, 48(1):239–254, 2012.

18 S. Har-Peled. A Simple Proof of the Existence of a Planar Separator. arXiv:1105.0103, 2011.

19 Nora Hartsfield and Gerhard Ringel. *Pearls in Graph Theory: A Comprehensive Introduction*. Academic Press, 1990.

20 Brad Jackson and Gerhard Ringel. Colorings of circles. *Amer. Math. Monthly*, 91:42–49, 1984.

21 Paul Koebe. Kontaktprobleme der konformen Abbildung. *Ber. Verh. Sächs. Akad. Leipzig, Math.-Phys. Klasse*, 88:141–164, 1936.

22 László Lovász. Geometric Representations of Graphs. web.cs.elte.hu/~lovasz/geomrep.pdf, December 2009. Draft version.

23 Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997.

24 B. Mohar. Circle packings of maps in polynomial time. *Europ. J. Comb.*, 18:785–805, 1997.

25 Bojan Mohar. Circle packings of maps—the Euclidean case. *Rend. Sem. Mat. Fis. Milano*, 67:191–206, 2000.

26 David Orden, Günter Rote, Francisco Santos, Brigitte Servatius, Herman Servatius, and Walter Whiteley. Non-crossing frameworks with non-crossing reciprocals. *Discrete and Computational Geometry*, 32:567–600, 2004. doi:10.1007/s00454-004-1139-x.

27 János Pach and Pankaj K. Agarwal. *Combinatorial Geometry*. John Wiley & Sons, 1995.

28 Gerhard Ringel. 250 Jahre Graphentheorie. In *Graphs in research and teaching*, pages 136–152. Franzbecker, 1985.

29 Burt Rodin and Dennis Sullivan. The convergence of circle packings to the Riemann mapping. *J. Differential Geom.*, 26:349–360, 1987.

30 Günter Rote, Francisco Santos, and Ileana Streinu. Pseudo-triangulations — a survey. In Jacob E. Goodman, János Pach, and Richard Pollack, editors, *Surveys on Discrete and Computational Geometry—Twenty Years Later*, volume 453 of *Contemporary Mathematics*, pages 343–410. American Mathematical Society, 2008. `arXiv:math/0612672`.

31 H. Sachs. Coin Graphs, Polyhedra, and Conformal Mapping. *Discr. Math.*, 134:133–138, 1994.

32 Oded Schramm. How to Cage an Egg. *Invent. Math.*, 107:534–560, 1992.

33 Oded Schramm. Square tilings with prescribed combinatorics. *Isr. J. Math.*, 84:97–118, 1993.

34 Oded Schramm. Combinatorically Prescribed Packings and Applications to Conformal and Quasiconformal Maps. arXiv:`0709.0710`, 2007. Modified version of PhD thesis from 1990.

35 Kenneth Stephenson. Circle packing: a mathematical tale. *Notices Amer. Math. Soc.*, 50:1376–1388, 2003.

36 Kenneth Stephenson. *Introduction to Circle Packing: The Theory of Discrete Analytic Functions*. Cambridge Univ. Press, 2005.

37 William Thurston. *Geometry and Topology of Three-Manifolds*. Princeton Univ., 1980. Lecture Notes. Electronic edition: `library.msri.org/books/gt3m/`.

38 W. T. Tutte. How to draw a graph. *Proc. London Math. Soc. (Ser. 3)*, 13:743–767, 1963.

39 G. Wegner. Problems in geometric convexity: Problem 86. In J. Tölke and J. M. Wills, editors, *Contributions to Geometry. Proceedings of the Geometry Symposium in Siegen 1978*, page 274. Birkhäuser, 1979.

A Proof of Lemma 4

▶ **Lemma 4.** *A simple bipartite plane graph with $|S| \geq 4$ nodes has at most $|E| \leq 2|S| - 4$ edges, with equality if and only if the graph is connected and every face is a quadrilateral with four distinct vertices.*

Proof. If the graph is not connected, we add a minimal set of edges to make it connected while keeping it plane and bipartite, resulting in a larger edge set E'.

Since the graph is bipartite, every face cycle has even length. Moreover, every face cycle contains at least 4 edges (possibly visiting both sides of a single edge). To see this, note that the only possible exception, a "digonal" face cycle, would have to be the two sides of a single isolated edge, or two parallel edges. Since $|S| \geq 3$ and the graph is connected and has no multiple edges this cannot happen.

Denoting the set of faces by F, standard double-counting gives the relation $4|F| \leq 2|E'|$, because every edge has 2 sides, and every face cycle goes through at least 4 sides of edges. Euler's formula gives then $|E'|+2 = |S|+|F| \leq |S|+|E'|/2$ and therefore $|E'| \leq 2|S|-4$, with equality if and only all face cycles have length 4. Together, in the chain $|E| \leq |E'| \leq 2|S|-4$, equality cannot hold if the original graph with edge set E was disconnected ($|E| < |E'|$).

We still have to exclude face cycles of length 4 that are not quadrilaterals (i. e., with 4 distinct vertices). Such a cycle could only be the face surrounding a path with two edges. This is excluded because $|S| \geq 4$ and the graph is connected. ◀

Asymmetric Convex Intersection Testing

Luis Barba
Department of Computer Science, ETH Zürich, 8092 Zürich, Switzerland
luis.barba@inf.ethz.ch

Wolfgang Mulzer[1]
Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
mulzer@inf.fu-berlin.de
https://orcid.org/0000-0002-1948-5840

Abstract

We consider *asymmetric convex intersection testing* (ACIT).

Let $P \subset \mathbb{R}^d$ be a set of n points and $\mathcal{H}$ a set of n halfspaces in d dimensions. We denote by $\textsc{ch}(P)$ the polytope obtained by taking the convex hull of P, and by $\textsc{fh}(\mathcal{H})$ the polytope obtained by taking the intersection of the halfspaces in $\mathcal{H}$. Our goal is to decide whether the intersection of $\mathcal{H}$ and the convex hull of P are disjoint. Even though ACIT is a natural variant of classic LP-type problems that have been studied at length in the literature, and despite its applications in the analysis of high-dimensional data sets, it appears that the problem has not been studied before.

We discuss how known approaches can be used to attack the ACIT problem, and we provide a very simple strategy that leads to a deterministic algorithm, linear on n and m, whose running time depends reasonably on the dimension d.

2012 ACM Subject Classification Mathematics of computing → Combinatorics

Keywords and phrases polytope intersection, LP-type problem, randomized algorithm

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.9

Related Version Also available on the arXiv as `https://arxiv.org/abs/1808.06460`.

Acknowledgements This work was initiated at the *Sixth Annual Workshop on Geometry and Graphs*, that took place March 11–16, 2018, at the Bellairs Research Institute. We would like to thank the organizers and all participants of the workshop for stimulating discussions and for creating a conducive research environment. We would also like Timothy M. Chan for answering our questions about LP-type problems and for pointing us to several helpful references.

1 Introduction

Let $d \in \mathbb{N}$ be a fixed constant. Convex polytopes in dimension d can be implicitly represented in two ways, either by its set of vertices, or by the set of halfspaces whose intersection defines the polytope. A polytope represented by its vertices is usually called a *V-polytope*, while a polytope represented by a set of halfspaces is known as an *H-polytope*. Note that the actual complexity of the polytopes can be much larger than the size of their representations [20, Theorem 5.4.5]. In this paper, we study the problem of testing the intersection of convex polytopes with different implicit representations. When both polytopes have the same representation, testing for their intersection reduces to linear programming. However, when

[1] Partially supported by DFG grant MU/3501/3 and ERC STG 757609.

there is a mismatch in the representation, the problem changes in nature and becomes more challenging.

To formalize our problem, let $P \subset \mathbb{R}^d$ be a set of n points in $\mathbb{R}^d$, and let $\mathcal{H}$ be a set of n halfspaces in $\mathbb{R}^d$.[2] Just as P implicitly defines the polytope $\text{CH}(P)$ obtained by taking the convex hull of P, the set $\mathcal{H}$ implicitly defines the polytope $\text{FH}(\mathcal{H})$ obtained by taking the intersection of the halfspaces in $\mathcal{H}$. In the *asymmetric convex intersection problem* (ACIT), our goal is to decide whether the intersection of $\mathcal{H}$ and the convex hull of P are disjoint.

We may assume that $\text{FH}(\mathcal{H})$ is nonempty. Otherwise, ACIT becomes trivial. If $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ intersect, we would like to find a *witness point* in both $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$; if not, we would like to determine the closest pair between $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ and a separating hyperplane.

Even though ACIT seems to be a natural problem that fits well into the existing work on algorithmic aspects of high-dimensional polytopes [1], we are not aware of any prior work on it. While intersection detection of convex polytopes has been a central topic in computational geometry [14, 21, 13, 6, 5, 9], when we deal with an intersection test between a V-polytope and an H-polytope, the problem seems to remain unstudied. Even the seemingly easy case of this problem in dimension $d = 2$ has no trivial solution running in linear time.

The lack of a solution for ACIT may be even more surprising considering that ACIT can be used in the analysis of high-dimensional data: given a high-dimensional data set, represented as a point cloud P, it is natural to represent the *interpolation* of the data as the convex hull $\text{CH}(P)$. Then, we would like to know whether the interpolated data set contains an item that satisfies certain *properties*. These properties are usually represented as linear constraints that must be satisfied, i.e., the data point must belong to the intersection of a set of halfspaces. Then, a witness point corresponds to an interpolated data point with the desired properties, and a separating hyperplane may indicate which properties cannot be fulfilled by the data at hand.

Even though ACIT has not been addressed before, several approaches for related problems[3] may be used to attack the problem. The range of techniques goes from simple brute-force, over classic linear programming [10], the theory of LP-type problems [8, 24] (also in implicit form [2]), to parametric search [19]. In Section 2, we will examine these in more detail and discuss their merits and drawbacks. Briefly, several of these approaches can be applied to ACIT. However, as we will see, it seems hard to get an algorithm that is genuinely simple and at the same time achieves linear (or almost linear) running time in the number of points and halfspaces, with a reasonable dependency on the dimension d.

Thus, in Section 4, we present a simple recursive primal-dual pruning strategy that leads to a deterministic linear time algorithm with a dependence on d that is comparable to the best bounds for linear programming. Even though the algorithm itself is simple and can be presented in a few lines, the analysis requires us to take a close look at the polarity transformation and how it interacts with two disjoint polytopes (Section 3). Its analysis is also non-trivial and its correctness spans over the entire Section 4.3. We believe in the development of simple and efficient methods. The analysis can be complicated, but the algorithm must remain simple. The simpler the algorithm, the more likely it is to be eventually implemented.

---

[2] We will assume that both P and $\mathcal{H}$ are in *general position* (the exact meaning of this will be made clear later)

[3] In particular, checking for the intersection of the convex hulls of to d-dimensional point sets

2 How to solve ACIT with existing tools

The first thing that might come to mind to solve ACIT is to cast it as a linear program. This is indeed possible, however the resulting linear program consists of n variables and $\Theta(n)$ constraints. We want to find a point x subject to being inside all halfspaces in $\mathcal{H}$, and being a convex combination of all points in P. That is, we want $x = \sum_{p\in P} \alpha_p\, p$, where $\sum_{p\in P} \alpha_p = 1$, and $\alpha_p \geq 0$, for all $p \in P$. Moreover, we want that $x \in H$, for all $H \in \mathcal{H}$, which can be expressed as n linear inequalities by looking at the scalar product of x and the normal vectors of the bounding hyperplanes of the halfspaces. Because the best combinatorial algorithms for linear programming provide poor running times when both the number of variables and constraints are large, this approach is far from efficient unless n is really small.[4]

Another trivial way to solve ACIT, the *brute force* algorithm, is to compute all facets of $\textsc{ch}(P)$. That is, we can compute $\textsc{ch}(P)$ explicitly to obtain a set $\mathcal{H}_P$ of the $O(n^{\lfloor d/2\rfloor})$ halfspaces with $\textsc{ch}(P) = \textsc{fh}(\mathcal{H}_P)$ [12, 7]. With this representation, we can test if $\textsc{fh}(\mathcal{H}_P)$ and $\textsc{fh}(\mathcal{H})$ intersect using a general linear program with d variables, or compute the distance between $\textsc{fh}(\mathcal{H}_P)$ and $\textsc{fh}(\mathcal{H})$ using either an LP-type algorithm (see below), or algorithms for convex quadratic programming [16, 17]. The running time is again quite bad for larger values of n, since the size of $\mathcal{H}_P$ might be as high as $\Theta(n^{\lfloor d/2\rfloor})$ [20, Theorem 5.4.5].

A more clever approach is to use the LP-type framework directly, as described below.

The LP-type Framework. The classic *LP-type framework* that was introduced by Sharir and Welzl [24] in order to extend the notion of low-dimensional linear programming to a wider range of problems. An LP-type problem $(\mathcal{C}, w)$ consists of a set $\mathcal{C}$ of k *constraints* and a *weight function* $w : 2^{\mathcal{C}} \to \mathbb{R}$ that assigns a real-valued weight $w(C)$ to each set $C \subseteq \mathcal{C}$ of constraints.[5] The weight function must satisfy the following three axioms:

- **Monotonicity**: For any set $C \subseteq \mathcal{C}$ of constraints and any $c \in \mathcal{C}$, we have $w\big(C \cup \{c\}\big) \leq w(C)$.
- **Existence of a Basis**: There is a constant $\tilde{d} \in \mathbb{N}$ such that for any $C \subseteq \mathcal{C}$, there is a subset $B \subseteq C$ with $|B| \leq \tilde{d}$ and $w(B) = w(C)$.
- **Locality**: For any $B \subseteq C \subseteq \mathcal{C}$ with $w(B) = w(C)$ and for any $c \in C$, we have that if $w\big(C \cup \{c\}\big) < w(C)$, then also $w\big(B \cup \{c\}\big) < w(B)$.

For $C \subseteq \mathcal{C}$, an inclusion-minimal subset $B \subseteq C$ with $w(B) = w(C)$ is called a *basis* for C. Solving an LP-type problem $(\mathcal{C}, w)$ amounts to computing a basis for $\mathcal{C}$. Many algorithms have been developed for this extension of linear programming, provided that base cases with a constant number of constraints can be solved in $O(1)$ time. Seidel proposed a simple randomized algorithm with expected $O(\tilde{d}!k)$ running time [23]. From there, several algorithms have been introduced improving the dependency on $\tilde{d}$ in the running time [3, 11, 23, 24]. The best known randomized algorithm solves LP-type problems in $O(\tilde{d}^2 k + 2^{O(\sqrt{\tilde{d}\log\tilde{d}})})$ time, while the best deterministic algorithms have still a running time of the form $O(\tilde{d}^{O(\tilde{d})}k)$. We would like to obtain an algorithm with a similar running time for ACIT.

[4] In fact, in the traditional computational model of computational geometry, the Real RAM [22], we cannot solve general linear programs in polynomial time, since the best known algorithms (e.g., ellipsoid, interior point methods) are only *weakly* polynomial with a running time that depends on the bit complexity of the input.

[5] Actually, we can allow weights from an arbitrary totally ordered set, but for our purposes, real weights will suffice.

ACIT as an LP-type problem. To use these existing machinery, one can try to cast ACIT as an LP-type problem. To this end, we fix $\mathcal{H}$, and define an LP-type problem (P, w) as follows. The *constraints* are modeled by the points in P. The weight function $w : 2^P \to \mathbb{R}$ is defined as $w(Q) = d\big(\textsc{ch}(Q), \textsc{fh}(\mathcal{H})\big)$, for any $Q \subseteq P$, where $d(\cdot, \cdot)$ is the smallest Euclidean distance between any pair of points from the two polytopes. It is a pleasant exercise to show that this indeed defines an LP-type problem of combinatorial dimension d.

Thus, the elegant methods to solve LP-type problems mentioned above become applicable. Unfortunately, this does not give an efficient algorithm. This is because the set $\mathcal{H}$ remains fixed throughout, making unfeasible to solve the base cases consisting of $O(1)$ constraints of P in constant time.

A randomized algorithm for ACIT. As an extension of the LP-type framework, Chan [2] introduced a new technique that allows us to deal with certain LP-type problems where the constraints are too numerous to write explicitly, and are instead specified "implicitly". More precisely, as mentioned above, ACIT can be seen as a linear program, with m constraints coming from $\mathcal{H}$, and $O(n^{\lfloor d/2 \rfloor})$ constraints coming from all the facets of $\textsc{ch}(P)$. The latter are implicitly defined by P using only n points. Thus an algorithm capable of solving implicitly defined linear programs would provide a solution for ACIT. The technique developed by Chan achieves this by using two main ingredients: a decision algorithm, and a partition of the problem into subproblems of smaller size whose recursive solution can be combined to produce the global solution of the problem. Using the power of randomization and geometric cuttings, this technique leads to a complicated algorithm to solve this implicit linear program, and hence ACIT, in expected $O(d^{O(d)}n)$ time [4]. Besides the complexity of this algorithm, the constant hidden by the big O notation resulting from using this technique seems prohibitive [18].

In hope of obtaining a deterministic algorithm for ACIT, one can turn to multidimensional parametric search [19] to try de-randomizing the above algorithm. However, even if all the requirements of this technique can be sorted out, it would lead to a highly complicated algorithm and polylogarithmic overhead.

In the following sections, we present the first deterministic solution for ACIT using a simple algorithm that overcomes the difficulties mentioned above. We achieve this solution by diving into the intrinsic duality of the problem provided by the polar transformation, while exploiting the LP-type-like structure of our problem. The resulting algorithm is quite simple, and a randomized version of it could be written with a few lines of code, provided that one has some LP solver at hand.

3 Geometric Preliminaries

Let $\mathbf{o}$ denote the *origin* of $\mathbb{R}^d$. A *hyperplane* h is a $(d-1)$-dimensional affine space in $\mathbb{R}^d$ of the form

$$h = \{x \in \mathbb{R}^d \mid \langle z, x \rangle = 1\},$$

for some $z \in \mathbb{R}^d \setminus \{\mathbf{o}\}$, where $\langle \cdot, \cdot \rangle$ represents the scalar product in $\mathbb{R}^d$. We exclude hyperplanes that pass through the origin. A (closed) *halfspace* is the closure of the point set on either side of a given hyperplane, i.e., a halfspace contains the hyperplane defining its boundary.

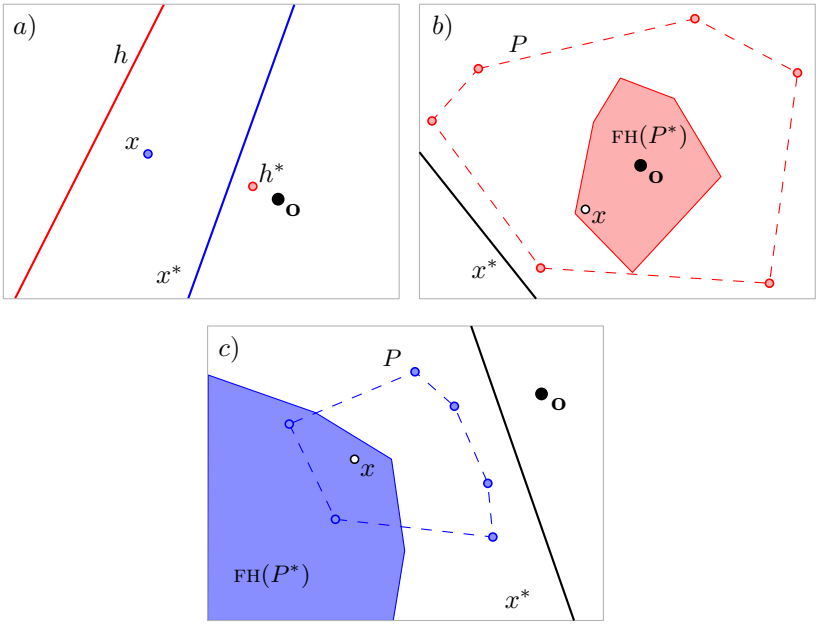


Figure 1 (a) The situation described in Lemma 3.1. (b) A valid set P of points that is embracing and its polar P^* that is also embracing. (c) A set P that is avoiding and its polar P^* that is also avoiding.

3.1 The Polar Transformation

Given a point $p \in \mathbb{R}^d$, we define the *polar* of p to be the hyperplane

$$p^* = \{x \in \mathbb{R}^d \mid \langle p, x\rangle = 1\}.$$

Given a hyperplane h in $\mathbb{R}^d$, we define its *polar* $h^* \in \mathbb{R}^d$ as the point with

$$h = \{x \in \mathbb{R}^d \mid \langle x, h^*\rangle = 1\}.$$

Let $\rho_{\mathbf{o}}(p) = \{x \in \mathbb{R}^d \mid \langle p, x\rangle \leq 1\}$ and $\rho_{\infty}(p) = \{x \in \mathbb{R}^d \mid \langle p, x\rangle \geq 1\}$ be the two halfspaces supported by p^* such that $\mathbf{o} \in \rho_{\mathbf{o}}(p)$ and $\mathbf{o} \notin \rho_{\infty}(p)$. Similarly, $h_{\mathbf{o}}$ and h_{∞} denote the halfspaces supported by h such that $\mathbf{o} \in h_{\mathbf{o}}$ and $\mathbf{o} \notin h_{\infty}$.

Note that the polar of a point $p \in \mathbb{R}^d$ is a hyperplane whose polar is equal to p, i.e., the polar operation is involutory (for more details, see Section 2.3 in Ziegler's book [25]). The following result is illustrated in Figure 1(a), for $d = 2$.

▶ **Lemma 3.1** (Lemma 2.1 of [1]). *Let p and h be a point and a hyperplane in $\mathbb{R}^d$, respectively. Then, $p \in h_{\mathbf{o}}$ if and only if $h^* \in \rho_{\mathbf{o}}(p)$. Also, $p \in h_{\infty}$ if and only if $h^* \in \rho_{\infty}(p)$. Finally, $p \in h$ if and only if $h^* \in p^*$.*

Let P be a set of points in $\mathbb{R}^d$. We say that P is *embracing* if $\mathbf{o}$ lies in the interior of $\text{CH}(P)$. We say that P is *avoiding* if $\mathbf{o}$ lies in the complement of $\text{CH}(P)$. Note that we do not consider point sets whose convex hull has $\mathbf{o}$ on its boundary. We say that P is *valid* if it is either embracing or avoiding.

Let $\mathcal{H}$ be a set of halfspaces in $\mathbb{R}^d$ such that $\text{FH}(\mathcal{H}) \neq \emptyset$, and the boundary of no halfspace in $\mathcal{H}$ contains $\mathbf{o}$. We say that $\mathcal{H}$ is *embracing* if $\mathbf{o} \in H$ for all $H \in \mathcal{H}$ (i.e., $\mathbf{o} \in \text{FH}(\mathcal{H})$). We say that $\mathcal{H}$ is *avoiding* if none of its halfspaces contains $\mathbf{o}$, i.e., $\mathbf{o} \notin \bigcup_{H\in\mathcal{H}} H$. We say that $\mathcal{H}$ is *valid* if it is either embracing or avoiding.

We now describe how to polarize convex polytopes defined as convex hulls of valid sets of points or as intersections of valid sets of halfspaces. Let $\mathcal{H}$ be a valid set of halfspaces in $\mathbb{R}^d$. To *polarize* $\mathcal{H}$, consider the set of hyperplanes bounding the halfspaces in $\mathcal{H}$, and let $\mathcal{H}^*$ be the set consisting of all the points being the polars of these hyperplanes.

▶ **Lemma 3.2.** *Let $\mathcal{H}$ be a valid set of halfspaces in $\mathbb{R}^d$. Then, $\mathcal{H}^*$ is embracing if and only if $\mathcal{H}$ is embracing.*

Proof. Recall that $\mathcal{H}$ is embracing if and only if $\textsc{fh}(\mathcal{H})$ is bounded and contains $\mathbf{o}$.

$\Rightarrow$). Assume that $\mathcal{H}^*$ is embracing. Thus, $\mathbf{o} \in \textsc{ch}(\mathcal{H}^*)$. In this case, there is a subset Q of $d+1$ points of $\mathcal{H}^*$ whose convex hull contains $\mathbf{o}$, by Carathéodory's theorem [20, Theorem 1.2.3]. Consider all halfspaces of $\mathcal{H}$ whose boundary polarizes to a point in Q. If none of these halfspaces contains the origin, then their intersection has to be empty. This is not allowed by the validity of $\mathcal{H}$. Thus, as $\mathcal{H}$ is valid, and as $\mathcal{H}$ cannot avoid the origin, we conclude that $\mathcal{H}$ is embracing.

$\Leftarrow$). For the other direction, assume that $\mathbf{o} \notin \textsc{ch}(\mathcal{H}^*)$. We want to prove that $\mathcal{H}$ is not embracing. For this, let h be a hyperplane that separates $\mathbf{o}$ from $\textsc{ch}(\mathcal{H}^*)$. That is, $\mathcal{H}^* \subset h_\infty$. Lemma 3.1 implies that the segment $\mathbf{o}h^*$ intersects the boundary of each plane in $\mathcal{H}$. Therefore, since the ray shooting from $\mathbf{o}$ in the direction of the vector $-h^*$ intersects no plane bounding a halfspace in $\mathcal{H}$, the polytope $\textsc{fh}(\mathcal{H})$ either does not contain the origin or is not bounded. Consequently, $\mathcal{H}$ is not embracing. ◀

Let P be a valid set of points in $\mathbb{R}^d$. To *polarize* P, let $\Pi(P)$ be the set of hyperplanes polar to the points of P. We have two natural ways of polarizing P, depending on whether $\mathbf{o}$ lies in the interior of $\textsc{ch}(P)$, or in its complement (recall that $\mathbf{o}$ cannot lie on the boundary of $\textsc{ch}(P)$). If $\mathbf{o} \in \textsc{ch}(P)$, then

$$P^* = \left\{ h_\circ \mid h \in \Pi(P) \right\}$$

is the *polarization* of P. Otherwise, if $\mathbf{o} \notin \textsc{ch}(P)$, then

$$P^* = \left\{ h_\infty \mid h \in \Pi(P) \right\}.$$

▶ **Lemma 3.3.** *Let P be a valid set of points in $\mathbb{R}^d$. Then P^* is valid, i.e., $\textsc{fh}(P^*) \neq \emptyset$ and P^* is either embracing or avoiding.*

Proof. If $\mathbf{o} \notin \textsc{ch}(P)$, then there is a hyperplane h such that $P \subset h_\infty$. Thus, h^* belongs to p^* for every $p \in P$, i.e., $h^* \in \textsc{fh}(P^*)$. Thus, $\textsc{fh}(P^*)$ is nonempty, and none of its halfspaces contains the origin by definition. That is, P^* is avoiding. If $\mathbf{o} \in \textsc{ch}(P)$, then $\mathbf{o} \in \textsc{fh}(P^*)$ by definition. Thus, to show that P^* is embracing, it remains only to show that it is bounded. To this end, assume for a contradiction that $\textsc{fh}(P^*)$ is unbounded. Then, we can take a point $x \in \textsc{fh}(P^*)$ at arbitrarily large distance from $\mathbf{o}$. Thus, x^* is a plane arbitrarily close to $\mathbf{o}$ such that $P \subset x^*$. Therefore, all points of P must lie on a single halfspace that contains $\mathbf{o}$ on its boundary. Because P is valid, we know that $\mathbf{o}$ cannot lie on the boundary of $\textsc{ch}(P)$ and hence, $\mathbf{o} \notin \textsc{ch}(P)$—a contradiction with our assumption that $\mathbf{o} \in \textsc{ch}(P)$. Therefore, $\textsc{fh}(P^*)$ is bounded and hence P^* is embracing. ◀

▶ **Lemma 3.4.** *Let $P \subset \mathbb{R}^d$ be a valid finite point set in d dimensions, and let $\mathcal{H}$ be a valid finite set of halfspaces in d dimensions. Then the polar operator is involutory: $P = (P^*)^*$ and $\mathcal{H} = (\mathcal{H}^*)^*$.*

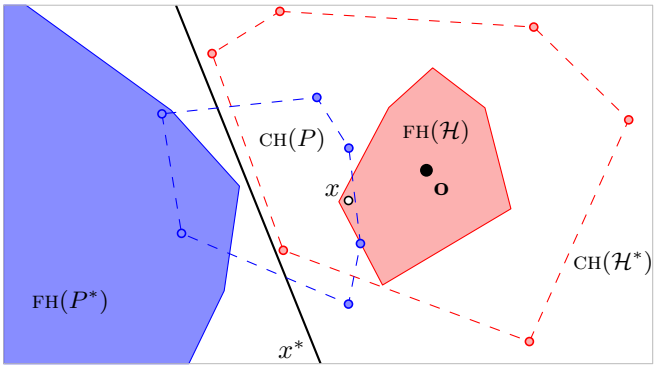


Figure 2 An example of Theorem 3.6 in dimension 2, where a point x lies in the intersection of $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ if and only if x^* separates $\text{CH}(\mathcal{H}^*)$ from $\text{FH}(P^*)$.

Proof. The equality $P = (P^*)^*$ follows directly from the definition, because the polar operator for points and hyperplanes is involutory. For equality $\mathcal{H} = (\mathcal{H}^*)^*$, we must check that the orientation of the halfspaces is preserved. First, if $\mathcal{H}$ is embracing, i..e, $\mathbf{o} \in \text{FH}(\mathcal{H})$, then every $H \in \mathcal{H}$ is of the form $H = h_{\mathbf{o}}$, for some $(d-1)$-dimensional hyperplane h. Moreover, Lemma 3.2 implies that $\mathbf{o} \in \text{CH}(\mathcal{H}^*)$. Thus, we have $\mathcal{H} = (\mathcal{H}^*)^*$ in this case. Similarly, if $\mathcal{H}$ is avoiding, i.e., $\mathbf{o} \notin \bigcup_{H \in \mathcal{H}} H$, then every $H \in \mathcal{H}$ is of the form $H = h_\infty$ for some $(d-1)$-dimensional hyperplane h, and by Lemma 3.2, we have $\mathbf{o} \notin \text{CH}(\mathcal{H}^*)$. Thus, we have again $\mathcal{H} = (\mathcal{H}^*)^*$. ◀

▶ **Corollary 3.5.** *Let P be a valid set of points in $\mathbb{R}^d$. Then, P^* is embracing if and only if P is embracing. Moreover, P^* is avoiding if and only if P is avoiding.*

Proof. Because P is valid, P^* is valid by Lemma 3.3. Therefore, Lemma 3.2 implies that P^* is embracing if and only if $(P^*)^*$ is embracing. Because $P = (P^*)^*$ by Lemma 3.4, we conclude that P^* is embracing if and only if P is embracing. Note that if a valid set P is not embracing, then it is avoiding, yielding the second part of the result. ◀

The following result is illustrated in Figure 2, for $d = 2$.

▶ **Theorem 3.6** (Consequence of Theorem 3.1 of [1])**.** *Let P be a finite set of points and let $\mathcal{H}$ be a valid finite set of halfspaces in $\mathbb{R}^d$ such that either* (1) *P is avoiding while $\mathcal{H}$ is embracing, or* (2) *P is embracing while $\mathcal{H}$ is avoiding. Then, a point x lies in the intersection of $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ if and only if the hyperplane x^* separates $\text{FH}(P^*)$ from $\text{CH}(\mathcal{H}^*)$. Also a hyperplane h separates $\text{CH}(P)$ from $\text{FH}(\mathcal{H})$ if and only if the point h^* lies in the intersection of $\text{FH}(P^*)$ and $\text{CH}(\mathcal{H}^*)$.*

Conditions (1) and (2) will be crucial in our algorithm. Note that by Corollary 3.5, we have that if P and $\mathcal{H}$ satisfy condition (1), then the point set $\mathcal{H}^*$ and the set P^* of halfspaces satisfy condition (2), and vice versa.

3.2 Conflict Sets, Epsilon-nets, and Closest Pairs

Let $P \subseteq \mathbb{R}^d$ be a finite point set in d dimensions, and let H be a halfspace in $\mathbb{R}^d$. We say that a point $p \in P$ *conflicts* with H if $p \in H$. The *conflict set* of P and H, denoted $V_H(P)$, consists of all points $p \in P$ that are in conflict with H, i.e., $V_H(P) = P \cap H$. Let $\varepsilon \in (0, 1)$

be a parameter. A set $N \subseteq P$ is called an *ε-net* for P if for every halfspace H in $\mathbb{R}^d$, we have

$$V_H(N) = \emptyset \Rightarrow \big|V_H(P)\big| < \varepsilon|P|. \tag{1}$$

By the classic *ε-net theorem* of Haussler and Welzl [15, Theorem 5.28], a random subset $N \subset P$ of size $\Theta\Big(\varepsilon^{-1}\log\big(\varepsilon^{-1}+\alpha^{-1}\big)\Big)$ is an ε-net for P with probability at least $1-\alpha$. For a deterministic algorithm running in linear time, we can compute such a net using the complicated algorithm of Chazelle and Matoušek [8, Chapter 4.3] or the much simpler algorithm introduced by Chan [3]. See the textbooks of Matoušek [20], Chazelle [8], or Har-Peled [15] for more details on ε-nets and their uses in computational geometry. The following observation shows the usefulness of conflict sets for our problem.

▶ **Lemma 3.7.** *Let $P \subseteq \mathbb{R}^d$ be a finite point set and $\mathcal{H}$ a finite set of halfspaces in d dimensions. Let $N \subseteq P$ such that $\textsc{fh}(\mathcal{H})$ and $\textsc{ch}(N)$ are disjoint, and let x, y be the closest pair between them, such that $x \in \textsc{fh}(\mathcal{H})$ and $y \in \textsc{ch}(N)$. Let H_y be the halfspace through y perpendicular to the segment xy, containing $\textsc{fh}(\mathcal{H})$. Then, we have $d\big(\textsc{fh}(\mathcal{H}), P\big) < d\big(\textsc{fh}(\mathcal{H}), N\big)$ if and only if $V_{H_y}(P) \neq \emptyset$.*

Proof. Since all points in $\mathbb{R}^d \setminus H_y$ have distance larger than $d\big(\textsc{fh}(\mathcal{H}), N\big)$ from $\textsc{fh}(\mathcal{H})$, the implication $V_{H_y}(P) = \emptyset \Rightarrow d\big(\textsc{fh}(\mathcal{H}), P\big) < d\big(\textsc{fh}(\mathcal{H}), N\big)$ is immediate.

Now assume that $V_{H_y}(P) \neq \emptyset$, say, $p \in V_{H_y}(P)$. Then, the line segment py is contained in $\textsc{ch}(P)$, and since $p \in V_{H_y}(P)$ and since p does not lie on the boundary of H_y be our general position assumption, it follows that the angle between the segments py and xy is strictly smaller than $\pi/2$. Hence, we have

$$d\big(\textsc{fh}(\mathcal{H}), P\big) \leq d(x, py) < \big(\textsc{fh}(\mathcal{H}), N\big),$$

as claimed. ◀

Similarly, let $\mathcal{H}$ be a finite set of halfspaces in d dimensions, and let $p \in \mathbb{R}^d$ be a point. The *conflict set* $V_p(\mathcal{H})$ of $\mathcal{H}$ and p consists of all halfspaces that do not contain p, i.e. $V_p(\mathcal{H}) = \{H \in \mathcal{H} \mid p \notin H\}$. We have the following polar version of Lemma 3.7:

▶ **Lemma 3.8.** *Let $P \subseteq \mathbb{R}^d$ be a finite point set and $\mathcal{H}$ a finite set of halfspaces in d dimensions. Let $\mathcal{H}' \subseteq \mathcal{H}$ such that $\textsc{fh}(\mathcal{H}')$ and $\textsc{ch}(P)$ are disjoint, and let x, y be the closest pair between them, such that $x \in \textsc{fh}(\mathcal{H}')$ and $y \in \textsc{ch}(P)$. Then, we have $d\big(\textsc{fh}(\mathcal{H}), P\big) > d\big(\textsc{fh}(\mathcal{H}'), P\big)$ if and only if $V_x(\mathcal{H}) \neq \emptyset$.*

Proof. First, if $V_x(\mathcal{H}) = \emptyset$, then $x \in \textsc{fh}(\mathcal{H})$, and since $\textsc{fh}(\mathcal{H}) \subseteq \textsc{fh}(\mathcal{H}')$, it follows that $d\big(\textsc{fh}(\mathcal{H}), P\big) = d\big(\textsc{fh}(\mathcal{H}'), P\big)$.

Second, suppose that $V_x(\mathcal{H}) \neq \emptyset$, say, $H \in V_x(\mathcal{H})$. Then, $x \notin H$, and since, by gneral position, x is the unique point in $\textsc{fh}(\mathcal{H}')$ with $d(x, \textsc{ch}(P)) = d(\textsc{fh}(\mathcal{H}'), \textsc{ch}(P))$, we have

$$d\big(\textsc{fh}(\mathcal{H}), P\big) \geq d\big(\textsc{fh}(\mathcal{H}' \cup \{H\}), P\big) > d\big(\textsc{fh}(\mathcal{H}'), P\big),$$

as claimed. ◀

The following lemma gives a polar meaning to the notion of ε-nets.

▶ **Lemma 3.9.** *Let $N \subseteq P$ be an ε-net for P such that if $\mathbf{o} \in \textsc{ch}(P)$, then also $\mathbf{o} \in \textsc{ch}(N)$. For any point $x \in \mathbb{R}^d$, it holds that if $x \in \textsc{fh}(N^*)$, then $\big|V_x\big(P^*\big)\big| \leq \varepsilon|P|$.*

Proof. First, suppose that $\mathbf{o} \in \text{CH}(P)$, then, we have $\mathbf{o} \in \text{CH}(N)$, and hence $\mathbf{o} \in \text{FH}(N^*)$. Since $x \in \text{FH}(N^*)$, we have $x \in \rho_{\mathbf{o}}(p)$, for all $p \in N$. By Lemma 3.1, we get $p \in \rho_{\mathbf{o}}(x)$, for all $p \in N$, so $N \cap \rho_\infty(x) = \emptyset$. Since N is an ε-net for P, we conclude $|P \cap \rho_\infty(x)| \leq \varepsilon|P|$. The claim now follows, because by Lemma 3.1, we have $|P \cap \rho_\infty(x)| = |V_x(P^*)|$.

Second, suppose that $\mathbf{o} \notin \text{CH}(P)$, then, we also get $\mathbf{o} \notin \text{CH}(N)$, and hence $\mathbf{o} \notin \bigcup_{H \in N^*} H$. Since $x \in \text{FH}(N^*)$, we have $x \in \rho_\infty(p)$, for all $p \in N$. By Lemma 3.1, we get $p \in \rho_\infty(x)$, for all $p \in N$, so $N \cap \rho_{\mathbf{o}}(x) = \emptyset$. Since N is an ε-net for P, we conclude $|P \cap \rho_{\mathbf{o}}(x)| \leq \varepsilon|P|$. The claim now follows, because by Lemma 3.1, we have $|P \cap \rho_{\mathbf{o}}(x)| = |V_x(P^*)|$. ◀

4 A Simple Algorithm

Let P be a valid set of n points and let $\mathcal{H}$ be a valid set of m halfspaces in $\mathbb{R}^d$ such that either (1) P is avoiding while $\mathcal{H}$ is embracing, or (2) P is embracing while $\mathcal{H}$ is avoiding. We first present a slightly more restrictive algorithm that requires conditions (1) or (2) to hold. We spend the next few sections proving its correctness and running time, and then we extend it to a general algorithm for the ACIT problem.

4.1 Description of the Algorithm

Our algorithm $\text{TEST}(P, \mathcal{H})$ takes P and $\mathcal{H}$ as input, such that either (1) or (2) is satisfied, and it computes either the closest pair between $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$, if $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ are disjoint, or the closest pair between $\text{CH}(H^*)$ and $\text{FH}(P^*)$, if $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ intersect. By Theorem 3.6, this is always possible.

The algorithm is recursive. Let $\alpha = c\,d^4 \log d$, for some appropriate constant $c > 0$. For the base case, if both $|P|, |\mathcal{H}| \leq \alpha$, we apply the brute force algorithm: we explicitly compute the polytope $\text{CH}(P)$ to obtain the set $\mathcal{H}_P$ of hyperplanes with $\text{CH}(P) = \text{FH}(\mathcal{H}_P)$, and we use a classic LP-type algorithm to find the closest pair between $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ or between $\text{FH}(P^*)$ and $\text{CH}(\mathcal{H}^*)$. Otherwise, we compute a $(1/d^4)$-net $N \subseteq P$, and if necessary, we add $d+1$ points to N such that if $\mathbf{o} \in \text{CH}(P)$, then $\mathbf{o} \in \text{CH}(N)$. These $d+1$ points can be found in $O(n)$ time using basic linear algebra. Then, we execute the following loop.

Repeat $2d+1$ times: Recursively call $\text{TEST}(\mathcal{H}^*, N^*)$; there are two possibilities.

Case 1: $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$ are disjoint. Then, $\text{TEST}(\mathcal{H}^*, N^*)$ returns the closest pair x, y, with $x \in \text{CH}(\mathcal{H}^*)$ and $y \in \text{FH}(N^*)$ (unique by our general position assumption). Let $V_y \subset P^*$ be conflict set of P^* and y. If $V_y = \emptyset$, then report that $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ intersect, and output x, y as the polar witness. Otherwise, add to N all elements of ${V_y}^*$, and continue with the next iteration.

Case 2: $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$ intersect. Then, $\text{TEST}(\mathcal{H}^*, N^*)$ returns the closest pair x, y between $\text{FH}((\mathcal{H}^*)^*) = \text{FH}(\mathcal{H})$ and $\text{CH}((N^*)^*) = \text{CH}(N)$, with $x \in \text{FH}(\mathcal{H})$ and $y \in \text{CH}(N)$. Let H be the halfspace that contains $\text{FH}(\mathcal{H})$ supported by the normal hyperplane of xy through y. Let V_H be the conflict set of P and H. If $V_H = \emptyset$, then report that $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ are disjoint, and output x, y as the witness. Otherwise, add to N all elements of V_H and continue with the next iteration.

If the loop terminates without a result, the algorithm finishes and returns an ERROR.

4.2 Running Time

While at this point we have no idea why $\text{TEST}(P, \mathcal{H})$ works, we can start by analyzing its running time. In the base case, when both P and $\mathcal{H}$ have of at most α elements, we can compute $\text{CH}(P)$ explicitly to obtain the $O\left(\alpha^{\lfloor d/2 \rfloor}\right)$ halfspaces of $\mathcal{H}_P$. We can do this

in a brute force manner by trying all d-tuples of P^d and checking whether all of P is on the same side of the hyperplane spanned by a given tuple, or we can run a convex-hull algorithm [12, 7]. The former approach has a running time $O\big(\alpha^{d+1}\big)$, while the latter needs $O\big(\alpha^{\lfloor d/2 \rfloor}\big)$ time [12, 7]. Once we have $\mathcal{H}_P$ at hand, we can run standard LP-type algorithms with $O(\alpha^{\lfloor d/2 \rfloor})$ constraints to determine the closest pair either between $\textsc{ch}(P)$ and $\textsc{fh}(\mathcal{H})$, or between $\textsc{ch}(\mathcal{H}^*)$ and $\textsc{fh}(P^*)$. The running time of such algorithms is $O(d^{O(d)}\alpha^{\lfloor d/2 \rfloor}) = O(d^{O(d)})$ [3].

To see what happens in the main loop of the algorithm, we apply the theory of ε-nets, as described in the Section 3. As mentioned there, the initial set N is a $(1/d^4)$-net for P. Thus, the size of each V_H added to N in Case 2 of the main loop of our algorithm is at most n/d^4. Using Lemma 3.9, the same holds for any set V_y added in Case 1. Thus, regardless of the case, the size of N at the beginning of the i-th loop iteration is at most $\max\{in/d^4, \alpha\}$.

The main loop runs for at most $2d + 1$ iterations. Thus, the size of N never exceeds $(2d + 1)n/d^4 \leq \beta n/d^3$, for some constant $\beta > 0$. Since the algorithm to compute the $(1/d^4)$-net N for P runs in time $O(d^{O(d)}n)$ [3, 8], we obtain the following recurrence for the running time:

$$T(n, m) \leq \begin{cases} \mathrm{LP}\Big(\alpha + \alpha^{\lfloor d/2 \rfloor}, d\Big) + O\big(\alpha^{\lfloor d/2 \rfloor}\big), & \text{if } n, m \leq \alpha, \\ (2d + 1) \cdot T\Big(m, \max\big\{\beta n/d^3, \alpha\big\}\Big) + O(d^{O(d)}n), & \text{otherwise.} \end{cases}$$

We look further into $T\big(m, \max\big\{\beta n/d^3, \alpha\big\}\big)$ and notice that if we do not reach the base case, then unfolding the recursion by one more step yields

$$T\big(m, \max\{\beta n/d^3, \alpha\}\big) \leq (2d + 1) \cdot T\big(\max\{\beta n/d^3, \alpha\}, \max\{\beta m/d^3, \alpha\}\big) + O(d^{O(d)}m).$$

Thus, by contracting two steps into one, we get the following more symmetric relation:

$$T(n, m) \leq (2d + 1)^2 \cdot T\big(\max\{\beta n/d^3, \alpha\}, \max\{\beta m/d^3, \alpha\}\big) + O\big(d^{O(d)}(n + m)\big),$$

for sufficiently large n and m. Together with the base case, one can show by induction that this yields a running time of $O(d^{O(d)}(n + m))$.

Remark. Because the best deterministic algorithm know for LP-type problems with n constraints runs also in time $O(d^{O(d)}n)$ [3], substantial improvements on the running time of our problem seem out of reach. If we allow randomization however, then we can improve in two places. First of all, by randomly sampling $O(\alpha \log n)$ elements of P, we obtain a $(1/d^4)$-net of P with high probability. Secondly, the base case could be solved with faster algorithms. The best known randomized algorithms for LP-type problems with n constraints have a running time of $O(d^2n + 2^{O(\sqrt{d \log d})})$, which substantially improves the dependency on d. Alternatively, we could use methods to solve convex quadratic programs in the base case to find the closest pair between two H-polytopes.

4.3 Correctness

We show that $\textsc{Test}(P, \mathcal{H})$ indeed tests whether $\textsc{ch}(P)$ and $\textsc{fh}(\mathcal{H})$ intersect. First, we verify that the input to each recursive call $\textsc{Test}(\cdot, \cdot)$ satisfies either condition (1) or (2).

▶ **Lemma 4.1.** *Let $P \subset \mathbb{R}^d$ be a finite point set and $\mathcal{H}$ a finite set of halfspaces in d dimensions, such that either* (1) *P is avoiding while $\mathcal{H}$ is embracing, or* (2) *P is embracing while $\mathcal{H}$ is avoiding. Consider a call of $\textsc{Test}(P, \mathcal{H})$. Then, the input to each recursive call satisfies either condition* (1) *or condition* (2)*.*

Proof. We do induction on the recursion depth. The base case holds by assumption. For the inductive step, we note that if the input $(P, \mathcal{H})$ satisfies condition (1), then $(N, \mathcal{H})$ also satisfies condition (1), for any subset $N \subseteq P$. For condition (2), first note that our algorithm ensures that if $\mathbf{o} \in \textsc{ch}(P)$, then also $\mathbf{o} \in \textsc{ch}(N)$. This implies that if $(P, \mathcal{H})$ satisfies condition (2), then $(N, \mathcal{H})$ also satisfies condition (2). Finally, if $(P, \mathcal{H})$ satisfies condition (1), then by Corollary 3.5 $\left(\mathcal{H}^*, P^*\right)$ satisfies condition (2), and vice-versa. The claim follows. ◀

We are now ready for the correctness proof. We show that $\textsc{Test}(P, \mathcal{H})$, with P and $\mathcal{H}$ satisfying either (1) or (2), computes either the closest pair between $\textsc{ch}(P)$ and $\textsc{fh}(\mathcal{H})$, if they are disjoint, or the closest pair between $\textsc{ch}(\mathcal{H}^*)$ and $\textsc{fh}(P^*)$, if the polytopes intersect.

We use induction on $\max\{|P|, |\mathcal{H}|\}$. For the base case, when the maximum is at most α, our algorithm uses the brute-force method. This certainly provides a correct answer, by our assumptions on P and $\mathcal{H}$ and by Theorem 3.6.

For the inductive set, we may assume that each recursive call to $\textsc{Test}(\cdot, \cdot)$ provides a correct answer. It remains to show (i) that the main loop succeeds in at most $2d+1$ iterations; and (ii) if the main loop succeeds, the algorithm returns a valid closest pair.

Number of Iterations. We show that the algorithm will never return an Error, i.e., that the loop will succeed in at most $2d + 1$ iterations. To start, we observe that the cases in the algorithm cannot alternate: first, we encounter only Case 2, then, we encounter only Case 1.

▶ **Lemma 4.2.** *It the main loop in algorithm* $\textsc{Test}(P, \mathcal{H})$ *encounters Case 1, it will never again encounter Case 2.*

Proof. In each unsuccessful iteration, the set N grows by at least one element, so the convex polytope $\textsc{fh}(N^*)$ becomes smaller. Once $\textsc{fh}(N^*)$ and $\textsc{ch}(\mathcal{H}^*)$ are disjoint, they will remain disjoint for the rest of the algorithm, and by our inductive hypothesis, this will be reported correctly by the recursive calls to $\textsc{Test}(\cdot, \cdot)$. ◀

We now bound the number of iterations in Case 2.

▶ **Lemma 4.3.** *The algorithm can have at most* $d + 1$ *iterations in Case 2. If there are* $d + 1$ *iterations in Case 2, then the last iteration is successful and the algorithm terminates.*

Proof. Suppose there are at least $d + 2$ iterations in Case 2. By Lemma 4.2, each iteration until this point encounters Case 2. Let $N_1 \subset N_2 \subset \cdots \subset N_{d+2}$ be the set N at the beginning of the first $d + 2$ iterations in Case 2. By Lemma 3.7 and the inductive hypothesis, each time we run into Case 2 unsuccessfully, the distance between $\textsc{ch}(N)$ and $\textsc{fh}(\mathcal{H})$ decreases strictly. Since the first $d + 1$ iterations in Case 2 are not successful, this means $d(\textsc{ch}(N_i), \textsc{fh}(\mathcal{H})) > d(\textsc{ch}(N_{i+1}), \textsc{fh}(\mathcal{H}))$, for $i = 1, \ldots, d + 1$.

Because the $(d+2)$-th iteration runs into Case 2, it follows that $\textsc{ch}(N_{d+2})$ does not intersect $\textsc{fh}(\mathcal{H})$. Let x, y be the closest pair between $\textsc{fh}(\mathcal{H})$ and $\textsc{ch}(N_{d+2})$, with $y \in \textsc{ch}(N_{d+2})$. Then, y must lie on a face of $\textsc{ch}(N_{d+2})$, and by Carathéodory's theorem [20, Theorem 1.2.3], there is a set $B \subseteq N_{d+2}$ with at most d elements such that $y \in \textsc{ch}(B_{d+2})$. We claim that in each prior iteration $i = 1, \ldots, d + 1$, the conflict set V_H must contain at least one new element of B. Otherwise, if all the elements of B were already in some N_i, with $i \leq d + 1$, then $\textsc{ch}(N_i)$ would contain y and hence have a distance to $\textsc{fh}(\mathcal{H})$ smaller or equal than $\textsc{ch}(N_{d+2})$, leading to a contradiction. Similarly, if in an iteration $i \leq d + 1$, all elements of B were contained in H, then the distance between $\textsc{ch}(N_i)$ and $\textsc{fh}(\mathcal{H})$ could not strictly decrease, by Lemma 3.7. However, B contains only d elements, and we have $d + 1$ iterations, so we

cannot add a new element of B at the end of each iteration Thus, the main loop can have at most d unsuccessful iterations in Case 2 before either encountering Case 2 successfully or reaching Case 1. ◀

▶ **Lemma 4.4.** *The algorithm can have at most $d+1$ iterations in Case 1.*

Proof. Similar to the proof of Lemma 4.3, assume that we have at least $d+2$ iterations in Case 1. Let $N_1 \subset N_2 \subset \cdots \subset N_{d+2}$ be the set N at the beginning of each such iteration. By Lemma 3.8 and the inductive hypothesis, each time we encounter Case 1 unsuccessfully, we strictly increase the distance between $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$. That is, $d(\text{CH}(\mathcal{H}^*), \text{FH}({N_i}^*)) > d(\text{CH}(\mathcal{H}^*), \text{FH}({N_{i+1}}^*))$, for $i = 1, \ldots, d+1$.

Because we run into Case 1 in the $(d+2)$-th iteration, it follows that $\text{CH}(\mathcal{H}^*)$ and $\text{FH}({N_{d+2}}^*)$ do not intersect. Let x, y be the closest pair between $\text{CH}(\mathcal{H}^*)$ and $\text{FH}({N_{d+2}}^*)$, with $y \in \text{FH}({N_{d+2}}^*)$. Let B be the at most d elements in N_{d+2} such that x, y is the closest pair of $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(B^*)$. Note that y could either be a vertex of $\text{FH}(B^*)$, or lie in the relative interior of one of its faces. Observe that in each unsuccessful iteration in Case 1, V_y must include a new member of B. Otherwise, if all the elements of B were already in some N_i with $i \leq d+1$, then $\text{FH}({N_i}^*)$ would have a distance to $\text{CH}(\mathcal{H}^*)$ larger or equal than $\text{FH}({N_{d+2}}^*)$, leading to a contradiction. Similarly, if in an iteration $i \leq d+1$, all elements of B^* were not in conflict with y, then the distance between $\text{FH}({N_i}^*)$ and $\text{CH}(\mathcal{H}^*)$ could not strictly decrease, by Lemma 3.8. However, this is impossible, because B has d elements and we have at least $d+1$ unsuccessful iterations. Thus, in the $(d+1)$-th iteration at the latest, we would observe that V_y is empty and the algorithm would finish. That is, the main loop can run for at most d unsuccessful iterations in Case 1. ◀

Lemmas 4.2, 4.3, and 4.4 guarantee that the algorithm will finish successfully within $2d+1$ iterations and that it will never return an Error. It remains to argue that the algorithm reports a correct closest pair if one of the two cases is encountered successfully.

Correctness of the Closest Pair. We first analyze the success condition of Case 1, i.e., when $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$ are disjoint. This condition is triggered when we have a set $N \subseteq P$ and a point $y \in \text{FH}(N^*)$ such that $V_y = \emptyset$. By Lemma 3.8, the closest pair x, y between $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$ then coincides with the closest pair between $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(P^*)$. In particular, this implies that $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(P^*)$ are disjoint. Because the recursive call returns correctly the closest pair x, y between $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$ by induction, it follows that the algorithm correctly returns the closest pair between $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(P^*)$.

Next, we analyze the success condition of Case 2, i.e., when $\text{CH}(\mathcal{H}^*)$ and $\text{FH}(N^*)$ intersect. This implies by Theorem 3.6 that $\text{CH}(N)$ and $\text{FH}(\mathcal{H})$ are disjoint. Let x, y be the closest pair between $\text{CH}(N)$ and $\text{FH}(\mathcal{H})$, with $y \in \text{CH}(N)$. The success condition of Case 2 is triggered when $V_H = \emptyset$. By Lemma 3.7, this means that x, y coincides with the closest pair between $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$. In particular, $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ are disjoint. Because the recursive call returns correctly the closest pair x, y between $\text{CH}(N)$ and $\text{FH}(\mathcal{H})$ by induction, the algorithm correctly returns the closest pair between $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$. This now shows that $\text{Test}(P, \mathcal{H})$ is indeed correct.

4.4 The Final Algorithm

Finally, we show how to remove the initial assumption that $(P, \mathcal{H})$ satisfies either condition (1) or condition (2).

▶ **Theorem 4.5.** *Let P be a set of n points in $\mathbb{R}^d$ and let $\mathcal{H}$ be a set of m halfspaces in $\mathbb{R}^d$. We can test if $\text{CH}(P)$ and $\text{FH}(\mathcal{H})$ intersect in $O(d^{O(d)}(n+m))$ time. If they do, then we compute a point in their intersection; otherwise, we compute a separating plane.*

Proof. Recall that our algorithm requires that either (1) $\mathbf{o} \notin \text{CH}(P)$ and $\mathbf{o} \in \text{FH}(\mathcal{H})$, or (2) $\mathbf{o} \in \text{CH}(P)$ and $\mathbf{o} \notin \bigcup_{H \in \mathcal{H}} H$ to work, which might not hold for the given P and $\mathcal{H}$.

Thus, before running $\text{TEST}(P, \mathcal{H})$, we first compute a point in the interior of $\text{FH}(\mathcal{H})$ using standard linear programming in $d+1$ dimensions. More precisely, consider the linear program $\max e$, subject to $Ax + e \cdot \mathbf{1} \leq \mathbf{1}$, $e \leq 1$, where $x \in \mathbb{R}^d$, $e \in \mathbb{R}$ are variables, $\mathbf{1}$ is the m-dimensional all-ones-vector, and $A \in \mathbb{R}^{m \times d}$ is the matrix whose rows are the normal vectors of the hyperplanes that bound the halfspaces in $\mathcal{H}$. Let (x^*, e^*) be an optimal solution. Then, if $e^* < 0$, the intersection $\text{FH}(\mathcal{H})$ is empty, and if $e^* = 0$, the intersection $\text{FH}(\mathcal{H})$ is not fully dimensional. Otherwise, if $e^* > 0$, the point x^* lies in the interior of $\text{FH}(\mathcal{H})$. We change the coordinate system so that x^* coincides with $\mathbf{o}$. Next, we use standard linear programming to test if $\mathbf{o} \in \text{CH}(P)$. These two linear programs have a running time of $O(d^{O(d)}(n+m))$ [3] Notice that if $\mathbf{o} \in \text{CH}(P)$, then we are done. Otherwise, we guarantee that condition (1) is satisfied, and we can run $\text{TEST}(P, \mathcal{H})$ in $O(d^{O(d)}(n+m))$ time. ◀

References

1. Luis Barba and Stefan Langerman. Optimal detection of intersections between convex polyhedra. In *Proc. 26th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1641–1654, 2015.
2. Timothy M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 430–436, 2004.
3. Timothy M Chan. Improved deterministic algorithms for linear programming in low dimensions. *ACM Trans. Algorithms*, 14(3):30, 2018.
4. Timothy M Chan. Personal communication, 2018.
5. B. Chazelle and D. Dobkin. Intersection of Convex Objects in Two and Three Dimensions. *J. ACM*, 34(1):1–27, January 1987.
6. Bernard Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21:586–591, 1992.
7. Bernard Chazelle. An Optimal Convex Hull Algorithm in Any Fixed Dimension. *Discrete Comput. Geom.*, 10:377–409, 1993.
8. Bernard Chazelle. *The Discrepancy Method - Randomness and Complexity.* Cambridge University Press, 2001.
9. Bernard Chazelle and David Dobkin. Detection is Easier than Computation (Extended Abstract). In *Proc. 12th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 146–153, 1980.
10. Vašek Chvátal. *Linear programming.* A Series of Books in the Mathematical Sciences. W. H. Freeman, 1983.
11. Kenneth L. Clarkson. A Las Vegas Algorithm for Linear Programming When the Dimension Is Small. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 452–456, 1988.
12. Kenneth L. Clarkson and Peter W. Shor. Application of Random Sampling in Computational Geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
13. David Dobkin and David Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6(3):381–392, 1985.
14. Herbert Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Algorithms*, 6(2):213–224, 1985.

15 Sariel Har-Peled. *Geometric approximation algorithms*. American Mathematical Society, 2011.

16 Sanjiv Kapoor and Pravin M. Vaidya. Fast Algorithms for Convex Quadratic Programming and Multicommodity Flows. In *Proc. 18th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 147–159, 1986.

17 M. K. Kozlov, S. P. Tarasov, and L. G. Hačijan. The polynomial solvability of convex quadratic programming. *Zh. Vychisl. Mat. i Mat. Fiz.*, 20(5):1319–1323, 1359, 1980.

18 Guy Louchard, Stefan Langerman, and Jean Cardinal. Randomized Optimization: a Probabilistic Analysis. *Discrete Mathematics & Theoretical Computer Science*, 2007.

19 Jiří Matoušek. Linear Optimization Queries. *J. Algorithms*, 14(3):432–448, 1993.

20 Jiří Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag, 2002.

21 David E. Muller and Franco P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7(2):217–236, 1978.

22 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry–An Introduction*. Springer-Verlag, 1985.

23 Raimund Seidel. Small-Dimensional Linear Programming and Convex Hulls Made Easy. *Discrete Comput. Geom.*, 6:423–434, 1991.

24 Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. *Proc. 9th Sympos. Theoret. Aspects Comput. Sci. (STACS)*, pages 567–579, 1992.

25 Günter M. Ziegler. *Lectures on Polytopes*. Springer-Verlag, 1995.

Relaxed Voronoi: A Simple Framework for Terminal-Clustering Problems*

Arnold Filtser[1]
Ben-Gurion University of the Negev, Be'er Sheva, Israel
arnoldf@cs.bgu.ac.il

Robert Krauthgamer[2]
Weizmann Institute of Science, Rehovot, Israel
robert.krauthgamer@weizmann.ac.il

Ohad Trabelsi[3]
Weizmann Institute of Science, Rehovot, Israel
ohad.trabelsi@weizmann.ac.il

Abstract

We reprove three known algorithmic bounds for terminal-clustering problems, using a single framework that leads to simpler proofs. In this genre of problems, the input is a metric space (X, d) (possibly arising from a graph) and a subset of terminals $K \subset X$, and the goal is to partition the points X such that each part, called a cluster, contains exactly one terminal (possibly with connectivity requirements) so as to minimize some objective. The three bounds we reprove are for Steiner Point Removal on trees [Gupta, SODA 2001], for Metric 0-Extension in bounded doubling dimension [Lee and Naor, unpublished 2003], and for Connected Metric 0-Extension [Englert et al., SICOMP 2014].

A natural approach is to cluster each point with its closest terminal, which would partition X into so-called Voronoi cells, but this approach can fail miserably due to its stringent cluster boundaries. A now-standard fix, which we call the `Relaxed-Voronoi` framework, is to use enlarged Voronoi cells, but to obtain disjoint clusters, the cells are computed greedily according to some order. This method, first proposed by Calinescu, Karloff and Rabani [SICOMP 2004], was employed successfully to provide state-of-the-art results for terminal-clustering problems on general metrics. However, for restricted families of metrics, e.g., trees and doubling metrics, only more complicated, ad-hoc algorithms are known. Our main contribution is to demonstrate that the `Relaxed-Voronoi` algorithm is applicable to restricted metrics, and actually leads to relatively simple algorithms and analyses.

2012 ACM Subject Classification Theory of computation → Random projections and metric embeddings, Theory of computation → Graph algorithms analysis

Keywords and phrases Clustering, Steiner point removal, Zero extension, Doubling dimension, Relaxed voronoi

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.10

Related Version Full version at arXiv:1809.00942

* In earlier versions this algorithm was called "Noisy Voronoi".

[1] Work partially supported by the Lynn and William Frankel Center for Computer Sciences, ISF grant 1817/17, and by BSF Grant 2015813.
[2] Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, a Minerva Foundation grant, and a Google Faculty Research Award.
[3] Work partly done at IBM Almaden.

1 Introduction

We consider *terminal clustering* problems, where the input is a metric space (X, d) with k terminals $K \subseteq X$, and the goal is to partition the points (vertices) into k clusters, each containing exactly one terminal, so as to minimize some objective. In the *graphical version* of this problem, the input is a weighted graph $G = (V, E, w)$ with terminals $K \subset V$ and the metric d is derived as the shortest-path metric on $X = V$ with respect to the non-negative edge weights w, and every output cluster should be connected (as an induced subgraph of G).

We present for these problems a simple algorithmic framework that generalizes two different known algorithms, from [3, 10]. Using this framework, we obtain simple algorithms for two specific metric/graph classes, and recover their known bounds from [13, 19, 7] in a unified manner that is arguably simpler and more insightful than previous work. In our case, even the analysis is short and simple. Thus, our main contribution is to identify and present the framework, and to (non-trivially) apply it to specific metric/graph classes, and we hope it will lead to new results in the future. We proceed to define the two specific problems that we investigate, and briefly survey their known bounds.

Metric 0-Extension (M0E). In this problem, the input is a metric space (X, d) and a set of k terminals $K \subset X$, and the goal is to find a distribution $\mathcal{D}$ over retractions f (i.e., functions $f : X \to K$ that satisfy $f(x) = x$ for all $x \in K$), such that

$$\forall x, y \in X, \qquad \mathop{\mathbb{E}}_{f \sim \mathcal{D}} [d(f(x), f(y))] \leq \alpha \cdot d(x, y),$$

where $\alpha \geq 1$, called the *expected distortion*, is as small as possible. Throughout, we seek the smallest α that holds for a class of metric spaces, for example all metrics with k terminals, and then $\alpha = \alpha(k)$.

The above is closely related to the well-known 0-*Extension* problem, in which the input is a set X, a terminal set $K \subseteq X$, a metric d_K over the terminals and a cost function $c : \binom{X}{2} \to \mathbb{R}_+$, and the goal is to find a retraction $f : X \to K$ that minimizes $\sum_{\{x,y\} \in \binom{X}{2}} c(x, y) \cdot d_K(f(x), f(y))$. The 0-Extension problem, first proposed by Karzanov [17], generalizes the Multiway Cut problem [6] by allowing d_K to be any discrete metric (instead of a uniform metric) and it is also a special case of the Metric Labeling problem [18], whose objective function has additional terms that represent assignment costs. Karzanov introduced a linear programming (LP) relaxation for 0-Extension, which can be described as finding a (semi-)metric d_X over X that agrees with d_K on K, and minimizes $\sum_{\{x,y\} \in \binom{X}{2}} c(x, y) \cdot d_X(x, y)$. Rounding this LP relaxation is equivalent to the M0E problem (by the minimax theorem). Consequently, most previous work on 0-Extension has actually focused on solving M0E, and so does our work.

A well-known open problem is to determine the smallest distortion $\alpha(k)$ that suffices for all metric spaces with k terminals. The currently known bounds are $O(\log k / \log \log k)$ due to Fakcharoenphol, Harrelson, Rao, and Talwar [8] (improving over [3]), and $\Omega(\sqrt{\log k})$ due to Calinescu, Karloff and Rabani [3]. Improved upper bounds are known for special classes of metric spaces X, for example $O(1)$ for the case where X is the shortest-paths metric of a graph excluding a fixed minor [3]. Another example is when the submetric on the terminals (i.e., the restriction of d to K) is β-decomposable, which admits an $O(\beta)$ upper bound [19] (a somewhat similar bound was obtained in [1]). This implies an $O(\mathrm{ddim}(K))$ upper bound, where $\mathrm{ddim}(K)$ denotes the doubling dimension of the terminals' submetric (see Section 2 for definition), and our results reproduce the latter bound.

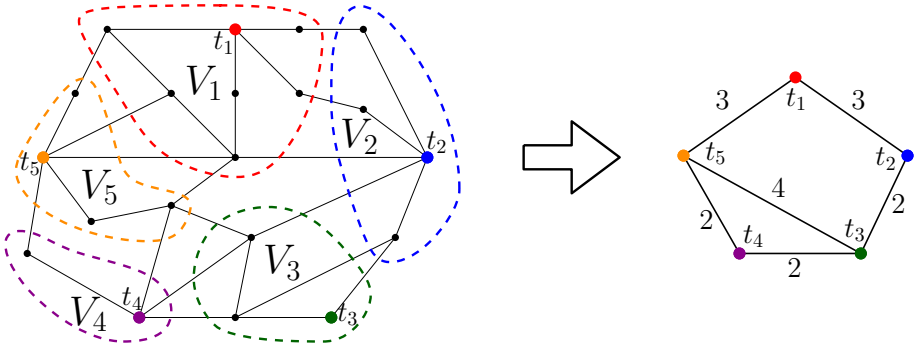


Figure 1 Example how a terminal partition of graph G (on left) induces a minor M (on right). The graph shown has unit weight edges and 5 terminals, and the terminal partition is shown using dashed curves. The distortion is $\frac{d_M(t_2,t_5)}{d_G(t_2,t_5)} = \frac{6}{2} = 3$.

Steiner Point Removal (SPR). In this problem, given a weighted graph $G = (V, E, w)$ and a set of terminals $K \subseteq V$, the goal is to find a minor $M = (K, E')$ of G (note its vertex set is exactly the set of terminals), that approximately preserves the distances between terminals, which means (using d_H to denote the shortest-path metric in H) that

$$\forall t, t' \in K, \qquad d_G(t, t') \le d_M(t, t') \le \alpha \cdot d_G(t, t'),$$

where $\alpha \ge 1$, called the *distortion*, is as small as possible. Again, we seek the best α that holds for a class of graphs, say all graphs with $k = |K|$ terminals.

Let us denote $K = \{t_1, \ldots, t_k\}$. A partition $\{V_1, \ldots, V_k\}$ of V is called a *terminal partition* (with respect to K) if for all $i = 1, \ldots, k$, the induced subgraph $G[V_i]$ is connected and contains t_i. The *induced minor* M of such a terminal partition is the minor obtained by contracting each V_i into a single vertex called (abusing notation) t_i. Thus, M has an edge between t_i and t_j iff G has an edge between V_i and V_j. The weight of this edge (if exists) is simply $d_G(t_i, t_j)$, which represents the shortest-path in G; see Figure 1 for an example. Most of the work on SPR so far used terminal partitions to obtain a minor, and so does our work.

For the case where the graph G is a tree, the smallest distortion possible for SPR is known to be 8. Gupta [13] constructed a tree achieving distortion 8; in fact, he was only interested in constructing a tree with vertex set K, and later Chan, Xia, Konjevod, and Richa [4] observed that Gupta's tree is actually a minor of the given tree G. Surprisingly, they further showed that 8 is the best possible distortion for the family of trees, as (unweighted) complete binary trees require distortion $8 - \epsilon$. Our results reproduce this upper bound of 8.

For SPR in general graphs there is currently a huge gap. The best lower bound known is just 8, known for trees, and recently Filtser [9] showed an $O(\log k)$ upper bound (improving over [16, 5]). No better upper bound is known even for seemingly much simpler cases such as planar graphs, and the only other bound known is $\alpha = O(1)$ for outerplanar graphs [2].

1.1 Algorithmic Framework

A natural and straightforward algorithm for terminal clustering is to simply partition the metric (or graph) into Voronoi cells, i.e., map each point (or vertex) to its closest terminal, to obtain a partition of X (or V) with one cluster for each terminal. However, there are easy examples where this algorithm fails miserably, because of the stringent cluster boundaries. A now-standard fix is to build around each terminal (iteratively) a cluster that is an enlarged Voronoi cell in the remaining metric (or graph).

Algorithm 1 `Metric-Relaxed-Voronoi`.

input : metric $M = (X, d)$, terminals K, ordering $\pi = (t_1, \ldots, t_k)$, magnitudes $R_1, \ldots, R_k \geq 1$
output : retraction $f : X \to K$ (i.e., $\forall x \in K,\ f(x) = x$)

```
1 for j = 1, ..., k do
2     for all unmapped points x such that d(t_j, x) ≤ R_j · D(x) do
3         set f(x) = t_j
4 return f
```

This approach was first used by Calinescu, Karloff and Rabani [3]. We generalize their method, so that all previous uses of this approach can be seen as instantiations of our algorithm with specific parameters. Our algorithm, called `Relaxed-Voronoi`, is formally described in Algorithm 1 where throughout we define

$$D(x) = d(x, K) = \min_{t \in K} d(x, t)$$

to be the distance from $x \in X$ to its closest terminal. The algorithm's parameters, formally presented as part of the input, are an ordering $\pi = (t_1, \ldots, t_k)$ of the terminals and corresponding magnitudes $R_1, \ldots, R_k \geq 1$ (one for each terminal). The algorithm is rather simple; each terminal t_j, in turn according to the ordering, creates a cluster $V_j = f^{-1}(t_j)$ containing all yet-unclustered points x at distance $d(x, t_j) \leq R_j \cdot D(v)$. That is, the cluster of t_j is a Voronoi cell "enlarged" by factor R_j in the remaining metric. Setting $R_1 = \cdots = R_k = 1$ recovers the partition into Voronoi cells.

The above algorithm cannot be used as is for the SPR problem, because a terminal partition has an additional connectivity requirement. Therefore, in the graphical case, instead of taking all remaining vertices x that satisfy $d_G(x, t_j) \leq R_j \cdot D(v)$, we create V_j in a Dijkstra-like iterative fashion, as follows. Initially $V_j = \{t_j\}$, and we repeatedly add to V_j any unclustered vertex that has a neighbor in V_j and is at distance $d_G(v, t_j) \leq R_j \cdot D(v)$. See Algorithms 2 and 3 for a formal description. This version of the `Relaxed-Voronoi` algorithm was first proposed by Filtser [10] for the SPR problem in general graphs. It is simpler to describe and to analyze than the `Ball-Growing` algorithm of previous work [16, 5, 9].[4] Filtser also showed that the `Relaxed-Voronoi` algorithm can be implemented in time $O(|E| \log |V|)$.[5]

1.2 Our Contribution

All previous uses of the `Relaxed-Voronoi` algorithm were on general metrics or graphs. Specifically, Calinescu et al. [3] and Fakcharoenphol et al. [8], used a uniformly random ordering π and a single random magnitude R (same for all terminals), and Filtser [10] used an arbitrary ordering π and magnitudes that are independently and identically distributed (i.i.d.) drawn from an exponential-like distribution. However, for special families of metrics or graphs,

[4] The `Ball-Growing` algorithm creates clusters in rounds, where each round iteratively enlarges every cluster, by increasing its radius around each terminal (in the remaining graph) by a value sampled from an exponential distribution.

[5] The $O(|E| \log |V|)$-time in [10] actually implements a slightly different algorithm, where the test $d_G(v, t_j) \leq R_j \cdot D(v)$ (line 4) is replaced by $d_{G[V_j \cup \{v\}]}(v, t_j) \leq R_j \cdot D(v)$. The distortion bound holds for this algorithm too.

Algorithm 2 `Graphic-Relaxed-Voronoi`.

input : weighted graph $G = (V, E, w)$, terminals K, ordering $\pi = (t_1, \ldots, t_k)$, magnitudes $R_1, \ldots, R_k \geq 1$
output : Minor M

$V_\perp \leftarrow V \setminus K$ // $V_\perp$ *is the currently unclustered vertices.*
for $j = 1, \ldots, k$ **do**
  $V_j \leftarrow$ `Create-Cluster`$(G, V_\perp, t_j, R_j)$
  $V_\perp \leftarrow V_\perp \setminus V_j$
return the terminal-centered minor M of G induced by $V_1, \ldots, V_k$

Algorithm 3 `Create-Cluster`.

input : weighted graph $G = (V, E, w)$, unclustered vertices $V_\perp$, terminal t_j, magnitude R_j
output : cluster V_j

$V_j \leftarrow \{t_j\}$, $U \leftarrow \emptyset$, $N \leftarrow \{$all neighbors of t_j in $V_\perp\}$
while $N \neq \emptyset$ **do**
  pick an arbitrary vertex $v \in N$ and remove it from N
  if $d_G(v, t_j) \leq R_j \cdot D(v)$ **then**
    add v to V_j
    add all the neighbors of v in $V_\perp \setminus (U \cup V_j)$ to N
  else
    add v to U
return V_j

this type of algorithm was never used; instead, ad-hoc algorithms were developed, leading to more involved algorithms and analyses. Our contribution is to tailor the `Relaxed-Voronoi` algorithm to special input families by choosing the ordering π deterministically but depending on the input at hand (rather than a random or arbitrary ordering). As a result, we reprove three known results using simpler algorithms and analyses. We believe that this approach will lead to additional and new results.

SPR on Trees. Gupta's algorithm [13], which achieves distortion 8, is designed specifically for trees and it is unclear how to generalize it. Its recursive definition makes it arguably difficult to understand intuitively how its output on a given tree would look like. For example, the fact that the algorithm is tight and produces a minor [4] was non-trivial and even surprising. This result has proved useful in the past, yet it is a bit mysterious why 8 is the optimal bound, i.e., what tradeoff does it optimize.

We use the `Relaxed-Voronoi` algorithm to construct a tree with optimal distortion 8. The choice of parameters in the algorithm is very simple – the magnitudes are all set to $R_j = 3$, and the ordering π is defined by listing the terminals in order of increasing distance from an arbitrary "root" vertex v (breaking ties arbitrarily). Our algorithm's description is simple and intuitive, its distortion bound 8 is explained by the analysis, and it is straightforward that the output tree is a minor of the input tree. Perhaps surprisingly, our algorithm outputs the same tree as Gupta's algorithm. Overall, our algorithm provides a better understanding of Gupta's celebrated result. We believe that this approach can be generalized to additional graph families, and hopefully achieve a constant distortion for SPR on (say) planar graphs (where the current bound is only $O(\log k)$, which holds for general graphs).

M0E on Doubling Metrics. Lee and Naor's [19] algorithm achieves $O(\text{ddim})$ when the submetric on the terminals (i.e., the metric's restriction to points in K) has doubling dimension at most ddim. Their algorithm is based on stochastic decompositions, specifically converting padded decompositions into separating decompositions, then defining (new) partial decompositions, and finally using these decompositions in all the possible distance scales.

We use the `Relaxed-Voronoi` algorithm to achieve the same $O(\text{ddim})$ upper bound, by setting the parameters as follows. The magnitudes R_j are i.i.d., each distributed like $2 \cdot e^Z$ where Z is drawn from an exponential distribution with parameter $\Theta(\text{ddim})$. We set π to be the Gonzalez order [12], where t_1 is an arbitrary terminal, and each successive t_i is the terminal farthest from $\{t_1, \ldots, t_{i-1}\}$, breaking ties arbitrarily. Our algorithm is much simpler, more elegant, and its straightforward implementation takes only $O(nk)$ time (assuming the input is given as a matrix of pairwise distances). We hope that our ideas could lead to a better upper bound for the SPR problem in the case where the metric restricted to the terminals has a bounded doubling dimension.

Connected M0E. This is a graphic version of the M0E problem. The input metric is the shortest-path metric of an edge-weighted graph $G = (V, E, w)$, and similarly to the M0E problem, the goal is to find a distribution over retractions $f : V \to K$, but with an additional requirement: each cluster $f^{-1}(t_j)$ must be connected (as a subgraph of G). Englert et al. [7] achieved for this problem expected distortion $\alpha = O(\log k)$ using an algorithm that partitions the graph vertices into clusters using stochastic decompositions in all possible distance scales, and then merging some clusters to enforce connectivity. We use a graphic version of the `Relaxed-Voronoi` algorithm (which guarantees connectivity) to achieve the same expected distortion $O(\log k)$. When describing this algorithm, we abuse notation and identify $f(v) = t_j$ with $v \in V_j$, i.e., when the algorithm adds a vertex v to cluster V_j, it should be understood as also assigning $f(v) = t_j$. The graphic `Relaxed-Voronoi` algorithm is much simpler than the previous algorithm of [7], and we set its parameters as follows. The ordering π is arbitrary, and the magnitudes R_j are i.i.d., each distributed like e^Z where Z is drawn from an exponential distribution with parameter $\Theta(\log k)$. Even though this problem is concerned with general graphs and there is nothing clever about the ordering, we still chose to present this result, as it gives further evidence to the strength and broad applicability of the `Relaxed-Voronoi` algorithm. Another advantage is that it can be implemented in $O(|E| \log |V|)$ time, while the algorithm of [7] requires more time (an unspecified polynomial). See Footnote 5 for additional details.

1.3 Related Work

The Voronoi-like approach was used also in other recent algorithms. Gupta and Talwar [15] introduced the `Random-Rates` algorithm, in which each terminal t_j samples a rate $\rho_j \geq 1$, and then every point x is clustered with the terminal t_j that minimizes the ratio $\frac{d(x,t_j)}{\rho_j}$. The main difference from the `Relaxed-Voronoi` algorithm is that in their algorithm, the terminals create their clusters simultaneously (rather than sequentially), which does not guarantee that the clusters are connected. Gupta and Talwar [15] proved an $O(\log k)$ expected distortion for this algorithm on the M0E problem. It seems unlikely that their algorithm can provide $O(\text{ddim}(K))$ upper bound, which usually follows by bounding the number of clusters relevant to any "separation event" by $2^{O(\text{ddim}(K))}$. We achieve this using the sequential ordering, but in their algorithm too many clusters can be relevant.

Miller, Peng and Xu [20] introduced the `Parallel-Partition` algorithm to partition a graph into low-diameter clusters (without a given set of terminals). In this algorithm,

each vertex u samples a random shift $s_u \geq 0$, and then every vertex x joins the cluster of u with minimum $d(x,u) - s_u$. This algorithm produces connected clusters, however, it gets as an input a target diameter $\Delta > 0$, and its guarantees are proportional to this parameter. In contrast, the `Relaxed-Voronoi` algorithm is scale-free and handles all distances scales simultaneously (similar to the above `Random-Rates` algorithm), and therefore it is more natural for terminal-partitioning problems.

2 Preliminaries

Consider an undirected graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}_{\geq 0}$ and let d_G denote the shortest-path metric in G. For a subset of vertices $A \subseteq V$, let $G[A]$ denote the *induced graph* on A. Fix $K = \{t_1, \ldots, t_k\} \subseteq V$ to be a set of the given *terminals*. As mentioned earlier, for a vertex $v \in V$ we define $D(v) = \min_{t \in K} d_G(v,t)$ to be the distance from v to its closest terminal.

A graph H is a *minor* of a graph G if it can be obtained from G by edge deletions, edge contractions, and vertex deletions. As defined earlier, a partition $\{V_1, \ldots, V_k\}$ of V is called a *terminal partition* (with respect to K) if for all $i = 1, \ldots, k$, the induced subgraph $G[V_i]$ is connected and contains t_i. The *minor induced* by a terminal partition $\{V_1, \ldots, V_k\}$ is the minor M obtained by contracting each set V_i into a single vertex called (abusing notation) t_i. Notice that M has an edge between t_i and t_j iff there are vertices $v_i \in V_i$ and $v_j \in V_j$ such that $\{v_i, v_j\} \in E$. The weight of this edge (if exists) is simply $d_G(t_i, t_j)$, which represents the shortest-path in G. It is easily verified that by the triangle inequality, for every pair of (not necessarily adjacent) terminals t_i, t_j, we have $d_M(t_i, t_j) \geq d_G(t_i, t_j)$. The *distortion* of the induced minor is $\max_{i \neq j} \frac{d_M(t_i,t_j)}{d_G(t_i,t_j)}$. It was proved in [10] that the `Relaxed-Voronoi` algorithm always returns a terminal partition.

▶ **Lemma 2.1** (Lemma 2 in [10]). *The sets $V_1, \ldots, V_k$ constructed by Algorithm 2 constitute a terminal partition.*

We say that a metric (X, d) has *doubling dimension* ddim if every ball of radius $r > 0$ can be covered by at most 2^{ddim} balls of radius $r/2$. We will use the following *packing property* of doubling spaces [14]: Consider a set N such that for every $x \neq y \in N$ it holds that $d(x,y) \geq \delta$. Then every ball of radius $\Delta \geq \delta$ contains at most $\left(\frac{4\Delta}{\delta}\right)^{O(\text{ddim})} = 2^{O\left(\text{ddim} \cdot \log \frac{\Delta}{\delta}\right)}$ points from N.

We denote by $\text{EXP}(\lambda)$ the *exponential distribution* with mean $\lambda > 0$, which has density function $f(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}}$ for $x \geq 0$. This distribution is *memoryless*: if $X \sim \text{EXP}(\lambda)$, then for all $a, b \geq 0$ we have $\Pr[X \geq a + b \mid X \geq a] = \Pr[X \geq b]$. In other words, conditioned on $X \geq a$, it holds that $X \sim a + \text{EXP}(\lambda)$.

3 SPR on trees

In this section we analyze the `Relaxed-Voronoi` algorithm (Algorithm 2) on trees.

▶ **Theorem 3.1.** *Let T be a tree and r be an arbitrary vertex. Let π be an ordering of the terminals according to an increasing distance from r. Then the tree T_K returned by the `Relaxed-Voronoi` algorithm on input $(T, K, \pi, \{3, 3, \ldots, 3\})$ has distortion at most 8. Moreover, the algorithm can be implemented in linear time.*

In Section 3.1 we bound the distortion produced by our algorithm, and in Section 6 we describe its linear time implementation. See Figure 2 for an example execution of the algorithm on a complete unweighted binary tree (the lower bound example used by [4]).

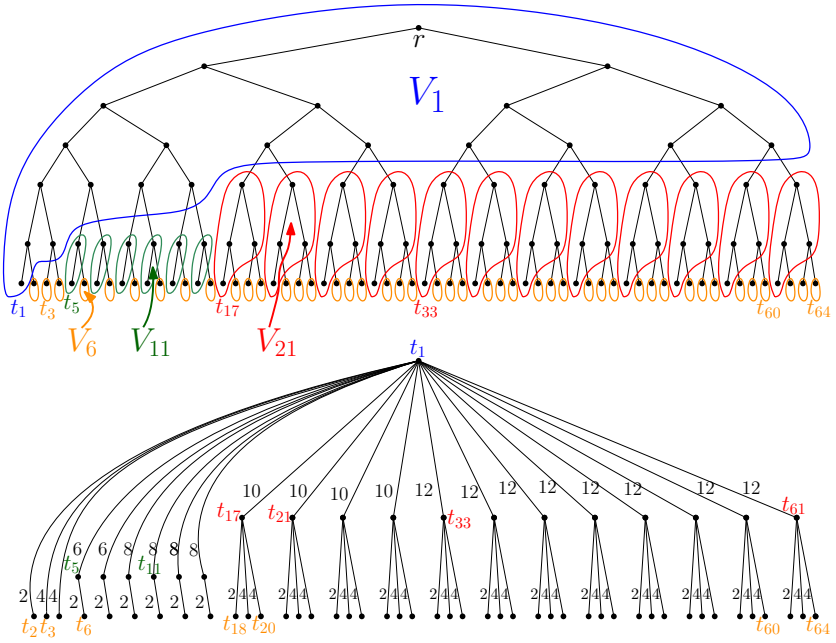


Figure 2 An example execution of the `Relaxed-Voronoi` algorithm. The top graph is the input, a complete binary tree of height 6 with all the leaves as terminals. Choosing r to be the root of the tree, the terminal ordering π can be arbitrary, and we choose a left-to-right ordering. The resulting clusters are shown using colored curves. The bottom graph shows the minor induced by the terminal partition above, representing every cluster by its top-most vertex (rather than the terminal). The distortion is $\frac{d_{T_k}(t_{64},t_{60})}{d_T(t_{64},t_{60})} = \frac{32}{6} = 5\frac{2}{3}$.

3.1 Distortion Analysis

To better understand the final distortion bound 8, we analyze the `Relaxed-Voronoi` algorithm for a general $R > 1$, and we optimize it only at the very end, setting $R = 3$ to obtain distortion 8.

Denote by T_K the tree minor returned by the algorithm, and call the vertex r (used to determine π) the root. Let t_1 be the first terminal w.r.t π, and let V_1 be the cluster that the `Relaxed-Voronoi` algorithm constructs for t_1. This terminal t_1 is the closest terminal to the root r, and actually also to every vertex on the shortest path from t_1 to r. Therefore r joins the cluster V_1. Let $C_1, \ldots, C_s$ be the connected components of the remaining graph $G \setminus V_1 = G[V \setminus V_1]$, and let $K_i = C_i \cap K$ be the subset of terminals in component C_i. We claim that for every vertex $v \in C_i$ (for every i), its closest terminal t_v satisfies $t_v \in K_i$. Indeed, assume towards contradiction that some vertex u on the path between v to $t_v \in K_i$ joined V_1. Consider then an arbitrary vertex u' on the path from v to u, and note that t_v is also the closest terminal to both u, u'. By the triangle inequality, $d_T(t_1, u') \leq d_T(t_1, u) + d_T(u, u') \leq R \cdot \big(d_T(t_v, u) + d_T(u, u')\big) = R \cdot d_T(t_v, u')$. This implies that every vertex on the path from u to v will join V_1 (recall we assumed u joins V_1, and the algorithm iteratively adds neighbor of vertices already in V_1), in contradiction with $v \in C_i$. See Figure 3 for illustration.

By construction, there is only a single edge e_i that connects C_i and V_1, and denote its two endpoints by $r_i \in C_i$ and $s_i \in V_1$. Let $\tilde{t}_i \in K$ be the closest terminal to r_i, thus $\tilde{t}_i \in C_i$. Observe that r_i is the closest vertex to r among all vertices in C_i, and in particular every path from a terminal $t \in C_i$ to r goes through r_i. Let π_i be an ordering of the terminals in K_i according to increasing distance from r_i. Note that π_i is just the order π restricted to K_i.

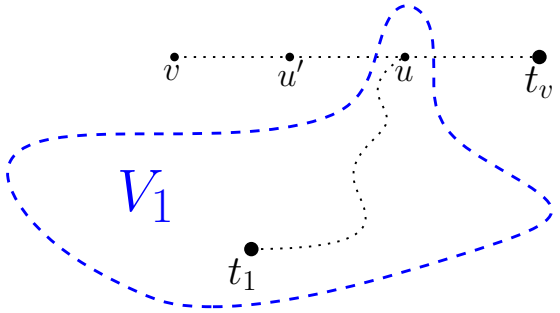


Figure 3 Illustrating the argument that for every $v \in C_i$, also its closest terminal $t_v \in C_i$. Assuming some u on the path between them joined V_1, we conclude the entire path from u to v joins V_1.

Since all the clusters created by the `Relaxed-Voronoi` algorithm are connected, no vertex in C_i can join a cluster associated with a terminal outside K_i. In particular, for every $v \in C_i$ the distance $D(v)$ to the closest terminal in the restricted tree $G[C_i]$ remains the same (as $t_v \in C_i$). Therefore, if we execute the `Relaxed-Voronoi` algorithm on C_i with terminal set K_i and order π_i, the partition of C_i to clusters will be identical to the partition of C_i induced by the original algorithm (on T with the order π). Accordingly, if we combine all the clusters created by such executions with V_1, we get the same terminal partition as produced by the `Relaxed-Voronoi` algorithm on the original graph.

Next, we argue by induction on the number of terminals that for every terminal t, $d_{T_K}(t_1, t) \leq \frac{R+1}{R-1} \cdot d_T(t_1, t)$. In a tree with a single or two terminals this claim is trivial. We now prove the induction step. Let t_i be some terminal which belongs to the connected component C_i (in $T \setminus V_1$). By applying the induction hypothesis to the tree C_i with order π_i, it holds that $d_{T_k}(t_i, \tilde{t}_i) \leq \frac{R+1}{R-1} \cdot d_T(t_i, \tilde{t}_i)$ as $\tilde{t}_i$ is the first terminal in the order π_i. Note that r_i will necessarily join the cluster of $\tilde{t}_i$, therefore the edge $(e_i = \{s_i, r_i\})$ crosses the clusters of t_1 and $\tilde{t}_i$, which implies that there is an edge between t_1 to $\tilde{t}_i$ in T_K. See Figure 4 for illustration.

As r_i has a neighbor in V_1 but did not join V_1, necessarily $d_T(t_1, r_i) > R \cdot D(r_i) = R \cdot d_T(r_i, \tilde{t}_i)$. We conclude,

$$\begin{aligned}
d_{T_K}(t_1, t_i) &\leq d_{T_K}(t_1, \tilde{t}_i) + d_{T_K}(\tilde{t}_i, t_i) \\
&\leq d_T(t_1, \tilde{t}_i) + \frac{R+1}{R-1} \cdot d_T(\tilde{t}_i, t_i) \\
&\leq d_T(t_1, r_i) + D(r_i) + \frac{R+1}{R-1} \cdot \big(D(r_i) + d_T(r_i, t_i)\big) \\
&< d_T(t_1, r_i) + \big(1 + \frac{R+1}{R-1}\big) \cdot \frac{d_T(t_1, r_i)}{R} + \frac{R+1}{R-1} \cdot d_T(r_i, t_i) \\
&= \frac{R+1}{R-1} \cdot \big(d_T(t_1, r_i) + d_T(r_i, t_i)\big) = \frac{R+1}{R-1} \cdot d_T(t_1, t_i) \ .
\end{aligned}$$

Finally, we show by induction that for every pair of terminals $t_i, t_j \in K \setminus \{t_1\}$, $d_{T_k}(t_i, t_j) < \frac{(R+1)^2}{R-1} \cdot d_T(t_i, t_j)$. If t_i, t_j belong to the same connected component of $T \setminus V_1$ then the argument follows by the induction hypothesis. Otherwise, $t_i \in C_i$ and $t_j \in C_j$ for $i \neq j$. Recall that there is a single edge $e_i = \{s_i, r_i\}$ from C_i to V_1. Clearly, the unique path in T from t_i to t_j goes through V_1 and in particular through s_i and s_j (note that it is possible that $s_i = s_j$). Therefore, $d_T(t_i, t_j) \geq d_T(t_i, s_i) + d_T(s_j, t_j)$. As $s_i \in V_1$, it holds that $d_T(t_1, s_i) \leq R \cdot D(s_i) \leq R \cdot d_T(t_i, s_i)$. Therefore,

$$d_T(t_1, t_i) \leq d_T(t_1, s_i) + d_T(s_i, t_i) \leq (R+1) \cdot d_T(s_i, t_i) \,. \tag{1}$$

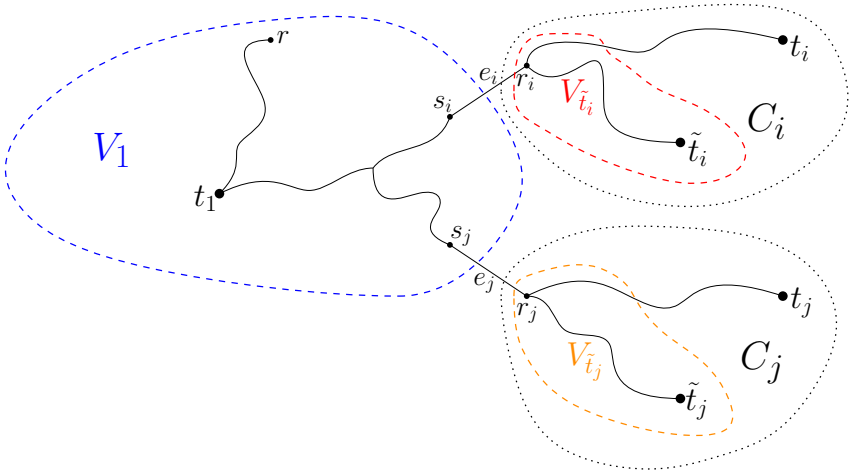


Figure 4 Illustrating the bound on $d_{T_k}(t_i, t_j)$. Initially $d_{T_k}(t_1, t_i)$ is bounded. Notice that $\tilde{t}_i$ is the closest terminal to r_i. Using the induction hypothesis we have that $d_{T_k}(\tilde{t}_i, t_i) \leq \frac{R+1}{R-1} \cdot d_T(\tilde{t}_i, t_i)$. As $\{t_1, \tilde{t}_i\}$ is an edge in T_k, the bound follows.
Next, the bound on $d_{T_k}(t_i, t_j)$. Notice that $d_T(t_i, t_j) \geq d_T(t_i, s_i) + d_T(t_j, s_j)$. $d_{T_k}(t_i, t_j)$ is upper bounded by going through t_1, using the assertion above.

Similarly $d_T(t_1, t_j) \leq (R+1) \cdot d_T(s_j, t_j)$. Using our claim above about t_1, we conclude (see Figure 4 for illustration)

$$\begin{aligned} d_{T_k}(t_i, t_j) &\leq d_{T_k}(t_i, t_1) + d_{T_k}(t_1, t_j) \\ &\leq \frac{R+1}{R-1} \cdot (d_T(t_i, t_1) + d_T(t_1, t_j)) \\ &\overset{(1)}{\leq} \frac{(R+1)^2}{R-1} \cdot (d_T(t_i, s_i) + d_T(s_j, t_j)) \\ &\leq \frac{(R+1)^2}{R-1} \cdot d_T(t_i, t_j). \end{aligned}$$

The expression $\frac{(R+1)^2}{R-1}$ is minimized by choosing $R = 3$, which proves the upper bound 8.

4 M0E for Doubling Metrics

In this section we analyze the `Relaxed-Voronoi` algorithm (Algorithm 1) for the M0E problem, in the case where the metric spaces restricted on the terminals has doubling dimension ddim. Given a metric space (X, d), Gonzalez's order [12] is defined as follows. x_1 is an arbitrary point, x_2 is the farthest point from x_1, and in general x_i is the farthest point from $\{x_1, \ldots, x_{i-1}\}$. In other words, x_i is the point maximizing $d(x_i, \{x_1, \ldots, x_{i-1}\})$.

▶ **Theorem 4.1.** *Let (X, d) be a metric space with a set of terminals $K \subseteq X$ such that the metric space restricted to the terminals has doubling dimension* ddim*. Let π be Gonzalez's order. Let $R_j = 2 \cdot e^{Z_j}$, where $Z_1, \ldots, Z_k$ are i.i.d. variables sampled according to the distribution* $\mathrm{EXP}(c \cdot \mathrm{ddim})$ *for large enough constant c. Then the expected distortion returned by the `Relaxed-Voronoi` algorithm for the M0E problem is $O(\mathrm{ddim})$.*

Proof. Consider a point $x \in X$, and let i_x be the minimal index such that $d(t_x, t_{i_x}) \leq D(x)$. Set $K_x = \{t_1, \ldots, t_{i_x}\}$. As $R_{i_x} \geq 2$, if x is unassigned until the i_x round, then $f(x) = t_{i_x}$.

Therefore, $f(x) \in K_x$. For every $t, t' \in K_x \setminus \{t_x\}$, $d(t,t') \geq D(x)$. Using the packing property, for $i \geq 1$, $\left|B\left(v, 2^i \cdot D(v)\right) \cap K_x\right| \leq \left|B\left(t_x, \left(2^i+1\right) \cdot D(v)\right) \cap K_x\right| = 2^{O(i \cdot \mathrm{ddim})}$.

▶ **Lemma 4.2.** *For every $x \in X$, $\mathbb{E}\left[d(x, f(x))\right] = O\left(1\right) \cdot D(x)$.*

Proof. For $i \geq 3$, let $K_i \subseteq K_x$ be the set of terminals at distance $[2^{i-1}, 2^i) \cdot D(v)$ from x. In order for the terminal $t_j \in K_i$ to cover x, it must be that $R_j \geq 2^{i-1}$, where a terminal t covers a point z if $f(z) = t$. This happens with probability at most

$$\Pr\left[R_j \geq 2^{i-1}\right] = \Pr\left[Z_j \geq (i-2) \cdot \ln 2\right] = e^{-c \cdot \mathrm{ddim} \cdot (i-2) \cdot \ln 2} \leq e^{-\frac{c}{5} \cdot \mathrm{ddim} \cdot i} .$$

By the union bound, the probability that some terminal from K_i covers x is bounded by $|K_i| \cdot e^{-\frac{c}{5} \cdot \mathrm{ddim} \cdot i}$. We conclude that for large enough constant c,

$$\begin{aligned}\mathbb{E}\left[d(x, f(x))\right] &\leq 2^2 \cdot D(x) + \sum_{i=3}^{\infty} \Pr\left[f(x) \in K_i\right] \cdot 2^i \cdot D(x) \\ &= 4 \cdot D(x) + D(x) \cdot \sum_{i=3}^{\infty} 2^{O(i \cdot \mathrm{ddim})} \cdot e^{-\frac{c}{5} \cdot \mathrm{ddim} \cdot i} \cdot 2^i = O(D(x)) .\end{aligned}$$

◀

Consider a pair of points $x, y \in X$ such that $d(x,y) = \epsilon \cdot \min\{D(x), D(y)\}$. If $\epsilon = \Omega(1)$, assume w.l.o.g that $D(x) \leq D(y)$, then $D(y) \leq D(x) + d(x,y) = O(1) \cdot d(x,y)$. Using Theorem 4.2 we conclude

$$\begin{aligned}\mathbb{E}\left[d(f(x), f(y))\right] &\leq \mathbb{E}\left[d(f(x), x)\right] + d(x,y) + \mathbb{E}\left[d(y, f(y))\right] \\ &= O\left(D(x) + D(y)\right) + d(x,y) = O\left(1\right) \cdot d(x,y) .\end{aligned} \tag{2}$$

Thus from now on we can assume that ϵ is upper bounded by small enough constant, and we also drop the assumption that $D(x) \leq D(y)$. We say that a terminal t_j *settles* the pair $\{x,y\}$ if it is the first terminal to cover at least one point among $\{x,y\}$, and denote this event by $\mathcal{S}_j$. We say that t_j *cuts* $\{x,y\}$ if t_j settles $\{x,y\}$ but covers only one of x, y, and denote this event by $\mathcal{C}_j$. Set $R_x = \frac{d(x,t_j)}{D(x)}$, $R_y = \frac{d(y,t_j)}{D(y)}$. Assuming w.l.o.g that $R_x \leq R_y$, we get

$$R_y = \frac{d(t_j, y)}{D(y)} \leq \frac{d(t_j, x) + d(v,u)}{D(x) - d(v,u)} \leq \frac{R_x \cdot D(x) + \epsilon \cdot D(x)}{D(x) - \epsilon \cdot D(x)} \leq \frac{1+\epsilon}{1-\epsilon} \cdot R_x < (1+3\epsilon) \cdot R_x . \tag{3}$$

Assuming that t_j settles $\{x,y\}$, using the memoryless property we can bound the probability that t_j cuts $\{x,y\}$.

$$\begin{aligned}\Pr\left[\mathcal{C}_j \mid \mathcal{S}_j\right] &= \Pr\left[R_j < R_y \mid R_j \geq R_x\right] \overset{(3)}{<} \Pr\left[2 \cdot e^{Z_j} < R_x \cdot (1+3\epsilon) \mid 2 \cdot e^{Z_j} < R_x\right] \\ &= \Pr\left[Z_j < \ln(1+3\epsilon)\right] < \Pr\left[Z_j < 3\epsilon\right] = 1 - e^{-3\epsilon \cdot c \cdot \mathrm{ddim}} \leq 6\epsilon \cdot c \cdot \mathrm{ddim} .\end{aligned} \tag{4}$$

Suppose that t_j indeed cuts $\{x,y\}$. Following the same arguments as Theorem 4.2, the expected distance between y to $f(y)$ still will be $O(D(y)) = O(\frac{1}{\epsilon}) \cdot d(x,y)$. Thus,

$$\begin{aligned}\mathbb{E}\left[d(f(x), f(y)) \mid \mathcal{C}_j\right] &\leq d(t_j, x) + d(x,y) + \mathbb{E}\left[d(y, f(y)) \mid \mathcal{C}_j\right] \\ &= d(t_j, \{x,y\}) + O\left(\tfrac{1}{\epsilon}\right) \cdot d(x,y) .\end{aligned} \tag{5}$$

For $i \geq 1$, denote by $\tilde{K}_i \subseteq K_x \cup K_y$ the set of terminals at distance $[2^{i-1}, 2^i) \cdot \min\{D(x), D(y)\}$ from $\{x,y\}$. By packing arguments, $|\tilde{K}_i| = 2^{O(i \cdot \mathrm{ddim})}$. By similar arguments to Theorem 4.2,

for $i \geq 3$, the probability that $\{x, y\}$ is settled by a terminal from $\tilde{K}_i$ is bounded by $2^{-\Omega(i \cdot \mathrm{ddim})}$. We conclude,

$$\begin{aligned}
&\mathbb{E}\left[d(f(x), f(y))\right] = \\
&= \sum_j \Pr\left[\mathcal{S}_j\right] \cdot \Pr\left[\mathcal{C}_j \mid \mathcal{S}_j\right] \cdot \mathbb{E}\left[d(f(x), f(y)) \mid \mathcal{C}_j\right] \\
&\overset{(4,5)}{\leq} 6\epsilon \cdot c \cdot \mathrm{ddim} \cdot \sum_j \Pr\left[\mathcal{S}_j\right] \cdot \left(d(t_j, \{x, y\}) + O\left(\tfrac{1}{\epsilon}\right) \cdot d(x, y)\right) \\
&= O\left(\mathrm{ddim}\right) \cdot d(x, y) + O\left(\epsilon \cdot \mathrm{ddim}\right) \cdot \left(4 + \sum_{i \geq 3} 2^{-\Omega(i \cdot \mathrm{ddim})} \cdot 2^i\right) \cdot \min\{D(x), D(y)\} \\
&= O\left(\mathrm{ddim}\right) \cdot d(x, y).
\end{aligned}$$

◀

5 Connected M0E

In this section we apply the (Graphic) `Relaxed-Voronoi` algorithm (Algorithm 2) to the connected-M0E problem.

▶ **Theorem 5.1.** *Let $G = (V, E, w)$ be a weighted graph and $K \subseteq X$ a set of terminals of size k. Let π be arbitrary, and let $R_j = e^{Z_j}$, where $Z_1, \ldots, Z_k$ are i.i.d. variables sampled according to distribution $\mathrm{EXP}(c \cdot \ln k)$ for large enough constant c. Then the expected distortion returned by the `Relaxed-Voronoi` algorithm for the connected M0E problem is $O(\log k)$.*

By the triangle inequality, it is enough to prove that for every edge $\{u, v\} \in E$ (where $d_G(v, u) = w(v, u)$) it holds that $\mathbb{E}_{f \sim \mathcal{D}}\left[d(f(u), f(v)\right] \leq \alpha \cdot d_G(v, u)$. The proof itself follows almost the same lines as the proof of Theorem 4.1. With high probability, $R_j \leq 2$ for every terminal t_j. Therefore, for every vertex v, $d(v, f(v)) \leq 2 \cdot D(v)$. Once a vertex v joins the cluster V_j, the probability that its unclustered neighbor vertex u, at distance $\epsilon \cdot D(v)$, does not join V_j is bounded by $O(\epsilon \cdot \log k)$ (similarly to Equation (4)). Using these two facts we can bound the expected distortion by $O(\log k)$. We skip the exact details.

6 Linear-Time Implementation

Our algorithm often uses $D(v)$. The next lemma state that this values can be computed efficiently.

▶ **Lemma 6.1.** *There is a linear-time algorithm, that given as an input a weighted graph $G = (V, E, w)$ and $K \subseteq V$ a set of terminals, outputs for every vertex $v \in V$ its distance from K.*

Proof. We describe the algorithm. We root the tree in some arbitrary vertex $r \in V$. Thus each vertex (other then r) has a parent vertex. Our algorithm has two phases. In the first phase we sweep the tree upwards from the leafs to the root. For a vertex v, denote by $d(v)$ the distance from v to it's closest terminal among its descendants (∞ if it has no descendant terminal). The goal of the first phase is for each vertex to learn $d(v)$, and this is done in a dynamic programing fashion according to the order induced by the tree. At the beginning each leaf v know $d(v)$ (0 if terminal and ∞ otherwise). Then, iteratively each internal vertex v with children $\{v_1, \ldots, v_s\}$ computes $d(v) = \min_i \{d(v_i) + d(v_i, v)\}$ or $d(v) = 0$ if v itself is a terminal. It is straightforward by induction that by the end of the first phase each vertex

has the right value of $d(v)$. Moreover, for the root vertex r, $D(r) = d(r)$ (as all the terminals are the descendants of r).

In the second phase we sweep the tree downwards from the root to the leaves. In the first step, r informs all its children the value $D(r)$. Then, iteratively, each vertex v with parent v' computes $D(v) = \min\{d(v), D(v') + d(v', v)\}$. Again, by induction this is indeed the right value (as every path ending in v which starts at a non-descendant of v must go through v'). By the end of the second phase each vertex knows the correct value of $D(v)$. The linear time implementation follows as we traversed each edge exactly twice. ◀

The execution of the `Relaxed-Voronoi` algorithm starts by computing the $D(v)$ values in linear time according to Theorem 6.1. Next, in order to determine the permutation π, we choose an arbitrary vertex r and run Dijkstra from it. In a tree, one can run the classic Dijkstra algorithm (as in [11]) using a queue instead of a heap. As there is a unique path from r to any other vertex, the algorithm still works properly. Next, we cluster the vertices according to the permutation π. The set N from the `Create-Cluster` procedure can be implemented as a simple queue. As there is a unique path between every pair of vertices, once a vertex v joins N, we can update $d(v, t_j)$ to its correct value. Moreover, there is no reason to maintain U. As in all the executions of the `Create-Cluster` procedure for all terminals, each edge is traversed exactly once, the total linear time follows.

References

1. Aaron Archer, Jittat Fakcharoenphol, Chris Harrelson, Robert Krauthgamer, Kunal Talwar, and Éva Tardos. Approximate classification via earthmover metrics. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 1079–1087, 2004. URL: `http://dl.acm.org/citation.cfm?id=982792.982952`.
2. A. Basu and A. Gupta. Steiner Point Removal in Graph Metrics. Unpublished Manuscript, available from `http://www.math.ucdavis.edu/~abasu/papers/SPR.pdf`, 2008.
3. Gruia Călinescu, Howard J. Karloff, and Yuval Rabani. Approximation Algorithms for the 0-Extension Problem. *SIAM J. Comput.*, 34(2):358–372, 2004. `doi:10.1137/S0097539701395978`.
4. T.-H. Chan, Donglin Xia, Goran Konjevod, and Andrea Richa. A Tight Lower Bound for the Steiner Point Removal Problem on Trees. In *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems, and 10th International Conference on Randomization and Computation*, APPROX'06/RANDOM'06, pages 70–81, 2006. `doi:10.1007/11830924_9`.
5. Yun Kuen Cheung. Steiner Point Removal - Distant Terminals Don't (Really) Bother. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1353–1360, 2018. `doi:10.1137/1.9781611975031.89`.
6. Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The Complexity of Multiway Cuts (Extended Abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC 1992*, pages 241–251, 1992. `doi:10.1145/129712.129736`.
7. Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex Sparsifiers: New Results from Old Techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014. `doi:10.1137/130908440`.
8. Jittat Fakcharoenphol, Chris Harrelson, Satish Rao, and Kunal Talwar. An improved approximation algorithm for the 0-extension problem. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 257–265, 2003. URL: `http://dl.acm.org/citation.cfm?id=644108.644153`.

9 Arnold Filtser. Steiner Point Removal with Distortion $O(\log k)$. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1361–1373, 2018. doi:10.1137/1.9781611975031.90.

10 Arnold Filtser. Steiner Point Removal with distortion $O(\log k)$, using the Noisy-Voronoi algorithm. *CoRR*, abs/1808.02800, 2018. arXiv:1808.02800.

11 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

12 Teofilo F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

13 Anupam Gupta. Steiner Points in Tree Metrics Don'T (Really) Help. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 220–227, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365448.

14 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded Geometries, Fractals, and Low-Distortion Embeddings. In *44th Symposium on Foundations of Computer Science (FOCS 2003)*, pages 534–543, 2003. doi:10.1109/SFCS.2003.1238226.

15 Anupam Gupta and Kunal Talwar. Random Rates for 0-Extension and Low-Diameter Decompositions. *CoRR*, abs/1307.5582, 2013. arXiv:1307.5582.

16 Lior Kamma, Robert Krauthgamer, and Huy L. Nguyen. Cutting Corners Cheaply, or How to Remove Steiner Points. *SIAM J. Comput.*, 44(4):975–995, 2015. doi:10.1137/140951382.

17 Alexander V. Karzanov. Minimum 0-Extensions of Graph Metrics. *Eur. J. Comb.*, 19(1):71–101, 1998. doi:10.1006/eujc.1997.0154.

18 Jon M. Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov random fields. *J. ACM*, 49(5):616–639, 2002. doi:10.1145/585265.585268.

19 James R. Lee and Assaf Naor. Metric decomposition, smooth measures, and clustering. Unpublished Manuscript, available from https://www.math.nyu.edu/~naor/homepage%20files/cluster.pdf, 2003.

20 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *25th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 196–203, 2013. doi:10.1145/2486159.2486180.

Towards a Unified Theory of Sparsification for Matching Problems

Sepehr Assadi[1]
Department of Computer and Information Science, University of Pennsylvania
Philadelphia, PA, US
sassadi@cis.upenn.edu

Aaron Bernstein
Department of Computer Science, Rutgers University
Piscataway, NJ, US
bernstei@gmail.com

Abstract

In this paper, we present a construction of a "matching sparsifier", that is, a sparse subgraph of the given graph that preserves large matchings approximately and is robust to modifications of the graph. We use this matching sparsifier to obtain several new algorithmic results for the maximum matching problem:

- An almost (3/2)-approximation one-way communication protocol for the maximum matching problem, significantly simplifying the (3/2)-approximation protocol of Goel, Kapralov, and Khanna (SODA 2012) and extending it from bipartite graphs to general graphs.
- An almost (3/2)-approximation algorithm for the stochastic matching problem, improving upon and significantly simplifying the previous 1.999-approximation algorithm of Assadi, Khanna, and Li (EC 2017).
- An almost (3/2)-approximation algorithm for the fault-tolerant matching problem, which, to our knowledge, is the first non-trivial algorithm for this problem.

Our matching sparsifier is obtained by proving new properties of the edge-degree constrained subgraph (EDCS) of Bernstein and Stein (ICALP 2015; SODA 2016) – designed in the context of maintaining matchings in dynamic graphs – that identifies EDCS as an excellent choice for a matching sparsifier. This leads to surprisingly simple and non-technical proofs of the above results in a unified way. Along the way, we also provide a much simpler proof of the fact that an EDCS is guaranteed to contain a large matching, which may be of independent interest.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners, Theory of computation → Graph algorithms analysis

Keywords and phrases Maximum matching, matching sparsifiers, one-way communication complexity, stochastic matching, fault-tolerant matching

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.11

Acknowledgements Sepehr Assadi is grateful to his advisor Sanjeev Khanna for many helpful discussions, and to Soheil Behnezhad for sharing a write-up of [9].

[1] Supported in part by the National Science Foundation grant CCF-1617851.

1 Introduction

A common tool for dealing with massive graphs is sparsification. Roughly speaking, a sparsifier of a graph G is a subgraph H that (approximately) preserves certain properties of G while having a smaller number of edges. Such sparsifiers have been studied in great detail for various properties: for example, a spanner [6, 29] or a distance preserver [18, 20] preserves pairwise distances, a cut sparsifier [26, 11, 22] preserves cut information, and a spectral sparsifier [32, 8] preserves spectral properties of the graph. An additional property that we often require of a graph sparsifier is *robustness*: it should continue to be a good sparsifier even as the graph changes. Some sparsifiers are robust by nature (e.g cut sparsifiers), but others (e.g spanners) are not, and for this reason there is an extensive literature on designing sparsifiers that can provide additional robustness guarantees.

In this paper, we study the problem of designing robust sparsifiers for the prominent problem of *maximum matching*. Multiple notions of sparsification for the matching problem have already been identified in the literature. One example is a subgraph that preserves the largest matching inside any given subset of vertices in G approximately. This notion is also known as a matching cover or a matching skeleton [23, 27] in the literature and is closely related to the communication and streaming complexity of the matching problem. Another example of a sparsifier is a subgraph that can preserve the largest matching on random subsets of edges of G, a notion closely related to the stochastic matching problem [15, 5]. An example of a robust sparsifier for matching is a fault-tolerant subgraph, namely a subgraph G that continue to preserve large matchings in G even after a fraction of the edges is deleted by an adversary. As far as we know, the fault-tolerant matching problem has not previously been studied, but it is a natural model to consider as it has received lots of attention in the context of spanners and distance preservers (see e.g. [19, 28, 7, 17, 16]).

Our first contribution is a subgraph H that we show is a robust matching sparsifier in *all* of the senses above. Our result is thus the first to unify these notions of sparsification for the maximum matching problem. In addition to unifying, our construction yields improved results for each individual notion of sparsification and the corresponding problems, namely, the one-way communication complexity of matching, stochastic matching, and fault-tolerant matching problems. Interestingly, our unified approach allows us to also provide much simpler proofs than all previously existing work for these problems. The subgraph we use as our sparsifier comes from a pair of papers by Bernstein and Stein on dynamic matching [13, 14] – they refer to this subgraph as an edge-degree constrained subgraph (EDCS for short). The EDCS was also very recently used in [2] to design sublinear algorithms for matching across several different models for massive graphs. Our applications of the EDCS in the current paper, as well as the new properties we prove for the EDCS, are quite different from those in [13, 14, 2]. Our first contribution thus takes an existing subgraph, and then provides the first proofs that it satisfies the three notions of sparsification described above.

Our second contribution is a much simpler (and even slightly improved) proof of the main property of an EDCS in previous work proved in [13, 14], namely that an EDCS contains a large matching of the original graph. Our new proof significantly simplifies the analysis of [14] and allows for simple and self-contained proofs of the results in this paper.

Definition of the EDCS. Before stating our results, we give a definition of the EDCS from [13, 14], as this is the subgraph we use for all of our results (see Section 2 for more details).

▶ **Definition 1** ([13]). For any graph $G(V, E)$ and integers $\beta \geq \beta^- \geq 0$, an *edge-degree constrained subgraph (EDCS)* (G, β, β^-) is a subgraph $H := (V, E_H)$ of G with the following two properties:

(P1) For any edge $(u, v) \in E_H$: $\deg_H(u) + \deg_H(v) \leq \beta$.

(P2) For any edge $(u, v) \in E \setminus E_H$: $\deg_H(u) + \deg_H(v) \geq \beta^-$.

It is not hard to show that an EDCS of a graph G always exists for any parameters $\beta > \beta^-$ and that it is sparse, i.e., only has $O(n\beta)$ edges. A key property of EDCS proven previously [13, 14] (and simplified in our paper) is that for any reasonable setting of the parameters (e.g. β^- being sufficiently close to β), any EDCS H of G contains an (almost) $3/2$ approximate matching of G.

1.1 Our Results and Techniques

We now give detailed definitions of the notions of sparsification and the corresponding problems addressed in this paper, as well as our results for each one. Our second contribution – a significantly simpler proof that an EDCS contains an almost $(3/2)$-approximate matching – is left for Section 3.

One-Way Communication Complexity of Matching. Consider the following two-player communication problem: Alice is given a graph $G_A(V, E_A)$ and Bob holds a graph $G_B(V, E_B)$. The goal for Alice is to send a single message to Bob such that Bob outputs an approximate maximum matching in $E_A \cup E_B$. What is the minimum length of the message, i.e., the one-way communication complexity, for achieving a certain fixed approximation ratio on all graphs? One can show that the message communicated by Alice to Bob is indeed a matching skeleton, namely a data structure (but not necessarily a subgraph), that allows Bob to find a large matching in a given subset of vertices in Alice's input (see [23] for more details).

This problem was first studied by Goel, Kapralov, and Khanna [23] (see also the subsequent paper of Kapralov [25]), owing to its close connection to one-pass streaming algorithms for matching. Goel *et al.* [23] designed an algorithm that achieves a $(3/2)$-approximation in bipartite graphs using only $O(n)$ communication and proved that any better than $(3/2)$-approximation protocol requires $n^{1+\Omega(\frac{1}{\log \log n})}$ communication even on bipartite graphs (see, e.g. [23, 4] for further details on this lower bound). A follow-up work by Lee and Singla [27] further generalized the algorithm of [23] to general graphs, albeit with a slightly worse approximation ratio of $5/3$ (compared to $3/2$ of [23]).

We extends the results in [23] to general graphs with almost no loss in approximation.

▶ Result 1. For any constant $\varepsilon > 0$, the protocol where Alice computes an EDCS of her graph with $\beta = O(1)$ and $\beta^- = \beta - 1$ and sends it to Bob is a $(3/2 + \varepsilon)$-approximation one-way communication protocol for the maximum matching problem with uses $O(n)$ communication.

We remark that both the previous algorithm of [23] as well as its extension in [27] are quite involved and rely on a fairly complicated graph decomposition as well as an intricate primal-dual analysis. As such, we believe that the main contribution in Result 1 is in fact in providing a simple and self-contained proof of this result.

Stochastic Matching. In the stochastic matching problem, we are given a graph $G(V, E)$ and a probability parameter $p \in (0, 1)$. A realization of G is a subgraph $G_p(V, E_p)$ obtained by picking each edge in G independently with probability p to include in E_p. The goal in

this problem is to find a subgraph H of G with max-degree bounded by a function of p (independent of number of vertices), such that the size of maximum matching in realizations of H is close to size of maximum matching in realizations of G. It is immediate to see that H in this problem is simply a sparsifier of G which preserves large matchings on random subsets of edges.

This problem was first introduced by Blum *et al.* [15] primarily to model the kidney exchange setting and has since been studied extensively in the literature [3, 5, 10, 34]. Early algorithms for this problem in [15, 3] (and the later ones for the weighted variant of the problem [10, 34]) all had approximation ratio at least 2, naturally raising the question that whether 2 is the best approximation ratio achievable for this problem. Assadi, Khanna, and Li [5] ruled out this perplexing possibility by obtaining a slightly better than 2-approximation algorithm for this problem, namely an algorithm with approximation ratio close to 1.999 (which improves to 1.923 for small p).

We prove that an EDCS results in a significantly improved algorithm for this problem.

▶ Result 2. For any constant $\varepsilon > 0$, an $EDCS$ of G with $\beta = O(\frac{\log(1/p)}{p})$ and $\beta^- = \beta - 1$ achieves a $(3/2 + \varepsilon)$-approximation algorithm for the stochastic matching problem with a subgraph of maximum degree $O(\frac{\log(1/p)}{p})$.

We remark that our bound on the maximum degree in Result 2 is optimal (up to an $O(\log(1/p))$ factor) for any constant-factor approximation algorithm (see [5]). In addition to significantly improving upon the previous best algorithm of [5], our Result 2 is much simpler than that of [5], in terms of the both the algorithm and (especially) the analysis.

Remark. Independently and concurrently, Behnezhad *et al.* [9] also presented an algorithm for stochastic matching with a subgraph of max-degree $O(\frac{\log(1/p)}{p})$ that achieves an approximation of almost $(4\sqrt{2} - 5)$ (≈ 0.6568 compared to 0.6666 in Result 2). They also provided an algorithm with approximation ratio strictly better than half for weighted stochastic matching (our result does not work for weighted graphs). In terms of techniques, our paper and [9] are entirely disjoint.

Fault-Tolerant Matching. Let $f \geq 0$ be an integer, $G(V, E)$ be a graph, and H be any subgraph of G. We say that H is an α-approximation f-tolerant subgraph of G iff for any subset $F \subseteq E$ of size $\leq f$, the maximum matching in $H \setminus F$ is an α-approximation to maximum matching in $G \setminus F$ – that is, H is a robust sparsifier of G. This definition is a natural analogy of other fault-tolerant subgraphs, such as fault-tolerant spanners and fault-tolerant distance preservers (see, e.g. [19, 28, 7, 17, 16]), to the maximum matching problem. Despite being such fundamental objects, quite surprisingly fault-tolerant subgraphs have not previously been studied for the matching problem.

We complete our discussion of applications of EDCS as a robust sparsifier by showing that it achieves an optimal size fault-tolerant subgraph for the matching problem.

▶ Result 3. For any constant $\varepsilon > 0$ and any $f \geq 0$, there exists a $(3/2+\varepsilon)$-approximation f-tolerant subgraph H of any given graph G with $O(f + n)$ edges in total.

The number of edges used in our fault-tolerant subgraph in Result 3 is clearly optimal (up to constant factors). In Appendix A.2, we show that by modifying the lower bound of [23] in the communication model, we can also prove that the approximation ratio of (3/2) is optimal for any f-tolerant subgraph with $O(f)$ edges, hence proving that Result 3 is optimal in a strong sense. We also show that several natural strategies for this problem cannot achieve

better than 2-approximation, hence motivating our more sophisticated approach toward this problem (see Appendix A.3).

The qualitative message of our work is clear: *An EDCS is a robust matching sparsifier under all three notions of sparsification described earlier, which leads to simpler and improved algorithms for a wide range of problems involving sparsification for matching problems in a unified way.*

Overall Proof Strategy

Recall that our algorithm in all of the results above is simply to compute an EDCS H of the input graph G (or G_A in the communication problem). The analysis then depends on the specific notion of sparsification at hand, but the same high- level idea applies to all three cases. In each case, we have an original graph G, and then a modified graph G^* produced by changes to G: G^* is $G_A \cup G_B$ in the communication model, the realized subgraph G_p in the stochastic matching, and the graph $G \setminus F$ after adversarially removing edges F in the fault-tolerant matching problem. Let H be the EDCS that our algorithm computes in G, and let H^* be the graph that results from H due to the modifications made to G. If we could show that H^* is an EDCS of G^* then the proof would be complete, since we know that an EDCS is guaranteed to contain an almost $(3/2)$-approximate matching. Unfortunately, in all the three problems that we study it might not be the case that H^* is an EDCS of G^*. Instead in each case we are able to exhibit subgraphs $\widetilde{H} \subseteq H^*$ and $\widetilde{G} \subseteq G^*$ such that $\widetilde{H}$ is an EDCS of $\widetilde{G}$, and size of maximum matching of $\widetilde{G}$ and G^* differ by at most a $(1+\varepsilon)$ factor. This guarantees an approximation ratio of almost $(3/2)(1+\varepsilon)$ (precisely what we achieve in all three results above), since the EDCS $\widetilde{H}$ preserves the maximum matching in $\widetilde{G}$ to within an almost $(3/2)$-approximation and $\widetilde{H}$ is a subgraph of H.

Organization. The rest of the paper is organized as follows. Section 2 includes notation, simple preliminaries, and existing work on the EDCS. In Section 3, we present a significantly simpler proof of the fact that an EDCS contains an almost $(3/2)$-approximation matching (originally proved in [14]). Sections 4, 5, and 6 prove the sparsification properties of the EDCS in, respectively, the one-way communication complexity of matching (Result 1), the stochastic matching problem (Result 2), and the fault-tolerant matching problem (Result 3). These three sections are designed to be self-contained (beside assuming the background in Section 2) to allow the reader to directly consider the part of most interest. The appendix contains some secondary observations.

2 Preliminaries and Notation

Notation. For any integer $t \geq 1$, $[t] := \{1, \ldots, t\}$. For a graph $G(V, E)$ and a set of vertices $U \subseteq V$, $N_G(U)$ denotes the neighbors of vertices in U in G and $E_G(U)$ denotes the set of edges incident on U. Similarly, for a set of edges $F \subseteq E$, $V(F)$ denotes the set of vertices incident on these edges. For any vertex $v \in V$, we use $\deg_G(v)$ to denote the degree of $v \in V$ in G (we may drop the subscript G in these definitions if it is clear from the context). We use $\mu(G)$ to denote the size of the maximum matching in the graph G.

Throughout the paper, we use the following two standard variants of the Chernoff bound.

▶ **Proposition 2** (Chernoff Bound). *Suppose $X_1, \ldots, X_t$ are t independent random variables that take values in $[0, 1]$. Let $X := \sum_{i=1}^{t} X_i$ and assume $\mathbb{E}[X] \leq \lambda$. For any $\delta > 0$ and*

integer $k \geq 1$,

$$\Pr\Big(|X - \mathbb{E}[X]| \geq \delta \cdot \lambda\Big) \leq 2 \cdot \exp\Big(-\frac{\delta^2 \cdot \lambda}{3}\Big),$$
$$\Pr\Big(|X - \mathbb{E}[X]| \geq k\Big) \leq 2 \cdot \exp\Big(-\frac{2k^2}{t}\Big).$$

We also need the following basic variant of Lovasz Local Lemma (LLL).

▶ **Proposition 3** (Lovasz Local Lemma; cf. [21, 1])**.** *Let* $p \in (0,1)$ *and* $d \geq 1$*. Suppose* $\mathcal{E}_1, \ldots, \mathcal{E}_t$ *are* t *events such that* $\Pr(\mathcal{E}_i) \leq p$ *for all* $i \in [t]$ *and each* $\mathcal{E}_i$ *is mutually independent of all but (at most)* d *other events* $\mathcal{E}_j$*. If* $p \cdot (d+1) < 1/e$ *then* $\Pr\big(\cap_{i=1}^{n} \overline{\mathcal{E}_i}\big) > 0$*.*

Hall's Theorem. We use the following standard extension of the Hall's marriage theorem for characterizing maximum matching size in bipartite graphs.

▶ **Proposition 4** (Extended Hall's marriage theorem; cf. [24])**.** *Let* $G(L, R, E)$ *be any bipartite graph with* $|L| = |R| = n$*. Then,* $\max\Big(|A| - |N(A)|\Big) = n - \mu(G)$*, where* A *ranges over* L *or* R*. We refer to such set* A *as a* witness set*.*

Proposition 4 follows from Tutte-Berge formula for matching size in general graphs [33, 12] or a simple extension of the proof of Hall's marriage theorem itself

Previously Known Properties of the EDCS

Recall the definition of an EDCS in Definition 1. It is not hard to show that an EDCS always exists as long as $\beta > \beta^-$ (see, e.g. [2]). For completeness, we repeat the proof in the Appendix A.1.

▶ **Proposition 5** (cf. [13, 14, 2])**.** *Any graph* G *contains an* $\text{EDCS}(G, \beta, \beta^-)$ *for any parameters* $\beta > \beta^-$*, which can be found in polynomial time.*

The key property of an EDCS, originally proved in [13, 14], is that it contains an almost (3/2)-approximate matching.

▶ **Lemma 6** ([13, 14])**.** *Let* $G(V, E)$ *be any graph and* $\varepsilon < 1/2$ *be a parameter. For parameters* $\lambda \leq \frac{\varepsilon}{100}$*,* $\beta \geq 32\lambda^{-3}$*, and* $\beta^- \geq (1-\lambda) \cdot \beta$*, in any subgraph* $H := \text{EDCS}(G, \beta, \beta^-)$*,* $\mu(G) \leq \big(\frac{3}{2} + \varepsilon\big) \cdot \mu(H)$*.*

Another particularly useful (technical) property of an EDCS is that it "balances" the degree of vertices and their neighbors in the EDCS; this property is implicit in [13] but we explicitly state and prove it here as it shows a main distinction in the properties of EDCS compared to more standard (and less robust) subgraphs in this context such as b-matchings.

▶ **Proposition 7.** *Let* $H := \text{EDCS}(G, \beta, \beta^-)$ *and* U *be any subset of vertices. If average degree of* U *in* H *is* $\bar{d}$ *then average degree of* $N_H(U)$ *from edges incident on* U *is* $\leq \beta - \bar{d}$*.*

Proof. Let H' be a subgraph of H containing the edges incident on U. Let $W := N_{H'}(U) = N_H(U)$ and $E' = E_H(U, W) = E_{H'}(U, W)$. We are interested in upper bounding the quantity $|E'| / |W|$. Firstly, by Property (P1) of EDCS, we have that $\sum_{(u,v) \in E'} \deg_{H'}(u) + \deg_{H'}(v) \leq$

$\beta \cdot |E'|$. We write the LHS in this equation as:

$$\begin{aligned}\sum_{(u,v)\in E'} \deg_{H'}(u) + \deg_{H'}(v) &= \sum_{u\in U} (\deg_{H'}(u))^2 + \sum_{w\in W} (\deg_{H'}(w))^2 \\ &\geq \sum_{u\in U} (\frac{|E'|}{|U|})^2 + \sum_{w\in W} (\frac{|E'|}{|W|})^2 \\ \text{(as } \textstyle\sum_u \deg_{H'}(u) = \sum_w \deg_{H'}(w) = \left|E'\right| \text{ and each is minimized when the summands are equal.)} \\ &= |E'| \cdot \left(\bar{d} + |E'| \,/\, |W|\right).\end{aligned}$$

By plugging in this bound in LHS above, we obtain $|E'| \,/\, |W| \leq \beta - \bar{d}$, finalizing the proof. ◀

3 A Simpler Proof of the Key Property of an EDCS

In this section we provide a much simpler proof of the key property that an EDCS contains an almost (3/2)-approximate matching. This lemma was previously used in [13, 14, 2]. Our proof is self-contained to this section, and for general graphs, our new proof even improves the dependence of β on parameter λ from $1/\lambda^3$ to (roughly) $1/\lambda^2$, thus allowing for an even sparser EDCS.

The proof contains two steps. We first give a simple and streamlined proof that an EDCS contains a (3/2)-approximate matching in bipartite graphs. Our proof in this part is similar to [13] but instead of modeling matchings as flows and using cut-flow duality, we directly work with matchings by using Hall's theorem. The main part of the proof however is to extend this result to general graphs. For this, we give a simple reduction that extends the result on bipartite graphs to general graphs by taking advantage of the "robust" nature of EDCS. This allows us to bypass the complicated arguments in [14] specific to non-bipartite graphs and to obtain the result directly from the one for bipartite graphs (the paper of [14] explicitly acknowledges the complexity of the proof and asks for a more "natural" approach).

A Slightly Simpler Proof for Bipartite Graphs

Our new proof should be compared to Lemma 2 in Section 4.1 of the Arxiv version of [13].

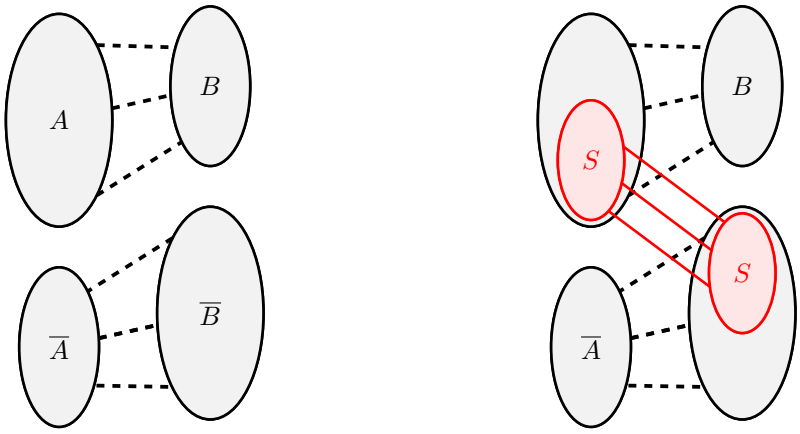
▶ **Lemma 8.** *Let $G(L, R, E)$ be any bipartite graph and $\varepsilon < 1/2$ be a parameter. For $\lambda \leq \frac{\varepsilon}{4}$, $\beta \geq 2\lambda^{-1}$, and $\beta^- \geq (1-\lambda)\cdot\beta$, in any subgraph $H := \text{EDCS}(G, \beta, \beta^-)$, $\mu(G) \leq \left(\frac{3}{2} + \varepsilon\right)\cdot\mu(H)$.*

Proof. Fix any $H := \text{EDCS}(G, \beta, \beta^-)$ and let A be any of its witness sets in extended Hall's marriage theorem of Proposition 4 and $B := N_H(A)$. Without loss of generality, let us assume A is a subset of L. Define $\overline{A} := L \setminus A$, $\overline{B} := R \setminus B$ (see Figure 1). By Proposition 4,

$$\left|\overline{A}\right| + |B| = n - (|A| - |B|) \leq n - (n - \mu(H)) = \mu(H). \tag{1}$$

On the other hand, since G has a matching of size $\mu(G)$, we need to have a matching M of size $(\mu(G) - \mu(H))$ between A and $\overline{B}$ as otherwise by Proposition 4, A would be a witness set in G that implies the maximum matching of G is smaller than $\mu(G)$ (to see why the set of edges between A and $\overline{B}$ is a matching simply apply Proposition 4 to a subgraph of G containing only a maximum matching of G). Let $S \subseteq A \cup \overline{B}$ be the end points of this matching (see Figure 1). As edges in M are all missing from H, by Property (P2) of EDCS H, we have that,

$$\sum_{v\in S} \deg_H(v) = \sum_{(u,v)\in M} (\deg_H(u) + \deg_H(v)) \geq (\mu(G) - \mu(H)) \cdot \beta^-. \tag{2}$$



(a) A and $B := N_H(A)$ form a Hall's theorem witness set in EDCS H and $\left|\overline{A} \cup B\right| \leq \mu(H)$.

(b) There is a matching of size $\mu(G) - \mu(H)$ between A and $\overline{B}$ (i.e., the set S) in $G \setminus H$.

Figure 1 The partitioning of vertices used in the proof of Lemma 8.

Consequently, as $|S| = 2(\mu(G) - \mu(H))$, the average degree of S is $\geq \beta^-/2$. As such, by Proposition 7, the average degree of of $N_H(S)$ (from S) is at most $\beta - \beta^-/2 \leq (1+\lambda)\beta/2$. Finally, note that $N_H(S) \subseteq \overline{A} \cup B$ as there are no edges between A and $\overline{B}$ in H, and hence by Eq (1), $|N_H(S)| \leq \mu(H)$. By double counting the number of edges between S and $N_H(S)$, i.e., $E_H(S)$:

$$|E_H(S)| \geq |S| \cdot \beta^-/2 \geq 2(\mu(G) - \mu(H)) \cdot \beta^-/2,$$
$$|E_H(S)| \leq |N_H(S)|\,(1+\lambda)\beta/2 \leq \mu(H) \cdot (1+\lambda)\beta/2.$$

This implies that,

$$2\mu(G) \leq 2\mu(H) + \mu(H) \cdot (1+\lambda)(\beta/2) \cdot (2/\beta^-) \leq 3\mu(H) \cdot \frac{1+\lambda}{1-\lambda} \leq 3\mu(H)(1+\varepsilon).$$

Reorganizing the terms above finalizes the proof. ◀

A Much Simpler Proof for Non-bipartite Graphs

Our new proof in this part should be compared to Lemma 5.1 on page 699 in [14]: see Appendix B of their paper for the full proof, as well Section 4 for an additional auxiliary claim needed.

▶ **Lemma 9.** *Let $G(V,E)$ be any graph and $\varepsilon < 1/2$ be a parameter. For $\lambda \leq \frac{\varepsilon}{32}$, $\beta \geq 8\lambda^{-2}\log(1/\lambda)$, and $\beta^- \geq (1-\lambda)\cdot\beta$, in any subgraph $H := \text{EDCS}(G,\beta,\beta^-)$, $\mu(G) \leq \left(\frac{3}{2}+\varepsilon\right)\cdot\mu(H)$.*

Proof. The proof is based on the probabilistic method and Lovasz Local Lemma. Let $M^\star$ be a maximum matching of size $\mu(G)$ in G. Consider the following randomly chosen bipartite subgraph $\widetilde{G}(L,R,\widetilde{E})$ of G with respect to $M^\star$, where $L \cup R = V$:

- For any edge $(u,v) \in M^\star$, with probability $1/2$, u belongs to L and v belongs to R, and with probability $1/2$, the opposite (the choices between different edges of $M^\star$ are independent).

- For any vertex $v \in V$ not matched by $M^\star$, we assign v to L or R uniformly at random (again, the choices are independent across vertices).
- The set of edges in $\widetilde{E}$ are all edges in E with one end point in L and the other one in R.

Define $\widetilde{H} := H \cap \widetilde{G}$. We argue that as H is an EDCS for G, $\widetilde{H}$ also remains an EDCS for $\widetilde{G}$ with non-zero probability. Formally,

▶ **Claim 10.** *$\widetilde{H}$ is an* EDCS$(\widetilde{G}, \widetilde{\beta}, \widetilde{\beta}^-)$ *for $\widetilde{\beta} = (1+4\lambda)\beta/2$ and $\widetilde{\beta}^- = (1-5\lambda)\beta^-/2$ with probability strictly larger than zero (over the randomness of $\widetilde{G}$).*

Before we prove Claim 10, we argue why it implies Lemma 9. Let $\widetilde{G}$ be chosen such that $\widetilde{H}$ is an EDCS$(\widetilde{G}, \widetilde{\beta}, \widetilde{\beta}^-)$ for parameters in Claim 10 (by Claim 10, such a choice of $\widetilde{G}$ always exist). By construction of $\widetilde{G}$, $M^\star \subseteq \widetilde{E}$ and hence $\mu(\widetilde{G}) = \mu(G)$. On the other hand, $\widetilde{G}$ is now a bipartite graph and $\widetilde{H}$ is its EDCS with appropriate parameters. We can hence apply Lemma 8 and obtain that $\mu(\widetilde{G}) \leq (3/2+\varepsilon)\mu(\widetilde{H})$. As $\widetilde{H} \subseteq H$, $\mu(\widetilde{H}) \leq \mu(H)$, and hence $(\mu(\widetilde{G}) =)\mu(G) \leq (3/2+\varepsilon)\mu(H)$, proving the assertion in the lemma statement. It thus only remains to prove Claim 10.

Proof of Claim 10. Fix any vertex $v \in V$, let $d_v := \deg_H(v)$ and $N_H(v) := \{u_1, \ldots, u_{d_v}\}$ be the neighbors of v in H. Let us assume v is chosen in L in $\widetilde{G}$ (the other case is symmetric). Hence, degree of v in $\widetilde{H}$ is exactly equal to the number of vertices in $N_H(v)$ that are chosen in R. As such, by construction of $\widetilde{G}$, $\mathbb{E}\left[\deg_{\widetilde{H}}(v)\right] = d_v/2$ ($+1$ iff v is incident on $M^\star \cap H$). Moreover, if two vertices u_i, u_j in $N_H(v)$ are matched by $M^\star$, then exactly one of them appears as a neighbor to v in $\widetilde{H}$ and otherwise the choices are independent. Hence, by Chernoff bound (Proposition 2),

$$\Pr\left(\left|\deg_{\widetilde{H}}(v) - d_v/2\right| \geq \lambda \cdot \beta\right) \leq \exp\left(-\frac{2\lambda^2 \cdot \beta^2}{\beta}\right) \leq \exp\left(-4\log\beta\right) \leq \frac{1}{\beta^4}.$$
$$\text{(as } \beta \geq 8\lambda^{-2}\log(1/\lambda) \text{ and hence } \beta \geq 2\lambda^{-2}\cdot\log\beta\text{)}$$

Define $\mathcal{E}_v$ as the event that $\left|\deg_{\widetilde{H}}(v) - d_v/2\right| \geq \lambda \cdot \beta$. Note that $\mathcal{E}_v$ depends only on the choice of vertices in $N_H(v)$ and hence can depend on at most β^2 other events $\mathcal{E}_u$ for vertices u which are neighbors to $N_H(v)$ (recall that for all $u \in V$, $\deg_H(u) \leq \beta$ in H by Property (P1) of EDCS). As such, we can apply Lovasz Local Lemma (Proposition 3) to argue that with probability strictly more than zero, $\cap_{v \in V} \overline{\mathcal{E}_v}$ happens. In the following, we condition on this event and argue that in this case, $\widetilde{H}$ is an EDCS of $\widetilde{G}$ with appropriate parameters. To do this, we only need to prove that both Property (P1) and Property (P2) hold for the EDCS $\widetilde{H}$ (with the choice of $\widetilde{\beta}$ and $\widetilde{\beta}^-$).

We first prove Property (P1) of EDCS $\widetilde{H}$. Let (u,v) be any edge in $\widetilde{H}$. By events $\overline{\mathcal{E}_v}$ and $\overline{\mathcal{E}_u}$,

$$\deg_{\widetilde{H}}(u) + \deg_{\widetilde{H}}(v) \leq \frac{1}{2} \cdot (\deg_H(u) + \deg_H(v)) + 2\lambda\beta \leq \beta/2 + 2\lambda\beta = (1+4\lambda)\cdot\beta/2,$$

where the second inequality is by Property (P1) of EDCS H as (u,v) belongs to H as well. We now prove Property (P2) of EDCS $\widetilde{H}$. Let (u,v) be any edge in $\widetilde{G} \setminus \widetilde{H}$. Again, by $\overline{\mathcal{E}_v}$ and $\overline{\mathcal{E}_u}$,

$$\begin{aligned}\deg_{\widetilde{H}}(u) + \deg_{\widetilde{H}}(v) &\geq \frac{1}{2} \cdot (\deg_H(u) + \deg_H(v)) - 2\lambda\beta \\ &\geq \beta^-/2 - 2\lambda(1-\lambda)\beta^- \\ &\geq (1-5\lambda)\cdot\beta/2,\end{aligned}$$

where the second inequality is by Property (P2) of EDCS H as $(u,v) \in G \setminus H$. ◀ Claim 10 ◀

Lemma 9 now follows immediately from Claim 10 as argued above. ◀ Lemma 9 ◀

4 One-Way Communication Complexity of Matching

In the one-way communication model, Alice and Bob are given graphs $G_A(V, E_A)$ and $G_B(V, E_B)$, respectively, and the goal is for Alice to send a small message to Bob such that Bob can output a large approximate matching in $E_A \cup E_B$. In this section, we show that if Alice communicates an appropriate EDCS of G_A, then Bob is able to output an almost (3/2)-approximate matching.

▶ **Theorem 11** (Formalizing Result 1)**.** *There exists a deterministic poly-time one-way communication protocol that given any $\varepsilon > 0$, computes a $(3/2+\varepsilon)$-approximation to maximum matching using $O(\frac{n \cdot \log(1/\varepsilon)}{\varepsilon^2})$ communication from Alice to Bob.*

Theorem 11 is based on the following protocol:

A one-way communication protocol for maximum matching.

1. Alice sends $H := \text{EDCS}(G_A, \beta, \beta-1)$ for $\beta := 32 \cdot \varepsilon^{-2} \cdot \log(1/\varepsilon)$ to Bob.
2. Bob computes a maximum matching in $H \cup G_B$ and outputs it as the solution.

By Proposition 5, the EDCS H computed by Alice always exists and can be found in polynomial time. Moreover, by Property (P1) of EDCS H, the total number of edges (and hence the message size) sent by Alice is $O(n\beta)$. We now prove the correctness of the protocol which concludes the proof of Theorem 11.

▶ **Lemma 12.** $\mu(G_A \cup G_B) \leq (3/2+\varepsilon) \cdot \mu(H \cup G_B)$.

Proof. Let $M^\star$ be a maximum matching in $G_A \cup G_B$ and $M^\star_A$ and $M^\star_B$ be its edges in G_A and G_B, respectively. Let $\widetilde{G} := G_A \cup M^\star_B$ and note that $\mu(\widetilde{G}) = \mu(G)$ simply because $M^\star$ belongs to $\widetilde{G}$. Define the following subgraph $\widetilde{H} \subseteq H \cup M^\star_B$ (and hence $\subseteq H \cup G_B$): $\widetilde{H}$ contains all edges in H and any edge $(u,v) \in M^\star_B$ such that $\deg_H(u) + \deg_H(v) \leq \beta$. In the following, we prove that $(\mu(G) =)\mu(\widetilde{G}) \leq (3/2+\varepsilon) \cdot \mu(\widetilde{H})$, which finalizes the proof as $\mu(\widetilde{H}) \leq \mu(H \cup G_B)$.

We show that $\widetilde{H}$ is an $\text{EDCS}(\widetilde{G}, \beta+2, \beta-1)$ and apply Lemma 9 to argue that $\widetilde{H}$ contains a (3/2)-approximate matching of $\widetilde{G}$. We prove the EDCS properties of $\widetilde{H}$ using the fact that for $v \in V$, $\deg_{\widetilde{H}}(v) \in \{\deg_H(v), \deg_H(v)+1\}$ as $\widetilde{H}$ is obtained by adding a matching ($\subseteq M^\star_B$) to H.

- Property (P1) of EDCS $\widetilde{H}$: For an edge $(u,v) \in \widetilde{H}$,

  if $(u,v) \in H$ then: $\deg_{\widetilde{H}}(u) + \deg_{\widetilde{H}}(v) \leq \deg_H(u) + \deg_H(v) + 2 \leq \beta + 2$, (by Property (P1) of EDCS H of G_A)

  if $(u,v) \in M^\star_B$ then: $\deg_{\widetilde{H}}(u) + \deg_{\widetilde{H}}(v) \leq \deg_H(u) + \deg_H(v) + 2 \leq \beta + 2$. (as $(u,v) \in M^\star_B$ is inserted to $\widetilde{H}$ iff $\deg_H(u) + \deg_H(v) \leq \beta$)

- Property (P2) of EDCS $\widetilde{H}$: For an edge $(u,v) \in \widetilde{G} \setminus \widetilde{H}$,

  if $(u,v) \in G_A \setminus H$ then: $\deg_{\widetilde{H}}(u) + \deg_{\widetilde{H}}(v) \geq \deg_H(u) + \deg_H(v) \geq \beta - 1$, (by Property (P2) of EDCS H of G_A)

  if $(u,v) \in M^\star_B \setminus \widetilde{H}$ then: $\deg_{\widetilde{H}}(u) + \deg_{\widetilde{H}}(v) \geq \deg_H(u) + \deg_H(v) > \beta$. (as $(u,v) \in M^\star_B$ is not inserted to $\widetilde{H}$ iff $\deg_H(u) + \deg_H(v) > \beta$)

As such, $\widetilde{H}$ is an EDCS$(\widetilde{G}, \beta+2, \beta-1)$. By Lemma 9 and the choice of parameter β, we obtain that $\mu(\widetilde{G}) \leq (3/2+\varepsilon) \cdot \mu(\widetilde{H})$, finalizing the proof. ◀

5 The Stochastic Matching Problem

Recall that in the stochastic matching problem, the goal is to compute a bounded-degree subgraph H of a given graph G, such that $\mathbb{E}[\mu(H_p)]$ is a good approximation of $\mathbb{E}[\mu(G_p)]$, where G_p is a realization of G (i.e a subgraph where every edge is sampled with probability p), and $H_p = H \cap G_p$. In this section, we formalize Result 2 by proving the following theorem.

▶ **Theorem 13** (Formalizing Result 2). *There exists a deterministic poly-time algorithm that given a graph $G(V,E)$ and parameters $\varepsilon, p > 0$ with $\varepsilon < 1/4$, computes a subgraph $H(V,E_H)$ of G with maximum degree $O(\frac{\log(1/\varepsilon p)}{\varepsilon^2 \cdot p})$ such that the ratio of the expected size of a maximum matching in realizations of G to realizations of H is at most $(3/2+\varepsilon)$, i.e., $\mathbb{E}[\mu(G_p)] \leq (3/2+\varepsilon) \cdot \mathbb{E}[\mu(H_p)]$.*

We note that while in Theorem 13, we state the bound in expectation, the same result also holds with high probability as long as $\mu(G) = \omega(1/p)$ (i.e., just barely more than a constant), by concentration of maximum matching size in edge-sampled subgraphs (see, e.g. [2], Lemma 3.1). The algorithm in Theorem 13 simply computes an EDCS of the input graph as follows:

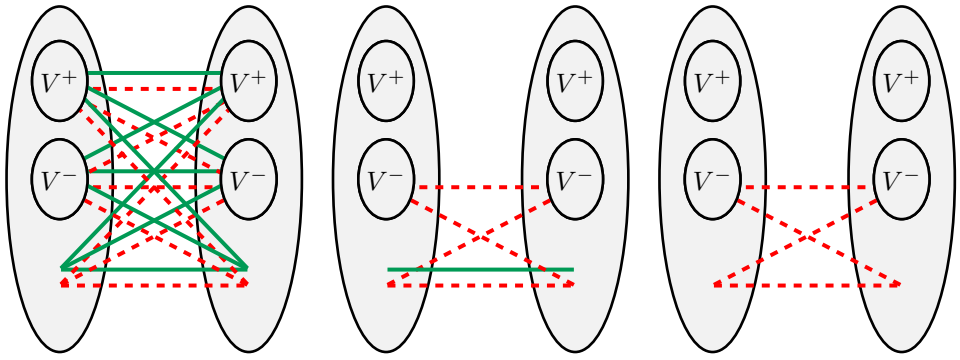
> An algorithm for the stochastic matching problem.
>
> Output the subgraph $H := \text{EDCS}(G, \beta, \beta-1)$ for $\beta := \frac{C \log(1/\varepsilon p)}{\varepsilon^2 p}$, for large enough constant C.

By Proposition 5, the EDCS H in the above algorithm always exists and can be found in polynomial time. Moreover, by Property (P1) of EDCS H, the total number of edges in this subgraph is $O(n\beta)$. We now prove the bound on the approximation ratio which concludes the proof of Theorem 13 (by re-parametrizing ε to be a constant factor smaller).

▶ **Lemma 14.** *Let $H_p := H \cap G_p$ denote a realization of H; then $\mathbb{E}[\mu(G_p)] \leq (3/2+O(\varepsilon)) \cdot \mathbb{E}[\mu(H_p)]$ where the randomness is taken over the realization G_p of G.*

Suppose first that H_p were an EDCS of G_p; we would be immediately done in this case as we could have applied Lemma 9 directly and prove Lemma 14. Unfortunately, however, this might not be the case. Instead, we exhibit subgraphs $\widetilde{H}_p \subseteq H_p$ and $\widetilde{G}_p \subseteq G_p$ with the following properties:

1. $\mathbb{E}[\mu(G_p)] \leq (1+\varepsilon)\,\mathbb{E}\left[\mu(\widetilde{G}_p)\right]$, where the expectation is taken over realizations G_p.
2. $\widetilde{H}_p$ is an EDCS$(G, (1+\varepsilon)p \cdot \beta, (1-2\varepsilon)p \cdot \beta)$ for $\widetilde{G}_p$.



(a) Realized Graph G_p. **(b)** Subgraph $\widetilde{G}_p \subseteq G_p$. **(c)** Subgraph $\widetilde{H}_p \subseteq H_p$.

Figure 2 Illustration of the sets V^+, V^- and the subgraphs $\widetilde{G}_p$ and $\widetilde{H}_p$ in the proof of Lemma 14 on a bipartite graph G. Here, (green) solid lines denote the edges of G_p that appear in each subgraph and (red) dashed lines denote the edges of H_p.

Showing these properties concludes the proof of Lemma 14, as for the EDCS in item (2) above, we have $\frac{(1+\varepsilon)p\cdot\beta}{(1-2\varepsilon)\cdot p\beta} = 1+O(\varepsilon)$, so by Lemma 9 we get that $\mu(\widetilde{G}_p) \leq (3/2+O(\varepsilon))\cdot\mu(\widetilde{H}_p)$. Combining this with item (1) then concludes $\mathbb{E}[\mu(G_p)] \leq (1+\varepsilon)\cdot(3/2+\varepsilon)\,\mathbb{E}[\mu(H_p)]$.

It now remains to exhibit $\widetilde{H}_p$ and $\widetilde{G}_p$ that satisfy the main properties stated above. Note that for any vertex $v \in V$, we have $\mathbb{E}\left[\deg_{H_p}(v)\right] = p\cdot\deg_H(v)$ by definition of a realization G_p (and hence H_p). We now want to separate out vertices that deviate significantly from this expectation.

▶ **Definition 15.** Let $V^+ \subseteq V$ contain all vertices v for which $\deg_{H_p}(v) > p\cdot\deg_H(v)+\varepsilon p\beta/2$. Similarly, let V^- contain all vertices v such that $\deg_{H_p}(v) < p\cdot\deg_H(v) - \varepsilon p\beta/2$ OR there exists an edge $(v,w) \in H$ such that $w \in V^+$, i.e., if v is neighbor to V^+.

▶ **Claim 16.** *$\mathbb{E}[|V^+|] \leq \varepsilon^7 p^7 \mu(G)$ and $\mathbb{E}[|V^-|] \leq \varepsilon^4 p^4 \mu(G)$, where the expectation is over the realization G_p of G. As we a result we also have $\mathbb{E}[|V^+| + |V^-|] \leq \varepsilon^3 p^3 \mu(G)$.*

Before proving this claim, let us consider why it completes the larger proof.

Proof of Lemma 14 (assuming Claim 16). To prove Lemma 14 it is enough to show the existence of subgraphs $\widetilde{G}_p$ and $\widetilde{H}_p$ that satisfy the properties above. We define $\widetilde{G}_p$ as follows: the vertex set is V and the edge-set is the same as G_p, except we remove all edges incident to V^+ and all edges $(u,v) \notin H$ that are incident to V^-. We define $\widetilde{H}_p$ to be the subgraph of H_p induced by the vertex set $V \setminus V^+$, that is, $\widetilde{H}_p$ contains all edges of H_p except those incident to V^+; see Figure 2.

For item (1), note that $\widetilde{G}_p$ differs from G_p by vertices in $V^+ \cup V^-$, so $\mu(\widetilde{G}_p) \geq \mu(G_p) - |V^+| - |V^-|$. It is also clear that $\mathbb{E}[\mu(G_p)] \geq p\cdot\mu(G)$ (as each edge in G is sampled w.p. p in G_p). By Claim 16,

$$\mathbb{E}\left[\mu(\widetilde{G}_p)\right] \geq \mathbb{E}[\mu(G_p)] - \mathbb{E}\left[|V^+| - |V^-|\right] \geq \mathbb{E}[\mu(G_p)] - p^3\varepsilon^3\mu(G) \geq (1-\varepsilon^3)\,\mathbb{E}[\mu(G_p)]\,.$$

The above equation then implies the desired $\mathbb{E}[\mu(G_p)] \leq (1+\varepsilon)\,\mathbb{E}\left[\mu(\widetilde{G}_p)\right]$.

For item (2), let us verify Property (P1) and Property (P2) for EDCS $\widetilde{H}_p$ of $\widetilde{G}_p$. Neither $\widetilde{H}_p$ nor $\widetilde{G}_p$ have any edge incident on V^+ and hence we can ignore these vertices entirely. Thus, for all vertices v we have $\deg_{\widetilde{H}_p}(v) \leq p\cdot\deg_H(v) + \varepsilon p\beta/2$, and for all $v \notin V^-$ we have

$\deg_{\widetilde{H}_p}(v) \geq p \cdot \deg_H(v) - \varepsilon p\beta/2$. Moreover, recall that $\widetilde{G}_p \setminus \widetilde{H}_p$ contains no edges incident to V^-. As such,

- Property (P1) of EDCS $\widetilde{H}_p$: For an edge $(u,v) \in \widetilde{H}_p$,

$$\deg_{\widetilde{H}_p}(u) + \deg_{\widetilde{H}_p}(v) \leq p \cdot \deg_H(u) + p \cdot \deg_H(v) + \varepsilon p\beta \leq (1+\varepsilon)p\beta.$$

(by Property (P1) of EDCS H of G)

- Property (P2) of EDCS $\widetilde{H}_p$: For any edge $(u,v) \in \widetilde{G}_p \setminus \widetilde{H}_p$, we have $u, v \notin V^-$ so:

$$\deg_{\widetilde{H}_p}(u) + \deg_{\widetilde{H}_p}(v) \geq p \cdot \deg_H(u) + p \cdot \deg_H(v) - \varepsilon p\beta \geq (1-2\varepsilon)p\beta.$$

(by Property (P2) of EDCS H of G)

This concludes the proof of Lemma 14 (assuming Claim 16). ◀ Lemma 14 ◀

All that remains is to prove Claim 16.

Proof of Claim 16. Let us start by bounding the size of V^+. Consider any vertex $v \in V$. We know that $\deg_H(v) \leq \beta$. Each edge then has probability p of appearing in H_p, so $\mathbb{E}\left[\deg_{H_p}(v)\right] = p \cdot \deg_H(v) \leq p\beta$. By the multiplicative Chernoff bound in Proposition 2 with $\lambda = p\beta$:

$$\Pr[v \in V^+] = \Pr[\deg_{H_p}(v) \geq p \cdot \deg_H(v) + \varepsilon p\beta/2] \leq e^{-O(\varepsilon^2 p\beta)} \leq e^{-O(\log(\varepsilon^{-1}p^{-1}))} \leq K^{-2}\varepsilon^{10}p^{10},$$

where K is a large constant and the last two inequalities follow from the fact that we set $\beta := \frac{C\log(1/\varepsilon p)}{\varepsilon^2 p}$, for large enough constant C. (Note that since constant C is in the exponent, we can easily set C large enough to achieve the final probability with a constant $K > C$.) This probability bound shows that $\mathbb{E}[|V^+|] \leq nK^{-2}\varepsilon^{10}p^{10}$, but that is not quite good enough since we want a dependence on $\mu(G)$ instead of on n. To achieve this, we observe that the total number of edges in H is at most $\beta\mu(G)$: the reason is that G has a vertex cover of size at most $2\mu(G)$, and all vertices in H have degree at most β (by Property (P1) of EDCS H). There are thus at most $2\beta\mu(G)$ vertices that have non-zero degree in H, each of which has at most a $\varepsilon^{10}p^{10}$ probability of being in V^+; all vertices with zero degree in H are clearly not in V^+ by definition. We thus have $\mathbb{E}[|V^+|] \leq 2\beta\mu(G) \cdot K^{-2}\varepsilon^{10}p^{10} \leq K^{-1}\varepsilon^7 p^7 \mu(G)$, where in the last inequality we use that $K > C$.

Let us now consider V^-. First let us bound the number of vertices $v \in V^-$ for which $\deg_{H_p}(v) < p \cdot \deg_H(v) - \varepsilon p\beta/2$. By an analogous argument to the one above, we have that the expected number of such vertices is at most $\varepsilon^7 p^7 \mu(G)$. A vertex can also end up in V^- because it has a neighbor in V^+ in H. But each vertex in H has degree at most β so we have

$$\mathbb{E}\left[|V^-|\right] \leq \varepsilon^7 p^7 \mu(G) + \beta\,\mathbb{E}\left[|V^+|\right] \leq \varepsilon^4 p^4 \mu(G),$$

where the last inequality again uses that $K > C$. ◀

▶ Remark. Interestingly, our result in Theorem 13 continues to hold as it is even when the edges sampled in realizations of G_p are only $\Theta(1/p)$-wise independent, by simply using a Chernoff bound for bounded-independence random variables (see, e.g. [31]) in the proof of Claim 16. Allowing correlation in the process of edge sampling is highly relevant to the main application of this problem to the kidney exchange setting (see [15]). To our knowledge, our algorithm is the first to work with such a little amount of independence between the edges.

6 A Fault-Tolerant Subgraph for Matching

In the fault-tolerant matching problem, we are given a graph $G(V, E)$ and an integer $f \geq 1$, and our goal is to compute a subgraph H of G, named an f-tolerant subgraph, such that for any subset $F \subseteq E$ of size f, $H \setminus F$ contains an approximate maximum matching of $G \setminus F$. We show that,

▶ **Theorem 17** (Formalizing Result 3). *There exists a deterministic poly-time algorithm that given any $\varepsilon > 0$ and integer $f \geq 1$, computes a $(3/2 + \varepsilon)$-approximate f-tolerant subgraph H of any given graph G with $O(\varepsilon^{-2} \cdot (n \log(1/\varepsilon) + f))$ edges.*

The algorithm in Theorem 17 simply computes an EDCS of the input graph as follows:

An algorithm for the fault-tolerant matching problem.

1. Define $\mu_{\min} := \min_F \Big(\mu(G \setminus F)\Big)$, where F is taken over all subsets of E with size f.
2. Output $H := \text{EDCS}(G, \beta, \beta - 1)$ for $\beta := \frac{C \cdot f}{\varepsilon^2 \cdot \mu_{\min}} + \frac{C \cdot \log(1/\varepsilon)}{\varepsilon^2}$ for a constant $C > 0$.

By Proposition 5, the EDCS H in the above algorithm always exists and can be found in polynomial time. The above algorithm as stated however is not a polynomial time algorithm because it is not clear how to compute the quantity $\mu_{\min}$. Nevertheless, for simplicity, we work with the above algorithm throughout this section, and at the end show how to fix this problem and obtain a poly-time algorithm. We start by proving that the subgraph H only has $O(f + n)$ edges.

▶ **Lemma 18.** *The total number of edges in H is $O(\frac{f}{\varepsilon^2} + n \cdot \frac{\log(1/\varepsilon)}{\varepsilon^2})$.*

Proof. Let $F^\star$ be a subset of E with size f such that $\mu_{\min} = \mu(G \setminus F^\star)$. Let $M^\star$ be a maximum matching of size $\mu_{\min}$ in $G \setminus F^\star$. Note that $V(M^\star)$ is a vertex cover for $G \setminus F^\star$. This means that all edges in G except for f of them are incident on $V(M^\star)$. As no vertex in the EDCS H can have degree more than β by Property (P1) of EDCS, the degree of vertices in $V(M^\star)$ in $E \setminus F^\star$ is at most β. This implies that:

$$|E_H| \leq |V(M^\star)| \cdot \beta + |F^\star| \leq 2\mu_{\min} \cdot \left(\frac{C \cdot f}{\varepsilon^2 \cdot \mu_{\min}} + \frac{C \cdot \log(1/\varepsilon)}{\varepsilon^2}\right) + f$$
$$= O(\frac{f}{\varepsilon^2} + n \cdot \frac{\log(1/\varepsilon)}{\varepsilon^2}),$$

finalizing the proof. ◀

We now prove the correctness of the algorithm in the following lemma.

▶ **Lemma 19.** *Fix any subset $F \subseteq E$ of size f and define $G_F := G \setminus F$ and $H_F := H \setminus F$. Then, $\mu(G_F) \leq (3/2 + O(\varepsilon)) \cdot \mu(H_F)$.*

We first need some definitions. We say that a vertex $v \in V$ is *bad* iff $\deg_{H_F}(v) < \deg_H(v) - \varepsilon\beta$, i.e., at least $\varepsilon\beta$ edges incident on v (in H) are deleted by F. We use B_F to denote the set of bad vertices with respect to F, and bound $|B_F|$ in the following claim.

▶ **Claim 20.** *Number of bad vertices in H_F is at most $|B_F| \leq \varepsilon \cdot \mu(G_F)$.*

Proof. Any deleted edge can decrease the degrees of exactly two vertices. Any vertex becomes bad iff at least $\varepsilon\beta$ edges incident on it from H_F are removed. As such, $|B_F| \leq \frac{2f}{\varepsilon\cdot\beta} \leq \frac{2f\cdot\varepsilon^2\cdot\mu_{\min}}{\varepsilon\cdot C\cdot f} \leq \varepsilon\cdot\mu(G_F)$, for sufficiently large $C > 0$, and since $\mu(G_F) \geq \mu_{\min}$ by definition.
◀ Claim 20 ◀

Proof of Lemma 19. Define a subgraph $\widetilde{G}_F \subseteq G_F$ as follows: $V(\widetilde{G}_F) = V(G_F)$ $(= V(G))$ and edges in $\widetilde{G}_F$ are all edges in G_F except that we remove any edge $(u,v) \in G_F$ such that $(u,v) \notin H_F$ and either of u or v is a bad vertex. We prove that $\mu(\widetilde{G}_F)$ is at least $(1-\varepsilon)$ fraction of $\mu(G_F)$, and moreover, H_F is an EDCS of $\widetilde{G}_F$ with appropriate parameters. We can then apply Lemma 9 to obtain that $\mu(G_F) \leq (1+2\varepsilon)\mu(\widetilde{G}_F) \leq (1+\varepsilon)\cdot(3/2+O(\varepsilon))\mu(H_F)$, finalizing the proof.

We first prove the bound on $\mu(\widetilde{G}_F)$. Fix any maximum matching M in G_F. It can have at most $|B_F|$ edges incident on vertices of B_F. Hence, even if we remove all edges incident on B_F, the size of this matching would be at least $\mu(G_F) - \varepsilon\cdot\mu(G_F)$, by the bound of $|B_F| \leq \varepsilon\cdot\mu(G_F)$ in Claim 20. However, this matching belongs to $\widetilde{G}_F$ entirely by the definition of this subgraph, and hence we have, $\mu(G_F) \leq (1+2\varepsilon)\mu(\widetilde{G}_F)$.

We now prove that H_F is an EDCS$(\widetilde{G}_F, \beta, (1-2\varepsilon)\beta - 1)$ of $\widetilde{G}_F$. It suffices to prove the two properties of EDCS for H_F using the fact that $\deg_{H_F}(v) \in [\deg_H(v) - \varepsilon\beta, \deg_H(v)]$ for vertices in $V \setminus B_F$, and that all edges incident on B_F in $\widetilde{G}_F$ also belong to H_F.

- Property (P1) of EDCS H_F of $\widetilde{G}_F$: For any edge $(u,v) \in H_F$:

$$\deg_{H_F}(u) + \deg_{H_F}(v) \leq \deg_H(u) + \deg_H(v) \leq \beta.$$
(by Property (P1) of EDCS H of G)

- Property (P2) of EDCS H_F of $\widetilde{G}_F$: For any edge $(u,v) \in \widetilde{G}_F \setminus H_F$ both $u, v \in V \setminus B_F$ and so:

$$\deg_{H_F}(u) + \deg_{H_F}(v) \geq \deg_H(u) + \deg_H(v) - 2\varepsilon\beta \geq (1-2\varepsilon)\beta - 1.$$
(by Property (P2) of EDCS H of G as (u,v) is missing from H)

As such, H_F is an EDCS$(\widetilde{G}_F, \beta, (1-2\varepsilon)\beta - 1)$ of $\widetilde{G}_F$ and by the lower bound on value of β in the algorithm (the second term in definition of β), we can apply Lemma 9, and obtain that $\mu(\widetilde{G}_F) \leq (3/2 + O(\varepsilon)) \cdot \mu(H_F)$, finalizing the proof. ◀

Theorem 17 now follows from Lemmas 18 and 19 by re-parametrizing ε to a sufficiently smaller constant factor of ε (by picking the integer C large enough) modulo the fact that the algorithm designed in this section is not a polynomial time algorithm. To make the algorithm polynomial time, we only need to make a simple modification: instead of finding $\mu_{\min}$ explicitly, we find the smallest value of β (by searching over all n possible choices of β, or by doing a binary search) such that the EDCS H has at least $\frac{2\cdot C\cdot f}{\varepsilon^2} + \frac{n\cdot C\cdot\log(1/\varepsilon)}{\varepsilon^2}$ many edges. By the proof of Lemma 18, any EDCS of G can have at most $2\mu_{\min}\cdot\beta + f$ edges. This implies that the chosen $\beta \geq \frac{C\cdot f}{\varepsilon^2\cdot\mu_{\min}} + \frac{C\cdot\log(1/\varepsilon)}{\varepsilon^2}$ as needed in the algorithm. This concludes the proof, as by definition of β, H has $O(\frac{C\cdot f}{\varepsilon^2} + \frac{n\cdot C\cdot\log(1/\varepsilon)}{\varepsilon^2})$ many edges, and hence satisfies the sparsity requirements of Theorem 17.

References

1 Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

2 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. *CoRR*, abs/1711.03076. To appear in SODA 2019, 2017.

3 Sepehr Assadi, Sanjeev Khanna, and Yang Li. The Stochastic Matching Problem with (Very) Few Queries. In *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 43–60, 2016.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On Estimating Maximum Matching Size in Graph Streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1723–1742, 2017.

5 Sepehr Assadi, Sanjeev Khanna, and Yang Li. The Stochastic Matching Problem: Beating Half with a Non-Adaptive Algorithm. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 99–116, 2017.

6 Baruch Awerbuch. Complexity of Network Synchronization. *J. ACM*, 32(4):804–823, 1985.

7 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 509–518, 2016.

8 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 255–262, 2009.

9 Soheil Behnezhad, Alireza Farhadi, MohammadTaghi Hajiaghayi, and Nima Reyhani. Stochastic Matching with Few Queries: New Algorithms and Tools. In *Manuscript. To appear in SODA 2019.*, 2018.

10 Soheil Behnezhad and Nima Reyhani. Almost Optimal Stochastic Weighted Matching with Few Queries. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 235–249, 2018.

11 András A. Benczúr and David R. Karger. Approximating s-t Minimum Cuts in $\tilde{O}(n^2)$ Time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55, 1996.

12 Claude Berge. *The theory of graphs*. Courier Corporation, 1962.

13 Aaron Bernstein and Cliff Stein. Fully Dynamic Matching in Bipartite Graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 167–179, 2015.

14 Aaron Bernstein and Cliff Stein. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711, 2016.

15 Avrim Blum, John P. Dickerson, Nika Haghtalab, Ariel D. Procaccia, Tuomas Sandholm, and Ankit Sharma. Ignorance is Almost Bliss: Near-Optimal Stochastic Matching With Few Queries. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC '15, Portland, OR, USA, June 15-19, 2015*, pages 325–342, 2015.

16 Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal Vertex Fault Tolerant Spanners (for fixed stretch). In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1884–1900, 2018.

17 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving Distances in Very Faulty Graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 73:1–73:14, 2017.

18 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 414–423, 2003.

19 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 435–444, 2009.

20 Don Coppersmith and Michael Elkin. Sparse Sourcewise and Pairwise Distance Preservers. *SIAM J. Discrete Math.*, 20(2):463–501, 2006.

21 Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *COLLOQUIA MATHEMATICA SOCIETATIS JANOS BOLYAI 10. INFINITE AND FINITE SETS, KESZTHELY (HUNGARY)*. Citeseer, 1973.

22 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 71–80, 2011.

23 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the Communication and Streaming Complexity of Maximum Bipartite Matching. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 468–485. SIAM, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095116.2095157`.

24 Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

25 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013. `doi:10.1137/1.9781611973105.121`.

26 David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 648–657, 1994.

27 Euiwoong Lee and Sahil Singla. Maximum Matching in the Online Batch-Arrival Model. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 355–367, 2017.

28 David Peleg. As Good as It Gets: Competitive Fault Tolerance in Network Structures. In *Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings*, pages 35–46, 2009.

29 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

30 Imre Z Ruzsa and Endre Szemerédi. Triple systems with no six points carrying three triangles. *Combinatorics (Keszthely, 1976), Coll. Math. Soc. J. Bolyai*, 18:939–945, 1978.

31 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding Bounds for Applications with Limited Independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

32 Daniel A. Spielman and Shang-Hua Teng. Spectral Sparsification of Graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

33 William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

34 Yutaro Yamaguchi and Takanori Maehara. Stochastic Packing Integer Programs with Few Queries. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 293–310, 2018.

A Missing Details and Proofs

A.1 Proof of Proposition 5

We give the proof of this proposition following the argument of [2], which itself was based on [14].

Proof. We give a polynomial local search algorithm for constructing an EDCS H of the graph G which also implies the existence of H. The algorithm is as follows. Start with empty graph H. While there exists an edge in H or $G \setminus H$ that violates Property (P1) or Property (P2) of EDCS, respectively, fix this edge by removing it from H for the former or inserting it to H for the latter.

We prove that this algorithm terminates after polynomial number of steps which implies both the existence of the EDCS as well as give a polynomial time algorithm for computing it. We define the following potential function Φ for this task:

$$\Phi_1(H) := (\beta - 1/2) \cdot \sum_{u \in V(H)} \deg_H(u), \qquad \Phi_2(H) := \sum_{(u,v) \in E(H)} \left(\deg_H(u) + \deg_H(v)\right),$$
$$\Phi(H) := \Phi_1(H) - \Phi_2(H).$$

We claim that after fixing each edge in H in the algorithm, Φ increases by at least 1. Since max-value of Φ is $O(n \cdot \beta^2)$, this implies that this procedure terminates in $O(n \cdot \beta^2)$ steps.

Let (u, v) be the fixed edge at this step, H_1 be the subgraph before fixing the edge (u, v), and H_2 be the resulting subgraph. Suppose first that the edge (u, v) was violating Property (P1) of EDCS. As the only change is in the degrees of vertices u and v, Φ_1 decreases by $(2\beta - 1)$. On the other hand, $\deg_{H_1}(u) + \deg_{H_1}(v) \geq \beta + 1$ originally (as (u, v) was violating Property (P1) of EDCS) and hence after removing (u, v), Φ_2 also decreases by $\beta + 1$. Additionally, for each neighbor w of u and v in H_2, after removing the edge (u, v), $\deg_{H_2}(w)$ decreases by one. As there are at least $\deg_{H_2}(u) + \deg_{H_2}(v) = \deg_{H_1}(u) + \deg_{H_1}(v) - 2 \geq \beta - 1$ choices for w, this means that in total, Φ_2 decreases by at least $(\beta + 1) + (\beta - 1) = 2\beta$. As a result, in this case $\Phi = \Phi_1 - \Phi_2$ increases by at least 1 after fixing the edge (u, v).

Now suppose that the edge (u, v) was violating Property (P2) of EDCS instead. In this case, degree of vertices u and v both increase by one, hence Φ_1 increases by $2\beta - 1$. Additionally, since edge (u, v) was violating Property (P2) we have $\deg_{H_1}(u) + \deg_{H_1}(v) \leq \beta^- - 1$, so the addition of edge (u, v) decreases Φ_2 by at most $\deg_{H_2}(u) + \deg_{H_2}(v) = \deg_{H_1}(u) + \deg_{H_1}(v) + 2 \leq \beta^- + 1$. Moreover, for each neighbor w of u and v, after adding the edge (u, v), $\deg_{H_2}(w)$ increases by one and since there are at most $\deg_{H_1}(u) + \deg_{H_1}(v) \leq \beta^- - 1$ choices for w, Φ_2 decreases in total by at most $(\beta^- + 1) + (\beta^- - 1) = 2\beta^-$. Since $\beta^- \leq \beta - 1$, we have that Φ increases by at least $(2\beta - 1) - (2\beta^-) \geq 1$ after fixing the edge (u, v), finalizing the proof. ◀

A.2 Optimality of the (3/2)-Approximation Ratio in Result 3

Our argument is a simple modification of the one in [23] for proving a lower bound on the one-way communication complexity of approximating matching and is provided for the sake of completeness.

Let $G_1(V_1, E_1)$ be a graph on N vertices such that its edges can be partitioned into $t := N^{\Omega(1/\log\log N)}$ *induced* matchings $M_1, \ldots, M_t$ of size $(1 - \delta)N/4$ for arbitrarily small constant $\delta > 0$. These graphs are referred to as (r, t)-Ruzsa-Szemerédi graphs [30] ((r, t)-RS graphs for short) and have been studied extensively in the literature (see [4, 23] for more details). In particular, the existence of such graphs with parameters mentioned above is proven in [23].

Let $G(V, E)$ be a graph with $n = 2N$ vertices consisting of $G_1(V_1, E_1)$ plus N additional vertices U that are connected via a perfect matching M_U to V_1. In the following, we prove that any f-fault tolerant subgraph H of G that achieves a $(3/2 - \varepsilon)$-approximation for some constant $\varepsilon > 0$ when $f = \Theta(n)$ requires $n^{1+\Omega(1/\log\log n)} = \omega(f)$ edges.

Suppose towards a contradiction that H contains $o(m)$ edges where m is the number of edges in the graph G. As edges in G_1 are partitioned into induced matchings $M_1, \ldots, M_t$, it means that there exists some induced matching M_i such that only $o(1)$ fraction of its edges belong to H. Let the set of deleted edge F be only the set of edges in the perfect matching between U and V_1, namely, M_U, which are incident to $V(M_i)$. The number of deleted edges is $O(n)$ and after deletion, M_U has size $N - (1-\delta)N/2 = (1+\delta)N/2$. As such, $\mu(G \setminus F) \geq (1+\delta)N/2 + (1-\delta)N/4 \geq 3N/4$, by picking the remainder of the matching M_U and the induced matching M_i (which is not incident on remainder of M_U by construction). However, we argue that $\mu(H \setminus F) \leq (1+\delta)N/2 + o(N)$, simply because only $o(N)$ edges of M_i belong H and all other matchings are incident to the remaining edges of M_U (we can assume remaining edges of M_U belong to *any* maximum matching of $H \setminus F$ because they are incident on degree one vertices). As such, $\mu(H \setminus F) < (2/3 + 2\delta)\mu(G \setminus F)$. By picking $\delta < \varepsilon/4$, we obtain that H is not a $(3/2 - \varepsilon)$-approximate f-fault tolerant subgraph of G.

A.3 Other Standard Algorithms for Fault-Tolerant Matching

Since the goal in fault-tolerant matching is to prepare for adversarial deletions, the most natural approach seem to be adding many different matchings by a finding maximum matching in G, adding it to the subgraph H, deleting it from G, and repeating until we have $O(f + n)$ edges. A similar approach would be to let H be a maximum b-matching, with b set appropriately to end up with $O(f + n)$ edges. We show a lower bound of 2 on the approximation ratio of these approaches.

Consider the following approach first: find a maximum matching M in G, add all the edges of M to the fault-tolerant subgraph H, remove all the edges of M from G, and repeat until the graph contains $C(f+n)$ edges for some large constant C. For $f = n/5$, we present a graph G where this approach yields a graph H where $\mu(H) = \mu(G)/2$. The graph is bipartite and the vertex set is partitioned into 5 sets X, Y, Y', Z, Z', each of size $n/5$. There is an edge in G from every vertex in X to every vertex in Y or Z, and there are also exactly $n/5$ vertex-disjoint edges from Y to Y', and similarly from Z to Z'; those are all the edge of G. The fault tolerant algorithm might choose the following subgraph H: H contains a perfect matching from Y to Y' and from Z to Z', as well as many edges from X to Y, but no edges from X to Z. (The algorithm can end up with such an H by first choosing the maximum matching in G that consists of the edges from Y to Y' and from Z to Z'; then for all future iterations the maximum matching size is only $|X| = n/5$, so the algorithm might always pick a maximum matching that only contains edges between X and Y.) Now consider the set of failures F which consists of the $n/5$ edges from Z to Z'. It is clear that $\mu(G \setminus F) = 2n/5$, while $\mu(H \setminus F) = n/5$. Note also that allowing H to contain more than $O(n+f)$ edges would still not allow this approach to break through the 2-approximation: in this lower-bound instance, even if H was allowed to have up to $n^2/100$ edges, H might still not contain any edges from X to Z, and so we would still have $\mu(H \setminus F) = n/5 = \mu(G \setminus F)/2$.

The other natural approach is to let H contain the edges of a maximum b-matching in G, where b is set to a value for which the resulting b-matching still contains $\Theta(f+n)$ edges. The lower-bound graph G is exactly the same as above, though in this case we use $f = 2n/5$. The maximum b-matching H might then contain the edges from Y to Y' and Z to Z', a single matching of size $n/5$ from X to Z, and then many edges from X to Y. It is easy to see that

this is a maximum b-matching. Now consider the following set F of deletions: F contains all edges from Z to Z', as well as the $n/5$ edges in H from X to Z. It is easy to see that we once again have $\mu(H) = n/5$ and $\mu(G) = 2n/5$. Also as above, setting B to be very large and allowing H to have $n^2/100$ edges would still not break through the 2-approximation.

A New Application of Orthogonal Range Searching for Computing Giant Graph Diameters

Guillaume Ducoffe
National Institute for Research and Development in Informatics,
The Research Institute of the University of Bucharest ICUB,
University of Bucharest, Faculty of Mathematics and Computer Science, Romania
guillaume.ducoffe@ici.ro

Abstract

A well-known problem for which it is difficult to improve the textbook algorithm is computing the graph diameter. We present two versions of a simple algorithm (one being Monte Carlo and the other deterministic) that for every fixed h and unweighted undirected graph G with n vertices and m edges, either correctly concludes that $diam(G) < hn$ or outputs $diam(G)$, in time $\mathcal{O}(m + n^{1+o(1)})$. The algorithm combines a simple randomized strategy for this problem (Damaschke, *IWOCA'16*) with a popular framework for computing graph distances that is based on range trees (Cabello and Knauer, *Computational Geometry'09*). We also prove that under the Strong Exponential Time Hypothesis (SETH), we cannot compute the diameter of a given n-vertex graph in truly subquadratic time, even if the diameter is an $\Theta(n/\log n)$.

2012 ACM Subject Classification Theory of computation → Shortest paths, Theory of computation → Problems, reductions and completeness

Keywords and phrases Graph diameter, Orthogonal Range Queries, Hardness in P, FPT in P

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.12

Funding This work was supported by the Institutional research programme PN 1819 "Advanced IT resources to support digital transformation processes in the economy and society - RESINFO-TD" (2018), project PN 1819-01-01"Modeling, simulation, optimization of complex systems and decision support in new areas of IT&C research", funded by the Ministry of Research and Innovation, Romania. This work was also supported by a grant of Romanian Ministry of Research and Innovation CCCDI-UEFISCDI. project no. 17PCCDI/2018.

Acknowledgements We wish to thank the referees for their careful reading of the first version of this manuscript, and their useful comments.

1 Introduction

We refer to [5] for any undefined terminology. Graphs in this study are finite, simple, connected and unweighted. For every graph $G = (V, E)$, let $n := |V|$ and $m := |E|$. The distance $dist_G(u, v)$ between any two vertices $u, v \in V$ is defined as the minimum number of edges on a uv-path in G. A *layer* is any set $L_i(v) := \{u \in V \mid dist_G(u, v) = i\}$ for some $v \in V$ and integer $i \geq 0$. Finally, the eccentricity of vertex v, denoted $ecc_G(v)$, is equal to $\max_{u \in V} dist_G(u, v)$, and the diameter of G, denoted $diam(G)$, is equal to $\max_{v \in V} ecc_G(v)$.

Computing the diameter of a graph is a fundamental problem with countless applications in computer science and other domains. As every undergraduate student (should) know, this problem can be solved in roughly quadratic time by running a single-source shortest-path algorithm from every vertex of the graph. It has been asked repeatedly whether one could improve on this textbook algorithm, in order to achieve truly subquadratic-time computation

of the diameter. Unfortunately, the answer to that question seems to be 'No' [15]. Specifically, under SETH it is already hard to recognize *split graphs* with diameter 2, and this holds even if their clique-number is an $\Theta(\log n)$ [6][1]. Small diameter graphs are usually said to be the hardest case for the problem, in the sense that as the diameter gets polynomial in n we can obtain an almost optimal $(1 - n^{-\mathcal{O}(1)})$-approximation in truly subquadratic time [3, 10]. However, the *exact* computation of such "giant" graph diameters has been less studied.

In [11], Damaschke asked whether one can compute the graph diameter in nearly linear time assuming it is a large fraction of the number of vertices. His question seems relevant to the study of chain-like structures, e.g.,, in road networks or chain molecules. Specifically, we consider the following problem in this note:

▶ **Problem 1** (h-DIAMETER).
Input: *A graph $G = (V, E)$; a constant $h \in (0; 1)$.*
Output: *The exact diameter of G if it is at least hn (otherwise, any value $< hn$).*

As a partial answer to Problem 1, Damaschke presented a deterministic linear-time algorithm for the special case $h > 1/2$. The latter is based on the decomposition of a graph by its biconnected components and a delicate removal procedure of irrelevant subgraphs. As noted by Damaschke himself, qualitatively different methods are needed to solve the general case. In [11], he also presented a Monte Carlo $\mathcal{O}(m + n \log n)$-time algorithm for the case $h > 1/3$. The probability of a correct result depends on the simultaneity of several random events. Recently in a master's thesis [4] some of Damaschke's students generalized his ideas to any h, thereby obtaining a (not so simple) $\mathcal{O}(n^2)$-time algorithm for the general case. The "big-oh" notation hides a large constant-factor in h.

1.1 Our results

We answer positively to the open question of [11]. Specifically, we first present a Monte Carlo algorithm that runs in time $\mathcal{O}(\frac{1}{h} \cdot (m + 2^{\mathcal{O}(\frac{1}{h})} n^{1+o(1)}))$ and that, given as entry a graph G such that $diam(G) \geq hn$, outputs with constant probability its diameter (Theorem 5).

- For that, we follow the same general (and quite intuitive) strategy as in [11, 4]: finding a separator of size $\mathcal{O}(1/h)$ that disconnects the two ends of some arbitrary diametral path. Such a separator can be easily computed, with constant probability, by performing a BFS from a random vertex v and choosing a smallest layer $L_i(v) := \{u \in V \mid dist_G(u, v) = i\}$ in the range $i \in \{hn/3, \ldots, 2hn/3\}$. Although a similar approach was used in [11, 4] our proofs are, in our opinion, cleaner and more direct than the ones given in these previous works. In particular, the correctness of our algorithm only depends on the random choice of the starting vertex v.
- Then, once we computed a separator as described above, we are left with computing the maximum distance between two vertices it disconnects. Previous works [11, 4] reduce this computation to a new problem called LARGEST MIXED SUM. Instead, we can directly apply a popular framework for computing graph distances that is based on a *range tree* [8]. Doing so, we give a new simple application of this textbook data-structure.
- Finally, we remark that in order to make our algorithm deterministic, it suffices to run a BFS from every vertex v into some (hn/c)-distance dominating set, for some small constant c (instead of picking this vertex randomly). We end up observing that such a distance dominating set of size $\left(\frac{1}{h}\right)^{\mathcal{O}(1)}$ can be computed by using a few BFS (Theorem 7).

[1] The logarithmic bounds on the clique-number are not explicitly stated in [6]. Nevertheless, they can be deduced from an easy application of the Sparsification Lemma, as noted e.g., in [1] for similar constructions.

For nonconstant h, our algorithm still outperforms the textbook algorithm for DIAMETER provided $h = \omega(1/\log n)$. Perhaps surprisingly, we prove this is (conditionally) optimal: any truly subquadratic algorithm for computing the diameter of a given n-vertex graph would falsify SETH, even if the diameter is an $\Theta(n/\log n)$ (Theorem 8).

2 Preliminaries

In what follows is a simple observation that is the cornerstone of our algorithms: in any consecutive subsequence of $\Theta(n)$ layers in a BFS-tree, there must be one of size $\mathcal{O}(1)$.

▶ **Lemma 1.** *Let $G = (V, E)$ be a graph of order n, let $0 < p < q < r < 1$, and let $v \in V$ have $ecc_G(v) \geq rn$. There exists $i \in \{\lceil pn \rceil, \ldots, \lfloor qn \rfloor\}$ such that the layer $L_i(v) := \{u \in V \mid dist_G(u, v) = i\}$ contains $< \frac{1-r}{q-p} + 1$ vertices.*

Proof. The number of layers $L_i(v)$ to be considered is:

$$\lfloor qn \rfloor - \lceil pn \rceil + 1 = \lfloor (q-p)n \rfloor + 1 > (q-p)n.$$

We also know that there are $\geq (rn - \lfloor qn \rfloor) + \lceil pn \rceil \geq (r - q + p)n$ vertices that are not contained into any of these layers. Therefore, the maximum number of vertices a smallest such layer can contain is:

$$< \frac{(1 - r + q - p)n}{(q-p)n} = \frac{1-r}{q-p} + 1.$$ ◀

Then, assume one such layer disconnects the two ends of an arbitrary diametral path of the graph. In order to compute the diameter of the graph, we only need to compute the maximum distance between two vertices this layer disconnects. This is a routine that naturally appears in the computation of the diameter and the Wiener index of *bounded treewidth* graphs [1, 7, 8], and as such there already exists a standard method for solving this problem:

▶ **Proposition 2** (implicit in [7]). *Let $G = (V, E)$ be a graph and $S \subseteq V$ be a separator, where $|S| \leq k$. We can compute $D_S := \max\{dist_G(x, y) \mid S \text{ is an } xy\text{-separator}\}$ in time $\mathcal{O}(k \cdot (m + \binom{k+1+\lceil \log n \rceil}{k+1} 2^k n))$, that is in $\mathcal{O}(k \cdot (m + 2^{\mathcal{O}(k)} n^{1+o(1)}))$.*

For the sake of completeness, let us give some intuition of how this above problem relates to *range trees*. Let $S = \{s_1, s_2, \ldots, s_k\}$ and $C_1, C_2, \ldots, C_\ell$ be the connected components of $G \setminus S$. For every $i \in \{1, \ldots, k\}$ we search for the furthest pair x, y such that: (a) S is an xy-separator, and (b) $dist_G(x, s_i) + dist_G(s_i, y) = dist_G(x, y)$, as follows. We first map every $v \in C_j$ to the $(k+1)$-dimensional point $P_v^i = (p_{v,0}^i, p_{v,1}^i, \ldots, p_{v,k}^i)$, where $p_{v,0}^i = j$, $p_{v,t}^i = dist_G(v, s_t) - dist_G(v, s_i)$ for every $t \geq 1$, and we associate to this point the value $f_i(v) = dist_G(v, s_i)$. To understand why, note that if $x \in C_j, y \in C_{j'}$ are such that $j < j'$ and $dist_G(x, s_i) + dist_G(s_i, y) = dist_G(x, y)$, then we have $dist_G(x, s_i) + dist_G(s_i, y) \leq dist_G(x, s_t) + dist_G(s_t, y) \iff -p_{x,t}^i \leq p_{y,t}^i$ for any $t \geq 1$. Therefore, given $x \in C_j$, in order to find a furthest vertex $y \in \bigcup_{j' > j} C_{j'}$ from x, it suffices to compute a point P_y^i such that:

$$p_{x,0}^i < p_{y,0}^i;\ -p_{x,t}^i \leq p_{y,t}^i \text{ for every } t \in \{1, \ldots, k\};\ \text{and } f_i(y) \text{ is maximized.}$$

The $(k+1)$-dimensional range tree is a classical data-structure that can be used in order to solve this above computation. Specifically, given that there are $|V \setminus S| = \mathcal{O}(n)$ points P_v^i to store, we can construct such a range tree in time $\mathcal{O}(k\binom{k+1+\lceil \log n \rceil}{k+1} n)$ in such a way that for every $x \notin S$, the corresponding query (computation of P_y^i) can be answered in time $\mathcal{O}(2^k \binom{k+1+\lceil \log n \rceil}{k+1})$ [14]. We stress that the analysis of this construction is involved, but its implementation is quite straightforward (e.g.,, see [7] for details).

ALGORITHM 1: GiantDiameter.

Input: graph $G = (V, E)$, h.

Output: a lower-bound on $diam(G)$.

1: Let $v \in V$ picked u.a.r.
2: **if** $ecc_G(v) < 2hn/3$ **then**
3: **return** $ecc_G(v)$. *// this occurs with proba.* $\leq 1 - 2h/3$ *if* $diam(G) \geq hn$.
4: Find a layer $i \in \{\lceil hn/3 \rceil, \ldots, \lfloor 2hn/3 \rfloor\}$ s.t. $|L_i(v)| \leq 3/h$.
5: Compute $D_i := \max\{dist_G(x, y) \mid L_i(v)$ is an xy-separator$\}$.
6: **return** $\max\{D_i\} \cup \{ecc_G(u) \mid u \in L_i(v)\}$.

Comparison with previous work. Damaschke's students solved a variant of the above problem in $\mathcal{O}(kn^2)$-time by reducing it to a new problem they called Largest Mixed Sum [4]. Roughly, their solution consists in a brute force range searching. We use range trees in order to improve their running time, although in doing so we sacrifice analytical simplicity. A potential drawback of our algorithms compared to [4] is that the range tree data-structure has relatively high preprocessing and storage costs, that make it less practical for moderate values of k [2]. Some way to address this issue could be the use of alternative data-structures for range searching [2, 9].

3 Monte Carlo algorithm

We present in this section a simple algorithm for the computation of graph diameters that are at least a fixed fraction of the number of vertices. Unlike previous works [11, 4], we use randomization only to choose the starting vertex of our BFS run (Algorithm GiantDiameter). Our algorithm is correct assuming this starting vertex is sufficiently close to an end of some (arbitrary) diametral path. We prove next that it happens with constant probability.

▶ **Lemma 3.** *Let $G = (V, E)$ be a graph and assume $diam(G) \geq hn$. For every vertex $v \in V$ that is drawn u.a.r., the following holds with probability $\geq 2h/3$: There exists a diametral pair $x, y \in V$ such that $dist_G(x, v) \leq hn/3$ (and so, $dist_G(y, v) \geq 2hn/3$).*

Proof. Fix an arbitrary *diametral* path P with ends $x, y \in V$. Every vertex $v \in V(P)$ such that either $dist_G(v, x) \leq hn/3$ or $dist_G(v, y) \leq hn/3$ satisfies the desired property, and there are exactly $2hn/3$ such vertices. ◀

▶ **Proposition 4.** *Let $G = (V, E)$ be a graph and assume $diam(G) \geq hn$. Algorithm* GiantDiameter *correctly computes $diam(G)$ with probability $\geq 2h/3$.*

Proof. By Lemma 1 (applied with $p = h/3$ and $q = r = 2h/3$), a small-size layer $L_i(v)$ as requested by the algorithm always exists if $ecc_G(v) \geq 2hn/3$. Then, the algorithm is correct if there exists a diametral pair $x, y \in V$ such that there is no connected component of $G \setminus L_i(v)$ that both contains x, y (possibly, $x \in L_i(v)$ or $y \in L_i(v)$). This is always the case if $\min\{dist_G(v, x), dist_G(v, y)\} \leq hn/3$ (and so, $\max\{dist_G(v, x), dist_G(v, y)\} \geq 2hn/3$), and by Lemma 3 the latter happens with probability $\geq 2h/3$. ◀

The bottleneck of Algorithm GiantDiameter is the computation of D_i. Using Proposition 2, we can conclude as follows:

▶ **Theorem 5.** *Let h be a fixed constant. In time $\mathcal{O}(\frac{1}{h} \cdot (m + 2^{\mathcal{O}(\frac{1}{h})} n^{1+o(1)}))$, we can either conclude a given graph G has diameter $< hn$, or compute its diameter, with probability of correctness $\geq 2h/3$.*

Proof. We run Algorithm GiantDiameter, whose output is correct with probability $\geq 2h/3$ by Proposition 4. The dominant step for the algorithm is the computation of D_i, that can be done in time $\mathcal{O}(\frac{1}{h} \cdot (m + 2^{\mathcal{O}(\frac{1}{h})} n^{1+o(1)}))$ by Proposition 2. ◀

As usual, the probability of correctness can be increased to $1 - n^{-\mathcal{O}(1)}$ by running Algorithm GiantDiameter $\mathcal{O}(\log n/h)$ times and outputting the maximum distance we obtained.

4 Deterministic algorithm

Next, we show how to derandomize our algorithm from the previous section. We recall that we use randomization only to choose the starting vertex v of some BFS run. Furthermore, the latter vertex v is correctly chosen if it is at a distance $\leq hn/3$ from an end of some arbitrary diametral path. Hence, instead of choosing v at random, we can try *all* the vertices contained into some (hopefully small) $(hn/3)$-distance dominating set. We prove next that there always exists such a set of size polynomial in $1/h$.

▶ **Lemma 6.** *Let $G = (V, E)$ be a graph. In $\mathcal{O}(m + n)$-time, we can output a set S where $|S| = \mathcal{O}(1/h)$ and such that $dist_G(v, S) \leq hn/3$ for every vertex $v \in V$.*

Proof. We use the constructive proof of Meir and Moon on k-distance dominating sets in trees [13]. Specifically, let T be an arbitrary spanning tree of G. Such a tree can be computed in $\mathcal{O}(n + m)$-time by using, say, a breadth-first search. For any integer $k \geq 0$, we will explain next how to construct a k-distance dominating set of size $\left\lceil \frac{n}{k+1} \right\rceil$ for T (and so, also for G) in time $\mathcal{O}(n)$. By setting $k = \lfloor hn/3 \rfloor$, this will prove the lemma.

For that, we first compute the two ends of a diametral path in T, that can be easily done in time $\mathcal{O}(n)$ using two BFS [12]. Let x be any one of these two ends. If $n \leq k + 1$ or more generally, $diam(T) \leq k$ then, we can output $S = \{x\}$. Otherwise, we compute in time $\mathcal{O}(n)$ a breadth-first search from x in T. For every $i \in \{0, 1, 2, \ldots, k\}$, let $S_i := \{v \in V \mid dist_T(x, v) \equiv i \pmod{k+1}\}$. Note that $S_i \neq \emptyset$ for every i since we assume $diam(T) > k$. Furthermore as proved in [13, Theorem 5], S_i is a k-distance dominating set of T for any fixed i. Since there are exactly $k + 1$ possibilities for i, there exists a i_0 such that $|S_{i_0}| \leq \left\lfloor \frac{n}{k+1} \right\rfloor$. We output $S = S_{i_0}$. ◀

▶ **Theorem 7.** *Let h be a fixed constant. In time $\mathcal{O}(\frac{1}{h^2} \cdot (m + 2^{\mathcal{O}(\frac{1}{h})} n^{1+o(1)}))$, we can either conclude a given graph G has diameter $< hn$, or compute its diameter.*

Proof. We apply Lemma 6 in order to compute an $(hn/3)$-distance dominating set S of size $\mathcal{O}(1/h)$. Then, we apply Algorithm GiantDiameter for every $v \in S$, and we output the maximum distance we obtain after these $|S|$ runs. ◀

5 Conditional Lower-bound

Our GiantDiameter algorithm still runs in truly subquadratic time if $h = \omega(1/\log n)$. It would be interesting to compute in truly subquadratic time the *exact* diameter of a graph when it is an $\Theta(n/\log n)$. We prove that it cannot be done under standard complexity assumptions.

▶ **Theorem 8.** *Under SETH, there exists a function $h(n) = \Theta(1/\log n)$ such that the following problem cannot be solved in time $\mathcal{O}(n^{2-\varepsilon})$, for any $\varepsilon > 0$: Given an n-vertex graph G, either correctly decide $diam(G) < h(n)n$, or compute $diam(G)$.*

Proof. Let $G = (K \cup S, E)$ be a n-vertex split graph such that K induces a clique of size $|K| = \mathcal{O}(\log n)$ and S induces a stable set. We construct a graph $G' = (V', E')$ as follows. The vertex-set of G' is partitioned into two disjoint copies S^0, S^1 of the stable set S and n disjoint copies $K_1, K_2, \ldots, K_n$ of the clique K. For every $s \in S$ we denote by s^0 and s^1 the respective copies of s in S^0 and S^1; in the same way, for every $v \in K$ and $i \in \{1, 2, \ldots, n\}$ we denote by v^i the copy of vertex v in K_i. Furthermore, every copy K_i induces a clique, we add an edge $\{v^i, v^{i+1}\}$ for every $v \in K, 1 \leq i < n$, and the two edges $\{s_0, v^1\}, \{s_1, v^n\}$ for every $s \in S, v \in K$ such that $\{s, v\} \in E$. By construction, G' has order $\mathcal{O}(n \log n)$ and size $\mathcal{O}(n \log^2 n)$. We also have:

- For $u, v \in K$, $dist_{G'}(u^i, v^j) = |j - i|$ if $u = v$, and $|j - i| + 1$ otherwise.
- For $s \in S, v \in K$, $dist_{G'}(s_0, v^i) = dist_{G'}(s_1, v^{k-1+i}) = i$ if $\{s, v\} \in E$, and $i+1$ otherwise.
- For $s, s' \in S$, $dist_{G'}(s_0, s'_0) = dist_{G'}(s_1, s'_1) = dist_G(s, s') \leq 3$.
- For $s, s' \in S$, $dist_{G'}(s_0, s'_1) = n - 1 + \max\{2, dist_G(s, s')\}$.

As a result, $n + 1 \leq diam(G') \leq n - 1 + diam(G) \leq n + 2$. In particular, $diam(G') = n - 1 + diam(G)$. This implies that computing $diam(G')$ is $\tilde{\mathcal{O}}(n)$-time equivalent to computing $diam(G)$, that cannot be done in truly subquadratic time under SETH [6, 15]. ◀

References

1. A. Abboud, V. Vassilevska Williams, and J. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA*, pages 377–391. SIAM, 2016. URL: `https://arxiv.org/abs/1506.01799`.
2. J. Bentley and J. Friedman. Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4):397–409, 1979.
3. P. Berman and S. Kasiviswanathan. Faster approximation of distances in graphs. In *WADS*, pages 541–552. Springer, 2007.
4. B. Block and M. Milakovic. Computing Diameters in Slim Graphs. Master's thesis, Chalmers University of Technology, University of Gothenburg, Sweden, 2018. URL: `http://publications.lib.chalmers.se/records/fulltext/255208/255208.pdf`.
5. J. A. Bondy and U. S. R. Murty. *Graph theory*. Grad. Texts in Math., 2008.
6. M. Borassi, P. Crescenzi, and M. Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science*, 322:51–67, 2016. URL: `https://arxiv.org/abs/1407.4972`.
7. K. Bringmann, T. Husfeldt, and M. Magnusson. Multivariate Analysis of Orthogonal Range Searching and Graph Distances Parameterized by Treewidth. In *IPEC*. LIPIcs, 2018. to appear. URL: `https://arxiv.org/abs/1805.07135`.
8. S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.
9. T. Chan. Orthogonal Range Searching in Moderate Dimensions: k-d Trees and Range Trees Strike Back. In *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7226`.
10. S. Chechik, D. Larkin, L. Roditty, G. Schoenebeck, R. Tarjan, and V. Vassilevska Williams. Better approximation algorithms for the graph diameter. In *SODA*, pages 1041–1052. SIAM, 2014.

11 P. Damaschke. Computing Giant Graph Diameters. In *IWOCA*, pages 373–384. Springer, 2016.

12 C. Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math*, 70(185):81, 1869.

13 A. Meir and J. Moon. Relations between packing and covering numbers of a tree. *Pacific Journal of Mathematics*, 61(1):225–233, 1975.

14 L. Monier. Combinatorial solutions of multidimensional divide-and-conquer recurrences. *Journal of Algorithms*, 1(1):60–74, 1980.

15 L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC*, pages 515–524. ACM, 2013.

Simplified and Space-Optimal Semi-Streaming $(2+\varepsilon)$-Approximate Matching

Mohsen Ghaffari
ETH Zurich, Zurich, Switzerland

David Wajc
Carnegie Mellon University, Pittsburgh, USA

Abstract

In a recent breakthrough, Paz and Schwartzman (SODA'17) presented a single-pass $(2+\varepsilon)$-approximation algorithm for the maximum weight matching problem in the semi-streaming model. Their algorithm uses $O(n \log^2 n)$ bits of space, for any constant $\varepsilon > 0$.

We present a simplified and more intuitive primal-dual analysis, for essentially the same algorithm, which also improves the space complexity to the optimal bound of $O(n \log n)$ bits – this is optimal as the output matching requires $\Omega(n \log n)$ bits.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Streaming, Semi-Streaming, Space-Optimal, Matching

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.13

Related Version A full version of the apper is available at [7], `https://arxiv.org/abs/1701.03730`.

Acknowledgements Mohsen Ghaffari is grateful to Gregory Schwartzman for sharing a write-up of [9], and to Ami Paz and Gregory Schwartzman for feedback on an earlier draft of this article. The work of David Wajc is supported in part by NSF grants CCF-1618280, CCF-1814603, CCF-1527110 and NSF CAREER award CCF-1750808.

1 Introduction and Related Work

The maximum weight matching (MWM) problem is a classical optimization problem, with diverse applications, which has been studied extensively since the 1965 work of Edmonds [4]. Naturally, this problem has received significant attention also in the *semi-streaming* model. This is a modern model of computation, introduced by Feigenbaum et al.[6], which is motivated by the need for processing massive graphs whose edge set cannot be stored in memory. In this model, roughly speaking, the edges of the graph arrive in a stream, and the algorithm should process this stream and eventually output its solution – a matching in our case – while using a small memory at all times. Ideally, this memory size should be close to what is needed for storing just the output. More formally, the setting of the MWM problem in the semi-streaming matching is as follows.

MWM in the Semi-Streaming Model. Let $G = (V, E, w)$ be a simple graph with non-negative edge weights $w : E \to \mathbb{R}_{>0}$ (for notational simplicity, we let $w_e = w(e)$). Let $n = |V|$, $m = |E|$. We assume that the edge weights are normalized so that the minimum edge weight is 1 and we use $W = \max_e w_e$ to denote the maximum edge weight. In the semi-streaming model, the input graph G is provided as a stream of edges. In each iteration, the algorithm receives an edge from the stream and processes it. The algorithm has a memory

much smaller than m and thus it cannot store the whole graph. The amount of memory that the algorithm uses is called its *space complexity* and we wish to keep it as small as possible. The objective in the semi-streaming *maximum weight matching* (MWM) problem is that, at the end of the stream, the algorithm outputs a matching whose weight is close to the weight of the maximum weight matching, denoted by $M^* = \arg\max_{\text{matching } M} w(M)$, where $w(M) = \sum_{e \in M} w_e$ is the weight of matching M.

State of the Art. There has been a sequence of successively improved approximation algorithms for MWM in the semi-streaming model. Feigenbaum et al. gave a 6 approximation[6], McGregor gave a 5.828 approximation[8] (and a $2+\varepsilon$ approximation, but using $O(1/\varepsilon^3)$ passes on the input), Epstein et al. gave a $4.911+\varepsilon$ approximation[5] and Crouch and Stubbs gave a $4+\varepsilon$ approximation[3]. However, these approximations remained far from the more natural and familiar 2 approximation which the sequential greedy method provides.

In a recent breakthrough, Paz and Schwartzman[9] presented a truly simple algorithm that achieves an approximation of $2+\varepsilon$ for any constant $\varepsilon > 0$, using $O(n \log^2 n)$ bits of space. More concretely, the algorithm maintains $O(n \log n)$ edges, while working through the stream, and at the end, it computes a matching using these maintained edges.

Our Contribution. We present an alternative analysis of (a slight variant of) the algorithm of Paz and Schwartzman, which has two advantages: (1) it implies that keeping merely $O(n)$ edges suffices, and thus improves the space complexity to $O(n \log n)$ bits, which is optimal, (2) it provides a more intuitive and also more flexible accounting of the approximation.

Concretely, Paz and Schwartzman[9] used an extension of *the Local Ratio theorem*[1, 2] to analyze their algorithm. We instead present an alternative simpler primal-dual argument, which is more flexible and allows us to improve the space-complexity to obtain the optimal space bound. The main appeal of the primal-dual method is in the simple explanation it provides for the extension of the local ratio technique in [9] using little more than LP duality.

Roadmap. In Section 2, as a warm up, we review (a simple version of) the algorithm of Paz and Schwartzman. In Section 3, we present our primal-dual analysis for this algorithm. In Section 4, we present the improved algorithm that achieves a space complexity of $O(n \log n)$, the analysis of which relies crucially on the flexibility of the analysis presented in Section 3.

2 Reviewing the Algorithm of Paz and Schwartzman

2.1 The Basic Algorithm

The starting point in the approach of Paz and Schwartzman[9] is the following basic yet elegant algorithm, implicit in [1, Section 3]. For the sake of explanation, consider a sequential model of computation. We later discuss the adaptation to the streaming model.

> **Basic Algorithm.** Repeatedly select an edge e with positive weight; reduce its weight from itself and its neighboring edges; push e into a stack and continue to the next edge, so long as edges with positive weight remain. At the end; unwind the stack and add the edges greedily to the matching.

In Section 3 we will see that this simple algorithm is 2-approximate.

Implementing the Basic Algorithm in the Semi-Streaming Model. To implement this while streaming, we just need to remember a parameter ϕ_v for each node v. This parameter is the total sum of the weight already reduced from the edges incident on vertex v, due to edges incident on v that were processed and put in the stack before. We assume for now that the space requirement of storing these n numbers is only $O(n \log n)$ bits (say, due to the edge weights having magnitude at most some poly(n)) and discuss the space requirement of these ϕ_v values at the end of the paper.

2.2 The Algorithm with Exponentially Increasing Weights

The space complexity of the basic algorithm can become $\Theta(n^2)$, as it may end up pushing $\Theta(n^2)$ edges into the stack. This brings us to the clever idea of Paz and Schwartzman [9], which reduces the space complexity to the equivalent of keeping $O(n \log W/\varepsilon)$ edges, where W is the normalized maximum edge weight, while still providing a $(2+\varepsilon)$-approximation.

The idea is to ensure that the edges incident on each node v that get pushed into the stack have exponentially increasing weights, by factors of $(1+\varepsilon)$. Thus, at most $O(\log_{1+\varepsilon} W) = O((\log W)/\varepsilon)$ edges per node are added to the stack, and the overall number of edges in the stack is at most $O((n \log W)/\varepsilon)$.

To attain this exponential growth, the method is as follows: When reducing the weight of an edge e from each neighboring edge e', we will decide between deducting either w_e or $(1+\varepsilon)w_e$ from the weight $w_{e'}$. In general, this can be any arbitrary decision. This arbitrary choice may seem mysterious, but as we shall see in Section 3, the particular choice is rather natural when considered in terms of LP duality. In the streaming model, this decision is done when we first see $e' = \{u, v\}$ in the stream, as follows.

- If $w_{e'} < (1+\varepsilon) \cdot (\phi_u + \phi_v)$ – i.e., if e' has less than $(1+\varepsilon)$ times of the total weight of the stacked up edges incident on u or v – then we deduct $(1+\varepsilon)w_e$ from $w_{e'}$ for each stacked up edge e incident on u or v. Hence, we effectively reduce the weight of $w_{e'}$ by $(1+\varepsilon) \cdot (\phi_u + \phi_v)$. This implies that we get left with an edge e' of negative weight, which can be ignored without putting it in the stack.
- Otherwise, if $w_{e'} \geq (1+\varepsilon) \cdot (\phi_u + \phi_v)$, we deduct only w_e from $w_{e'}$, for each edge e incident on u or v that is already in the stack. Thus, in total, we deduct only $(\phi_u + \phi_v)$ weight from $w_{e'}$ for the previously stacked edges. Thus, now we have an edge e' whose leftover weight is $w'_{e'} \geq (1+\varepsilon) \cdot (\phi_u + \phi_v) - (\phi_u + \phi_v) = \varepsilon \cdot (\phi_u + \phi_v)$. Then, we add e' to the stack, and thus ϕ_u and ϕ_v increase by $w'_{e'}$. Therefore, both ϕ_u and ϕ_v increase by at least a $(1+\varepsilon)$ multiplicative factor.

The concrete algorithm that formalizes the above scheme is presented in Algorithm 1.

▶ **Observation 2.1.** *When an edge $e = \{u, v\}$ is added to the stack, the value of ϕ_u increases by a $1+\varepsilon$ factor.*

The above observation also implies that the edges incident on each node that are added to the stack have an exponential growth in weight, which directly implies that the total number of edges pushed to the stack cannot exceed $O(n \log_{1+\varepsilon} W) = O((n \log W)/\varepsilon)$. In Section 3, we prove that this algorithm is $2(1+\varepsilon)$-approximate.

3 Simplified Analysis

In this section we present our primal-dual analysis of Algorithm 1, proving it yields a $2(1+\varepsilon)$ approximation of the MWM. Note that the basic algorithm above can be seen as Algorithm 1 run with $\varepsilon = 0$, and so we obtain that the basic algorithm is a 2-approximate algorithm.

Algorithm 1 The Paz-Schwartzman Algorithm [9] with Exponentially Increasing Weights.

1: $S \leftarrow$ *emptystack*
2: $\forall v \in V : \phi_v = 0$
3: **for** $e = \{u, v\} \in E$ **do**
4: **if** $w_e < (1+\varepsilon) \cdot (\phi_u + \phi_v)$ **then**
5: continue ▷ skip to the next edge
6: $w'_e \leftarrow w_e - (\phi_u + \phi_v)$
7: $\phi_u \leftarrow \phi_u + w'_e$
8: $\phi_v \leftarrow \phi_v + w'_e$
9: $S.push(e)$
10: $M \leftarrow \emptyset$
11: **while** $S \neq \emptyset$ **do**
12: $e \leftarrow S.pop()$
13: **if** $M \cap N(e) = \emptyset$ **then** $M \leftarrow M \cup \{e\}$
14: **return** M

Primal		Dual	
maximize	$\sum_{e \in E} w_e \cdot x_e$	minimize	$\sum_{v \in V} y_v$
subject to:		subject to:	
$\forall v \in V$:	$\sum_{e \ni v} x_e \leq 1$	$\forall \{u, v\} \in E$:	$y_u + y_v \geq w_{\{u,v\}}$
$\forall e \in E$:	$x_e \geq 0$	$\forall v \in V$:	$y_v \geq 0$

Figure 1 The LP relaxation of MWM and its dual.

▶ **Lemma 3.1.** *The matching M returned by Algorithm 1 is a $2(1+\varepsilon)$ approximation of the maximum weight matching.*

To prove Lemma 3.1 we will rely on a few observations. The first observation our duality-based proofs rely on is that $\vec{\phi_v}$ forms a (nearly) feasible solution to the dual of the LP relaxation of the MWM problem, in Figure 1. Indeed, this fact is not accidental, and it is the underlying reason for the choice of when to decrease weights of an edge neighboring a processed edge e by $(1+\varepsilon)w_e$ or by w_e.

▶ **Observation 3.2.** *The variables $\{\phi_v\}_{v \in V}$ scaled up by $1+\varepsilon$ form a feasible dual solution.*

Proof. Each edge $e = \{u, v\} \in E$ has its dual constraint satisfied by $(1+\varepsilon) \cdot \vec{\phi}$ after inspection; i.e., $w_e \leq (1+\varepsilon) \cdot (\phi_u + \phi_v)$. This constraint is either satisfied before e is inspected, or after performing lines 7 and 8, following which the new and old values of ϕ_u and ϕ_v satisfy $w_e \leq w_e + w'_e = (\phi_u^{old} + \phi_v^{old} + w'_e) + w'_e = \phi_u^{new} + \phi_v^{new} \leq (1+\varepsilon) \cdot (\phi_u^{new} + \phi_v^{new})$. As the ϕ_v variables never decrease, each dual constraint is satisfied by $(1+\varepsilon) \cdot \vec{\phi}$ in the end. ◀

By LP duality, the above implies the following upper bound on the optimal matching.

▶ **Corollary 3.3.** *The weight of any matching M^* satisfies*

$$w(M^*) \leq OPT(LP) \leq (1+\varepsilon) \cdot \sum_v \phi_v.$$

Let $\Delta\phi^e$ be the change to $\sum_{v \in V} \phi_v$ in Lines 7 and 8 of the algorithm during the inspection of edge e. Note that if the algorithm does not reach these lines when inspecting edge e (due

to the test in Line 4), then we have $\Delta\phi^e = 0$. By definition, at the time that the algorithm terminates, $\sum_v \phi_v = \sum_e \Delta\phi^e$. The following lemma relates the weight of an edge e to $\Delta\phi^{e'}$ of edges e' incident on a common vertex with e (including e) inspected no later than e.

▶ **Lemma 3.4.** *For each edge $e = \{u, v\} \in E$ added to the stack S, if we denote its preceding neighboring edges by $\mathcal{P}(e) = \{e' \mid e' \cap e \neq \emptyset, e' \text{ inspected no later than } e\}$, then*

$$w_e \geq \frac{1}{2} \cdot \sum_{e' \in \mathcal{P}(e)} \Delta\phi^{e'} = \sum_{e' \in \mathcal{P}(e)} w'_{e'}.$$

Proof. First, we note that $\Delta\phi^e = 2w'_e$, due to Lines 7 and 8, or put otherwise $w'_e = \frac{1}{2} \cdot \Delta\phi^e$. On the other hand, if we denote the values of ϕ_u and ϕ_v before the inspection of e by ϕ^e_u and ϕ^e_v, respectively, we have that $\phi^e_u + \phi^e_v \geq \frac{1}{2} \cdot \sum_{e' \in \mathcal{P}(e) \setminus \{e\}} \Delta\phi^{e'}$. Consequently, we have that indeed

$$w_e = w'_e + (\phi^e_u + \phi^e_v) \geq \frac{1}{2} \cdot \sum_{e' \in \mathcal{P}(e)} \Delta\phi^{e'} = \sum_{e' \in \mathcal{P}(e)} w'_{e'}. \qquad \blacktriangleleft$$

Proof of Lemma 3.1. By the algorithm's definition, every edge ever added to S and not taken into M has a previously-inspected neighboring edge taken into M. So, by Lemma 3.4 and Corollary 3.3 we have that $w(M) = \sum_{e \in M} w_e \geq \frac{1}{2} \sum_e \Delta\phi^e = \frac{1}{2} \sum_v \phi_v \geq \frac{1}{2(1+\varepsilon)} \cdot w(M^*)$. In other words, the matching M output by Algorithm 1 is a $2(1+\varepsilon)$-approximate MWM. ◀

4 Improved Algorithm

The $(2+\epsilon)$-approximate algorithm of the previous section stores $O(n \log W)$ edges for any constant $\epsilon > 0$. To improve the space complexity, we would like to keep only $O(n)$ edges, which is asymptotically the bare minimum necessary for keeping just a matching. For that purpose, we will limit the number of edges incident on each vertex v that are in the stack to a constant $\beta = \frac{3 \log 1/\varepsilon}{\varepsilon} + 1$. When there are more edges, we will take out the earliest one and remove it from the stack. This will be easy to implement using a queue $Q(v)$ for each of vertex v, where we keep the length of the $Q(v)$ capped at β, resulting in $O\big(n \cdot \frac{\log(1/\varepsilon)}{\varepsilon}\big)$ edges stored overall; i.e., $O(n)$ edges, for any constant $\epsilon > 0$. The pseudo-code is presented in Algorithm 2. We will prove in the following that this cannot hurt the approximation factor more than just increasing the parameter ε by a constant factor.

Remark. Paz and Schwartzman[9] used a similar algorithmic idea to keep only $O(n \log n)$ edges in total, instead of $O(n \log W)$ edges.[1] To be precise, they keep $\gamma = \Theta(\log n/\varepsilon)$ edges per node. Unfortunately, this also leads to quite a bit of complications in their analysis, as they need to adapt the local ratio theorem[1, 2] to handle the loss. In a rough sense,

[1] We note that keeping $O(n \log n)$ edges can be done in a much simpler way, by remembering the maximum edge weight W_{max} observed so far in the stream and discarding all edges of weight below $\delta = \varepsilon W_{\max}/(2(1+\varepsilon)n^2)$. This effectively narrows the range of weights that Algorithm 1 sees to $W' = O(n^2/\varepsilon)$, making its related space complexity $O(n \log n)$. On the other hand, ignoring all edges of weight below $\delta \leq \varepsilon W_{\max}/n^2$ can decrease the MWM, $w(M^*)$, by at most $n^2\delta \leq \varepsilon W_{\max} \leq \varepsilon w(M^*)$; that is, a $(1-\varepsilon)$ multiplicative term. Moreover, for each vertex v the edges of weight at most δ can increase ϕ_v by at most $n\delta = \varepsilon W_{\max}/(2(1+\varepsilon)n)$, thus decreasing the effective weight of edges of weight at least δ by no more than $(1+\varepsilon)(\phi_u + \phi_v) \leq \varepsilon W_{\max}/n \leq \varepsilon w(M^*)/n$. The weight of the maximum weight matching M^* under this new weight function is therefore at most $(1-\varepsilon)$ smaller than under w, so running Algorithm 1 on these weights yields a $2(1+O(\varepsilon))$-approximate solution to the MWM under w.

Algorithm 2 The Space-Optimal Algorithm.

1: $S \leftarrow$ *empty stack*
2: $\forall v \in V : Q(v) \leftarrow$ *empty queue*
3: $\forall v \in V : \phi_v = 0$
4: **for** $e = \{u, v\} \in E$ **do**
5: **if** $w_e < (1+\varepsilon) \cdot (\phi_u + \phi_v)$ **then**
6: continue ▷ skip to the next edge
7: $w'_e \leftarrow w_e - (\phi_u + \phi_v)$
8: $\phi_u \leftarrow \phi_u + w'_e$
9: $\phi_v \leftarrow \phi_v + w'_e$
10: $S.push(e)$
11: **for** $x \in \{u, v\}$ **do**
12: $Q(x)$.enqueue(e)
13: **if** $|Q(x)| > \beta = \frac{3\log(1/\varepsilon)}{\varepsilon} + 1$ **then**
14: $e' \leftarrow Q(x).dequeue()$
15: remove e' from the stack S
16: $M \leftarrow \emptyset$
17: **while** $S \neq \emptyset$ **do**
18: $e \leftarrow S.pop()$
19: **if** $M \cap N(e) = \emptyset$ **then** $M \leftarrow M \cup \{e\}$
20: **return** M

their argument was that, per step, the process of limiting the queue size to γ creates a loss of $(1 - \exp(-\gamma))$ factor in the approximation in the accountings of the local ratio theorem. Thus, over all the $O(n^2)$ edges in the stream, the loss is $(1 - \exp(-\gamma))^{O(n^2)}$. The fact that m could be $\Omega(n^2)$ is why they had to set $\gamma = \Theta(\log n)$ to make this loss negligible.

Handling the loss caused by this queue-limitation is much simpler with the simple primal-dual argument that we used in Section 3. Furthermore, our analysis will allow us to curtail the per-node queue size to $\beta = O(1)$, while keeping the loss negligible. We now address the loss due to queue size limitation in Lines 11-15, starting with the following observation.

▶ **Observation 4.1.** *Suppose that an edge $e = \{u, v\}$ in the stack gets removed from the stack because another edge $e' = \{u, v'\}$ was pushed to the stack later and made the size of the queue $Q(v)$ exceed β. Then, we say e' evicted e. In that case, $w'_{e'} \geq w'_e/\varepsilon$ if $\varepsilon \leq 1$.*

Proof. Since e was evicted by e', there must have been $\beta - 1$ edges incident on u that arrived after e (following which $\phi_u \geq w'_e$) but before e' which were pushed into the stack. Hence, because of Observation 2.1, we have that before the arrival of e', $\phi_u \geq (1+\varepsilon)^{\beta-1} w'_e \geq w'_e/\varepsilon^2$ (the last inequality holds because $\beta - 1 = \frac{3\log(1/\varepsilon)}{\varepsilon} \geq \frac{2\log(1/\varepsilon)}{\log(1+\varepsilon)}$ for all $\varepsilon \in (0, 1]$). But since e' was added to the stack, we know that $w'_{e'} \geq \varepsilon(\phi_u + \phi'_v) \geq \varepsilon\phi_u \geq w'_e/\varepsilon$. ◀

The following recursive definition will prove useful when bounding the loss due to eviction of edges from the queue.

▶ **Definition 4.2.** *We say an edge e' which was inserted into S was discarded if it was later removed from S, and say the edge was kept otherwise. We say a discarded edge e' was discarded by a kept edge e if e' was evicted by e or if e' was evicted by an edge e'' which was itself later discarded by e. That is, there is a sequence of evictions which starts with e' and*

ends in edge e where in this sequence, each edge is evicted by the next edge in the chain. We denote the set of edges discarded by e by $\mathcal{D}(e)$.

We now bound the leftover weights of edges discarded by a given edge e.

▶ **Lemma 4.3.** *For all $\varepsilon \leq 1/4$, for each edge $e \in E$, we have $w'_e \geq \frac{1}{4\varepsilon} \cdot \sum_{e' \in \mathcal{D}(e)} w'_{e'}$.*

Proof. The set $\mathcal{D}(e)$ contains at most two edges evicted by e – one for each endpoint of e. By Observation 4.1, we know that every such edge e' evicted by e satisfies $w'_e \geq w'_{e'}/\varepsilon$. Similarly, any edge e' evicted by e can evict at most two edges in $\mathcal{D}(e)$, where each such edge e'' satisfies $w'_{e''} \leq \varepsilon \cdot w'_{e'} \leq \varepsilon^2 \cdot w'_e$, and so on, by induction. Generally, the edges of $\mathcal{D}(e)$ can be partitioned into sets of at most 2^i edges each for all $i \in \mathbb{N}$, where edges e' in the i-th set have $w'_{e'} \leq \varepsilon^i \cdot w'_e$. Summing over all these sets, we find that indeed, as $\varepsilon \leq 1/4$,

$$\sum_{e' \in \mathcal{D}(e)} w'_{e'} \leq \sum_{i=1}^{\infty} (2\varepsilon)^i \cdot w'_e \leq 2\varepsilon \cdot \sum_{i=0}^{\infty} 2^{-i} \cdot w'_e \leq 4\varepsilon \cdot w'_e. \qquad \blacktriangleleft$$

Combining the simple arguments of Section 3 and Lemma 4.3 we obtain the following.

▶ **Theorem 4.4.** *For all $\varepsilon \leq \frac{1}{4}$, Algorithm 2 is $2(1+6\varepsilon)$-approximate.*

Proof. We note that Observation 3.2 (and consequently Corollary 3.3) as well as Lemma 3.4 hold for Algorithm 2, just as they did for Algorithm 1, as their proofs are unaffected by limiting of the queue sizes in Lines 11-15, which is the only difference between these algorithms.

Now, by Lemma 3.4, for each edge $e \in M$, we have $w_e \geq \sum_{e' \in \mathcal{P}(e)} w'_{e'}$. On the other hand, by Lemma 4.3, we have that $4\varepsilon \cdot w_e \geq 4\varepsilon \cdot w'_e \geq \sum_{e' \in \mathcal{D}(e)} w'_{e'}$. Therefore,

$$w_e \cdot (1+4\varepsilon) \geq \sum_{e' \in \mathcal{P}(e)} \left(w'_{e'} + \sum_{e'' \in \mathcal{D}(e')} w'_{e''} \right).$$

But each kept edge e' not added to M is due to a kept edge e which was added to M; that is, some e such that $e' \in \mathcal{P}(e)$. Likewise, each discarded edge is discarded due to some kept edge. Consequently, the right hand side of the above expression summed over all edges of the output matching M is precisely $\sum_{e' \in E} w'_{e'} = \frac{1}{2} \cdot \sum_{e' \in E} \Delta\phi^{e'} = \frac{1}{2} \cdot \sum_{v \in V} \phi_v$, which by Corollary 3.3 yields

$$\sum_{e \in M} w_e \geq \frac{1}{2(1+4\varepsilon)} \cdot \sum_{v \in V} \phi_v \geq \frac{1}{2(1+4\varepsilon)(1+\varepsilon)} \cdot w(M^*) \geq \frac{1}{2(1+6\varepsilon)} \cdot w(M^*). \qquad \blacktriangleleft$$

As the processing time per edge of Algorithm 2 is clearly $O(1)$, we obtain the following.

▶ **Theorem 4.5.** *For any $\varepsilon > 0$, there exists a semi-streaming algorithm computing a $(2+\varepsilon)$-approximation for MWM, using $O(n \log n \cdot \frac{\log(1/\varepsilon)}{\varepsilon})$ space and $O(1)$ processing time.*

Storing ϕ_v values. In the paper we implicitly assumed the maximum edge weight W is $\text{poly}(n)$, implying the n dual variables ϕ_v can be stored using $O(n \log n)$ bits. For general W, this space requirement is $\Omega(n \log W)$. We briefly outline how to improve this space usage to $O(\log \log W + n \log n)$ bits by only keeping the *ratio* of these ϕ_v and the maximum weight observed so far, $W_{\max}$. First, by rounding all edge weights down to the nearest integer power of $(1+\varepsilon)$ (increasing the approximation ratio by at most a $(1+\varepsilon)$ multiplicative factor in the process), we can store $W_{\max}$ by simply storing $\log_{1+\varepsilon} W$, using

$O(\log\log_{1+\varepsilon} W) = O\big(\log\log W + \log(1/\varepsilon)\big)$ bits. Next, upper bounding the contribution of edge weights below $\varepsilon W_{\max}/n^2$ to ϕ_v by $\varepsilon W_{max}/n$, we can store ϕ_v using a bit array representing the $O(\log_{1+\varepsilon}(n^2/\varepsilon)) = O\big(\frac{\log n + \log(1/\varepsilon)}{\varepsilon}\big)$ possible values summed this way by ϕ_v. (Note that the values summed by ϕ_v are all distinct by Observation 2.1 and rounding of weights to powers of $(1+\varepsilon)$.) Arguments similar to those of Footnote 1 show that this approach keeps the $(2+O(\varepsilon))$ approximation guarantee, while by the above it only requires $O(\log\log W + n\log n)$ bits of memory for any constant $\varepsilon > 0$. That is, for any $W = 2^{2^{O(n\log n)}}$, this is still $O(n\log n)$ bits.

References

1 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48(5):1069–1090, 2001.

2 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon Even 1935-2004. *ACM Computing Surveys (CSUR)*, 36(4):422–463, 2004.

3 Michael Crouch and Daniel M Stubbs. Improved Streaming Algorithms for Weighted Matching, via Unweighted Matching. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 96–104, 2014.

4 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

5 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.

6 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

7 Mohsen Ghaffari and David Wajc. Simplified and Space-Optimal Semi-Streaming for $(2+\varepsilon)$-Approximate Matching. *arXiv preprint*, 2018. `arXiv:1701.03730`.

8 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

9 Ami Paz and Gregory Schwartzman. A $(2+\varepsilon)$-Approximation for Maximum Weight Matching in the Semi-Streaming Model. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2153–2161, 2017.

Simple Greedy 2-Approximation Algorithm for the Maximum Genus of a Graph

Michal Kotrbčík
Department of Computer Science, Comenius University, 842 48 Bratislava, Slovakia
kotrbcik@dcs.fmph.uniba.sk

Martin Škoviera[1]
Department of Computer Science, Comenius University, 842 48 Bratislava, Slovakia
skoviera@dcs.fmph.uniba.sk

Abstract

The maximum genus $\gamma_M(G)$ of a graph G is the largest genus of an orientable surface into which G has a cellular embedding. Combinatorially, it coincides with the maximum number of disjoint pairs of adjacent edges of G whose removal results in a connected spanning subgraph of G. In this paper we describe a greedy 2-approximation algorithm for maximum genus by proving that removing pairs of adjacent edges from G arbitrarily while retaining connectedness leads to at least $\gamma_M(G)/2$ pairs of edges removed. As a consequence of our approach we also obtain a 2-approximate counterpart of Xuong's combinatorial characterisation of maximum genus.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms → Graph algorithms analysis, Mathematics of computing → Graph algorithms, Mathematics of computing → Graphs and surfaces

Keywords and phrases maximum genus, embedding, graph, greedy algorithm

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.14

Acknowledgements The authors would like to thank Rastislav Královič and Jana Višňovská for reading preliminary versions of this paper and making useful suggestions.

1 Introduction

One of the paradigms in topological graph theory is the study of all surface embeddings of a given graph. The *maximum genus* $\gamma_M(G)$ parameter of a graph G is then the maximum integer g such that G has a cellular embedding in the orientable surface of genus g. A result of Duke [12] implies that a graph G has a cellular embedding in the orientable surface of genus g if and only if $\gamma(G) \leq g \leq \gamma_M(G)$ where $\gamma(G)$ denotes the (minimum) genus of G. The basic problem of the area, namely the determination of the set of genera of orientable surfaces upon which G can be embedded, thus reduces to calculation of $\gamma(G)$ and $\gamma_M(G)$. Minimum genus, similarly to virtually all nonplanar topological graph invariants, is notoriously difficult. Its complexity remained open for more than 10 years after being included as one of the most prominent 12 open problems in the first edition of Garey and Johnson's book [15]. Eventually, Thomassen proved that it is NP-hard [48], even in the class of cubic graphs [50], but the problem is very difficult in practice [4]. For a related problem of Euler genus, a recent breakthrough by Kawarabayashi and Sidiropoulos [27] provided a $O(g^{256}\log^{189} n)$ approximation. Many other algorithms for topological invariants

[1] This work was supported in part by VEGA 1/0813/18 and by the Slovak Research and Development Agency under the contract No. APVV-15-0220.

are complex, error-prone, and very difficult to implement, see the discussion in [36] and [37]. On the other hand, maximum genus is considered rather well-understood mainly due to min-max characterisations [29, 52, 28, 38] the existence of polynomial-time algorithms [17, 13], and the fact that the problem is easy for 4-edge-connected graphs. However, the algorithm in [13] for the general case is essentially a reduction to optimum matching forest [16], another one of Garey and Johnson's 12 open problems. Optimum matching forest problem itself is a special case of linear matroid parity, the problem used in the original reduction in [13] and a common generalisation of matroid intersection and matching in general graphs. While linear matroid parity is polynomial [32], similarly to existing algorithms in topological graph theory, the algorithms for matroid parity are quite involved, not providing the desired intuition, insight, and ease of implementation. Consequently, there is a gap between the actual situation and the perceived status of maximum genus.

In this paper we show that classical edge-addition techniques – ideas dating back to Norhaus et al. [39] and Ringeisen [42] – can be used to efficiently approximate maximum genus, essentially reducing the problem to repeated connectivity testing. The resulting algorithm differs from most other algorithms for nonplanar topological invariants by combining favourable approximation ratio, conceptual simplicity, ease of implementation, and running time guarantees.

Related work

Maximum genus is a well-established [49, 45, 6, 7, 9, 5, 43, 24] and generalised problem [19, 21, 46, 47, 44, 1, 18, 20]. It is known that every 4-edge-connected graph has two edge-disjoint spanning trees [30] and Xuong's theorem ([52], Theorem 3 below) implies that it has an embedding with one or two faces, depending on the parity of the cycle rank. This determines the maximum genus exactly, rendering the problem trivial for 4-edge-connected graphs. However, this fact does not make the determination of maximum genus much easier for graphs that are not 4-edge-connected and even bounding the maximum genus remains relatively complicated, see for example [45, 6, 24, 2].

Building on characterisations of maximum genus, Furst et al. [13] and Glukhov [17] independently devised polynomial-time algorithms for determining the maximum genus of an arbitrary graph. The algorithm of [13] uses Xuong's characterisation of maximum genus and exploits a reduction to the cographic matroid parity on an auxiliary graph; its running time is bounded by $O(mn\Delta \log^6 m)$, where n, m, and Δ are the number of vertices, edges, and the maximum degree of the graph, respectively. As observed by Mohar and Thomassen [35], a reduction to a slightly less general problem of optimum matching forest is sufficient. However, optimum matching forest is a common generalisation of branching in directed graphs and matching in general graphs. While polynomial, the primal-dual algorithm solving optimum matching forest uses both the algorithms for optimum branchings and matchings in general graphs as subroutines [16]. For the linear matroid parity problem, itself rather difficult, Lovász [32, 34, 33] proved a min-max formula and derived a polynomial-time algorithm. To-date, the two fastest deterministic algorithms for the cographic [14], respectively linear [41], case of matroid parity are both (different) generalisations of the Edmonds' graph matching algorithm, and no significantly simpler algorithm is known. While the area is still active [31, 10, 25], it seems unlikely that the present level of sophistication can be avoided.

A matroidal structure is also in the background of the second algorithm for maximum genus derived in [17], albeit in a different way. Starting with any spanning tree T of G, the algorithm greedily finds a sequence of graphs F_i such that $T = F_0 \subseteq \cdots \subseteq F_n \subseteq G$, $|E(F_{i+1}) - E(F_i)| = 2$, and $\gamma_M(F_i) = i$ for all i, and $\gamma_M(F_n) = \gamma_M(G)$. However, the

extension from F_i to F_{i+1} is nontrivial; the underlying ideas of frame decompositions and recombinations can be seen in a slightly more general context of signed graphs in [44]. Consequently, it seems fair to say that the result is quite involved and inaccessible. The running time of this algorithm is bounded by $O(m^6)$.

A clever greedy approximation algorithm for the maximum genus of a graph was developed by Chen [5]. The algorithm has two main phases. First, it modifies a given graph G into a 3-regular graph H by vertex splitting, chooses an arbitrary spanning tree T of H, and finds a set P of disjoint pairs of adjacent edges in $H - E(T)$ with the maximum possible size. Second, it constructs a single-face embedding of $T \cup P$ and then inserts the remaining edges into the embedding while trying to raise the genus as much as possible. A doubly-linked face list data structure is maintained to support attempted edge insertions and to keep track of the embedding constructed. After all the edges had been inserted, a high-genus embedding of G in the same surface is constructed by contracting the edges created by vertex splitting. The algorithm constructs an embedding of G with genus at least $\gamma_M(G)/4$ and its running time $O(m \log n)$ is dominated by the second phase, that is, by operations on an embedded spanning subgraph of H. The disadvantages of the algorithm as presented in [5] are the need to actually work with the embedding, and the limited insight it provides into the relationship between the combinatorial structure of the decomposition and the genus of the resulting embedding. The first of these problems can be rectified and the whole process somewhat simplified by focusing on the value of the approximate maximum genus, as opposed to an embedding with such genus. The resulting algorithm then still requires vertex splitting into a 3-regular graph, constructing an optimal adjacency matching in a cotree, and examining the remaining components, and again yields only partial information about which edges of the graph eventually lead to an increase in the genus.

Several recent works deal with practical aspects of computations in topological graph theory [36, 4, 23, 37]. The common theme seems to be that from the practical perspective non-planar topological graph invariants are difficult to compute – for many problems there is either no suitable algorithm or the algorithms are hard to implement [36, 37]. For example, the project of determining all forbidden minors for torus is progressing very slowly due to the existing tools being insufficient, see [37]. In particular, the algorithm presented and used to find forbidden minors in [37] is conceptually simpler and faster in practice than the previous approaches, but has exponential running time. Even in the cases where there is a feasible approach, the inherent difficulty of the problem [4], or the size of the problem space [23] present significant obstacles. For example, it is not uncommon for a single graph on 10–20 vertices to require computation with length in days, although sparse graphs can be usually tackled reasonably well [23, 37].

Our terminology in the rest of the paper is standard and consistent with [35], in particular, $E(G)$ denotes the set of edges of a graph G, and the cycle rank $\beta(G)$ of a connected graph with n vertices and m edges is $\beta(G) = m - n + 1$. Additionaly, we use $\cup$ for set union and $G + e$ for the addition of an edge e to a graph G. For more details on embeddings we refer the reader to [22] or [35]; a recent survey of maximum genus can be found in [3, Chapter 2].

2 Edge-addition techniques and maximum genus

One of the earliest results on embeddings of graphs is the following observation, which is sometimes called Ringeisen's edge-addition lemma. Although it is implicit in [40], Ringeisen [42] was perhaps the first to draw an explicit attention to it.

▶ **Lemma 1.** *Let Π be an embedding of a connected graph G and let e be an edge not contained in G, but incident with vertices in G.*

(i) *If both ends of e are inserted into the same face of Π, then this face splits into two faces of the extended embedding of $G + e$ and the genus does not change.*

(ii) *If the ends of e are inserted into two distinct faces of Π, then in the extended embedding of $G + e$ these faces are merged into one and the genus raises by one.*

The next lemma, independently obtained in [29], [26], and [52], constitutes the cornerstone of proofs of Xuong's theorem. It follows easily from Lemma 1.

▶ **Lemma 2.** *Let G be a connected graph and $\{e, f\}$ a pair of adjacent edges not contained in G, but incident with vertices in G. If G has an embedding with a single face, then so does $G \cup \{e, f\}$.*

For a spanning tree T of G, let $\xi(G, T)$ denote the number of components of $G - E(T)$ with an odd number of edges.

▶ **Theorem 3** (Xuong's Theorem)**.** *Let G be a connected graph. Then*

$$\gamma_M(G) = (\beta(G) - \min_T \xi(G, T))/2$$

where the minimum is taken over all spanning trees of G.

It is not difficult to see that every cotree component with an even number of edges can be partitioned into pairs of adjacent edges, and that every cotree component with an odd number of edges can be partitioned into pairs of adjacent edges and one unpaired edge. Therefore, any spanning tree S minimising $\xi(G, S)$ maximises the number of pairs in the above partition of the cotree. The proof strategy of Xuong's theorem can now be summarised as follows. First, embed S in the 2-sphere arbitrarily. Then repeatedly apply Lemma 2 to pairs obtained from the partition of the components of $G - E(S)$, each time raising the genus by one. Finally, add the remaining edges. Lemma 1 guarantees that the addition of the remaining edges cannot lower the genus. The result of this process is an embedding of G with genus at least $(\beta(G) - \min_T \xi(G, T))/2$.

The fact that a spanning tree minimising $\xi(G, T)$ maximises the number of pairs of adjacent edges in the cotree suggests a slightly different, yet essentially equivalent combinatorial characterisation of maximum genus. It is due to Khomenko et al. [29] and in fact is older than Xuong's theorem itself.

▶ **Theorem 4.** *The maximum genus of a connected graph equals the maximum number of disjoint pairs of adjacent edges whose removal leaves a connected graph.*

The following useful lemma, found for example in [8], is an extension of Lemma 2 to embeddings with more than one face. It can either be proved directly by using Ringeisen's edge-adding technique or can be derived from Xuongs's theorem. The lemma was used in [8] to devise an algorithm that constructs an embedding of genus $\gamma_M(G) - 1$ whenever such an embedding exists (cf. Lemma 4.3 of [5]).

▶ **Lemma 5.** *Let G be a connected graph and $\{e, f\}$ a pair of adjacent edges not contained in G, but incident with vertices in G. Then $\gamma_M(G \cup \{e, f\}) \geq \gamma_M(G) + 1$.*

Our algorithm is based on the rather obvious, but never fully exploited fact that Lemma 5 can be applied to sets of pairs of adjacent edges which do not necessarily have the maximum possible size. Indeed, if we find any k pairs of adjacent edges $(e_i, f_i)_{i=1}^k$ in a graph G such

that $G - \bigcup_{i=1}^{k} \{e_i, f_i\}$ is connected, then by Lemma 5 we can assert that the maximum genus of G is at least k. This suggests that identifying a large number of pairs of adjacent edges whose removal leaves a connected subgraph can be utilised to obtain a simple approximation algorithm for the maximum genus. Indeed, in the following section we show that choosing the pairs of adjacent edges arbitrarily yields a 2-approximation of maximum genus.

3 The Algorithm

In this section we present a greedy algorithm for finding at least $\gamma_M(G)/2$ pairs of adjacent edges while the rest of the graph remains connected. The idea is simple: if the removal of a pair of adjacent edges does not disconnect the graph, then we remove it.

Algorithm 1: Greedy-Max-Genus

Input : Connected graph G
Output : Set P of paiwise disjoint pairs of adjacent edges of G such that $G - P$ is a connected spanning subgraph of G

```
1  H ← G
2  P ← ∅
3  foreach pair of adjacent edges e, f from H do
4      if H − {e, f} is connected then
5          H ← H − {e, f}
6          P ← P ∪ (e, f)
7  return P
```

To prove that the set output by Greedy-Max-Genus Algorithm always contains at least $\gamma_M(G)/2$ pairs of adjacent edges we employ the following lemma, which can be easily proved either using Xuong's theorem or directly from Lemma 1.

▶ **Lemma 6.** *Let G be a connected graph and let e be an arbitrary edge of G such that $G - e$ is connected. Then*

$$\gamma_M(G) - 1 \leq \gamma_M(G - e) \leq \gamma_M(G).$$

The final ingredient is the following characterisation of graphs with maximum genus 0.

▶ **Theorem 7.** *The following statements are equivalent for every connected graph G.*

(i) $\gamma_M(G) = 0$
(ii) *No two cycles of G have a vertex in common.*
(iii) *G contains no pair of adjacent edges whose removal leaves a connected graph.*

The equivalence (i) ⇔ (ii) in Theorem 7 was first proved by Nordhaus et al. in [39]. The equivalence (ii) ⇔ (iii) is easy to see, nevertheless it is its appropriate combination with Lemma 6 which yields the desired performance guarantee for Greedy-Max-Genus algorithm, as shown in the following theorem.

▶ **Theorem 8.** *For every connected graph G, the set of pairs output by Greedy-Max-Genus Algorithm run on G contains at least $\gamma_M(G)/2$ pairs of adjacent edges.*

Proof. Assume that the algorithm stops after the removal of k disjoint pairs of adjacent edges from G. For $i \in \{0, 1, \ldots, k\}$ let H_i denote the graph obtained from G by the removal of the the first i pairs of edges. By Lemma 6, the removal of a single edge from a graph can

lower its maximum genus by at most one. Therefore, the removal of two edges can lower the maximum genus by at most two. It follows that $\gamma_M(H_i) \geq \gamma_M(G) - 2i$ for each i; in particular, $\gamma_M(H_k) \geq \gamma_M(G) - 2k$. From Theorem 7 we get that $\gamma_M(H_k) = 0$. By combining these expressions we get $2k \geq \gamma_M(G)$, which yields $k \geq \gamma_M(G)/2$, as desired. ◀

Clearly, any maximal set of pairs of adjacent edges of G whose removal from G yields a connected graph can possibly be the output of Greedy-Max-Genus Algorithm run on G. Hence, as a corollary of Theorem 8 we obtain the following 2-approximate counterpart of Theorem 4.

▶ **Theorem 9.** *Let G be a connected graph and let P be any inclusion-wise maximal set of disjoint pairs of adjacent edges of G whose removal leaves a connected subgraph. Then $|P| \geq \gamma_M(G)/2$.*

4 Notes

Considering a star $K_{1,2n}$ with every edge doubled and a loop attached to every pendant vertex shows that the bound in Theorems 8 and 9 is best possible. At the same time, the last example shows that processing vertices in the decreasing order with respect to their degrees does not lead to an algorithm with better approximation ratio.

The simplest realisation of the algorithm considers all pairs of edges $\{e, f\}$ with a common end-vertex and test whether removing the pair does not disconnect the graph. The running time is $O((\tau + \rho)\sum_{i=1}^{n} d_i^2)$, where τ is the time required to test the connectivity, ρ is the time required to update the underlying data structure, and d_i is the degree of the i-th vertex. If the input graph is simple, then $\sum_{i=1}^{n} d_i^2 = O(m^2/n)$ by [11]. If the input graph is not simple, it can be preprocessed by including in the solution a pair of adjacent parallel edges and/or loops and keeping at least one edge from each set of parallel edges until there are no sets of more than two parallel edges and no adjacent loops. It is easy to see that the resulting graph again satisfies $\sum_{i=1}^{n} d_i^2 = O(m^2/n)$. Therefore, using a linear-time connectivity test the running time is $O(m^3/n)$. Trading simplicity for running time and using algorithms for dynamic graph connectivity [51] it is possible to support updates in $\rho = O(\log^2 n/\log\log n)$ amortized time and queries in $\tau = O(\log n/\log\log n)$ worst-case time. This would yield running time $O(m^2 \log^2 n/(n \log\log n))$.

In our opinion, obtaining a simple algorithm (i. e. one notably simpler than the tools used in the exact algorithms) with approximation ratio better than 2 would be a significant achievement.

References

1 D. Archdeacon, C. P. Bonnington, and J. Širáň. A Nebeský-Type Characterization for Relative Maximum Genus. *J. Combin. Theory Ser. B*, 73(1):77–98, 1998.

2 D. Archdeacon, M. Kotrbčík, R. Nedela, and M. Škoviera. Maximum genus, connectivity, and Nebeský's Theorem. *Ars Math. Contemp.*, 9:51–61, 2015.

3 L. W. Beineke, R. J. Wilson, J. L. Gross, and T. W. Tucker, editors. *Topics in Topological Graph Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.

4 S. Beyer, M. Chimani, I. Hedtke, and M. Kotrbčík. A Practical Method for the Minimum Genus of a Graph: Models and Experiments. In A. V. Goldberg and A. S. Kulikov, editors, *Proceedings of 15th International Symposium on Experimental Algorithms SEA'16*, pages 75–88. Springer International Publishing, 2016.

5 J. Chen. Algorithmic graph embeddings. *Theoret. Comput. Sci.*, 181:247–266, 1997.

6 J. Chen, D. Archdeacon, and J. L. Gross. Maximum genus and connectivity. *Discrete Math.*, 149:19–29, 1996.

7 J. Chen, S. P. Kanchi, and A. Kanevsky. On the Complexity of Graph Embeddings (extended abstract). In F. Dehme et. al., editor, *Algorithms and Data Structures WADS'93*, volume 709 of *Lecture Notes in Comp. Sci.*, pages 234–245. Springer-Verlag, Berlin, 1993.

8 J. Chen, S. P. Kanchi, and A. Kanevsky. On the Complexity of Graph Embeddings (extended abstract). In F. Dehme et al., editor, *Algorithms and Data Structures*, volume 709 of *Lecture Notes in Comp. Sci.*, pages 234–245. Springer-Verlag, Berlin, 1993.

9 J. Chen, S. P. Kanchi, and A. Kanevsky. A note on approximating graph genus. *Inform. Process. Lett.*, 6(61):317–322, 1997.

10 H. Y. Cheung, L. C. Lau, and K. M. Leung. Algebraic Algorithms for Linear Matroid Parity Problems. *ACM Trans. Algorithms*, 10(3):10:1–10:26, 2014.

11 D. de Caen. An upper bound on the sum of squares of degrees in a graph. *Discrete Math.*, 185(1–3):245–248, 1998.

12 R. A. Duke. The genus, regional number, and the Betti number of a graph. *Canad. J. Math.*, 18:817–822, 1966.

13 M. L. Furst, J. L. Gross, and L. A. McGeoch. Finding a maximum-genus embedding. *J. ACM*, 35:523–534, 1988.

14 H. M. Gabow and M. Stallmann. Effecient Algorithms for Graphic Matroid Intersection and Parity (extended abstract). In *Automata, Languages and Programming ICALP'85*, volume 194 of *LNCS*, pages 210–220. Springer-Verlag, Berlin, 1985.

15 M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the theory of NP-completeness*. Bell Telephone Laboratories, 1979.

16 R. Giles. Optimum Matching Forests I: Special Weights. *Math. Programming*, 22:1–12, 1982.

17 A. D. Glukhov. A contribution to the theory of maximum genus of a graph. In *Structure and Topological Properties of Graphs*, pages 15–29. Inst. Mat. Akad. Nauk Ukrain. SSR, Kiev, 1981. In Russian.

18 J. L. Gross. Embeddings of graphs of fixed treewidth and bounded degree. *Ars Math. Contemp.*, 7(2):379–403, 2014.

19 J. L. Gross and M. L. Furst. Hierarchy for imbedding-distribution invariants of a graph. *J. Graph Theory*, 11(2):205–220, 1987.

20 J. L. Gross, T. Mansour, T. W. Tucker, and D. Wang. Log-Concavity of Combinations of Sequences and Applications to Genus Distributions. *SIAM J. Discrete Math.*, 29(2):1002–1029, 2015.

21 J. L. Gross and R. G. Rieper. Local extrema in genus-stratified graphs. *J. Graph Theory*, 15:159–171, 1991.

22 J. L. Gross and T. W. Tucker. *Topological Graph Theory*. Wiley, 1987.

23 M. Hellmuth, A. S. Knudsen, M. Kotrbčík, D. Merkle, and N. Nøjgaard. Linear Time Canonicalization and Enumeration of Non-Isomorphic 1-Face Embeddings. In R. Pagh and S. Venkatasubramanian, editors, *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments ALENEX'18*, pages 154–168. Society for Industrial and Applied Mathematics, 2018.

24 Y. Huang. Maximum genus and chromatic number of graphs. *Discrete Math.*, 271(1):117–127, 2003.

25 S. Iwata and Y. Kobayashi. A Weighted Linear Matroid Parity Algorithm. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing STOC'17*, pages 264–276. ACM, 2017.

26 M. Jungerman. A charactrerization of upper embeddable graphs. *Trans. Amer. Math. Soc.*, 241:401–406, 1978.

27 K. Kawarabayashi and A. Sidiropoulos. Beyond the Euler characteristic: Approximating the genus of general graphs. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing STOC'15*, pages 675–682. ACM, 2015.

28 N. P. Khomenko and A. D. Glukhov. Single-component 2-cell embeddings and maximum genus of a graph. In *Some topological and combinatorial properties of graphs*, pages 5–23. Inst. Mat. Akad. Nauk Ukrain. SSR, Kiev, 1980. In Russian.

29 N. P. Khomenko, N. A. Ostroverkhy, and V. A. Kuzmenko. The maximum genus of a graph. In *ϕ-Transformations of Graphs*, pages 180–207. Inst. Mat. Akad. Nauk Ukrain. SSR, Kiev, 1973. In Ukrainian with English summary.

30 S. Kundu. Bounds on Number of Disjoint Spanning Trees. *J. Combinatorial Theory Ser. B*, 17:199–203, 1974.

31 J. Lee, M. Sviridenko, and J. Vondrák. Matroid Matching: The Power of Local Search. *SIAM J. Computing*, 42(1):357–379, 2013.

32 L. Lovász. The matroid matching problem. In *Algebraic methods in Graph Theory*, volume 25 of *Colloquia Mathematica Soc. János Bolyai*, pages 495–517. North-Holland, Amsterdam, 1978.

33 L. Lovász. Matroid matching and some applications. *J. Combin. Theory Ser. B*, 28(2):208–236, 1980.

34 L. Lovász. Selecting independent lines from a family of lines in a space. *Acta Sci. Math.*, 42:121–131, 1980.

35 B. Mohar and C. Thomassen. *Graphs on Surfaces*. The Johns Hopkins University Press, 2001.

36 W. Myrvold and W. Kocay. Errors in graph embedding algorithms. *J. Comp. System Sci.*, 77(2):430–438, 2011.

37 W. Myrvold and J. Woodcock. A Large Set of Torus Obstructions and How They Were Discovered. *Electronic J. Combin.*, 25(1):♯P1.16, 2018.

38 L. Nebeský. A new characterization of the maximum genus of a graph. *Czechoslovak Math. J.*, 31(106):604–613, 1981.

39 E. A. Nordhaus, R. D. Ringeisen, B. M. Stewart, and A. T. White. A Kuratowski-type theorem for the maximum genus of a graph. *J. Combin. Theory Ser. B*, 12:260–267, 1972.

40 E. A. Nordhaus, B. M. Stewart, and A. T. White. On the maximum genus of a graph. *J. Combin. Theory Ser. B*, 11(3):258–267, 1971.

41 J. Orlin. A Fast, Simpler Algorithm for the Matroid Parity Problem. In A. Lodi, A. Panconesi, and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, volume 5035 of *Lecture notes in comp. sci.*, pages 240–258. Springer-Verlag, 2008.

42 R. D. Ringeisen. Determining all compact orientable 2-manifolds upon which $K_{m,n}$ has 2-cell imbeddings. *J. Combin. Theory Ser. B*, 12(2):101–104, 1972.

43 R. D. Ringeisen. Survey of results on the maximum genus of a graph. *J. Graph Theory*, 3:1–13, 1979.

44 J. Širáň and M. Škoviera. Characterization of the maximum genus of a signed graph. *J. Combinatorial Theory ser. B*, 52:124–146, 1991.

45 M. Škoviera. The maximum genus of graphs of diameter two. *Discrete Math.*, 52:124–146, 1991.

46 S. Stahl. On the Number of Maximum Genus Embeddings of Almost all Graphs. *European J. Combin.*, 13:119–126, 1992.

47 S. Stahl. On the average genus of the random graph. *J. Graph Theory*, 20(1):1–18, 1995.

48 C. Thomassen. The Graph Genus Problem is NP-Complete. *J. Algorithms*, 10:568–576, 1989.

49 C. Thomassen. Bidirectional retracting-free double tracings and upper embeddability of graphs. *J. Combinatorial Theory ser. B*, 50(2):198–207, 1990.

50 C. Thomassen. The Genus Problem for Cubic Graphs. *J. Combin. Theory Ser. B*, 69(1):52–58, 1997.

51 C. Wulff-Nilsen. Faster Deterministic Fully-dynamic Graph Connectivity. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms SODA'13*, pages 1757–1769. Society for Industrial and Applied Mathematics, 2013.

52 N. H. Xuong. How to determine the maximum genus of a graph. *J. Combin. Theory Ser. B*, 26:217–225, 1979.

A Note on Max k-Vertex Cover: Faster FPT-AS, Smaller Approximate Kernel and Improved Approximation

Pasin Manurangsi[1]
University of California, Berkeley, CA, USA
pasin@berkeley.edu

Abstract

In Maximum k-Vertex Cover (Max k-VC), the input is an edge-weighted graph G and an integer k, and the goal is to find a subset S of k vertices that maximizes the total weight of edges covered by S. Here we say that an edge is covered by S iff at least one of its endpoints lies in S.

We present an FPT approximation scheme (FPT-AS) that runs in $(1/\varepsilon)^{O(k)}\mathsf{poly}(n)$ time for the problem, which improves upon Gupta, Lee and Li's $(k/\varepsilon)^{O(k)}\mathsf{poly}(n)$-time FPT-AS [30, 29]. Our algorithm is simple: just use brute force to find the best k-vertex subset among the $O(k/\varepsilon)$ vertices with maximum weighted degrees.

Our algorithm naturally yields an (efficient) approximate kernelization scheme of $O(k/\varepsilon)$ vertices; previously, an $O(k^5/\varepsilon^2)$-vertex approximate kernel is only known for the unweighted version of Max k-VC [43]. Interestingly, this also has an application outside of parameterized complexity: using our approximate kernelization as a preprocessing step, we can directly apply Raghavendra and Tan's SDP-based algorithm for 2SAT with cardinality constraint [52] to give an 0.92-approximation algorithm for Max k-VC in polynomial time. This improves upon the best known polynomial time approximation algorithm of Feige and Langberg [23] which yields $(0.75 + \delta)$-approximation for some (small and unspecified) constant $\delta > 0$.

We also consider the minimization version of the problem (called Min k-VC), where the goal is to find a set S of k vertices that minimizes the total weight of edges covered by S. We provide a FPT-AS for Min k-VC with similar running time of $(1/\varepsilon)^{O(k)}\mathsf{poly}(n)$. Once again, this improves on a $(k/\varepsilon)^{O(k)}\mathsf{poly}(n)$-time FPT-AS of Gupta et al. On the other hand, we show, assuming a variant of the Small Set Expansion Hypothesis [50] and $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, that there is no polynomial size approximate kernelization for Min k-VC for any factor less than two.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Mathematics of computing → Approximation algorithms

Keywords and phrases Maximum k-Vertex Cover, Minimum k-Vertex Cover, Approximation Algorithms, Fixed Parameter Algorithms, Approximate Kernelization

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.15

Related Version A full version of the paper is available at [44], `https://arxiv.org/abs/1810.03792`.

1 Introduction

In the *Vertex Cover* problem, we are given a graph G and an integer k, and the goal is to determine whether there is a set S of k vertices that covers all the edges, where the edge is said to be covered by S if at least one of its endpoints lies in S. Vertex Cover is a classic

[1] Supported by NSF under Grants No. CCF 1655215 and CCF 1815434.

graph problem and is among Karp's original list of 21 NP-complete problems [37]. This NP-hardness has led to studies of variants of the problems. One such direction is to consider the optimization versions of the problem. Arguably, the two most natural optimization formulations of Vertex Cover are the *Minimum Vertex Cover (Min VC)* problem, where the constraint that every edge is covered is treated as a hard constraint and the goal is to find S with smallest size that satisfies this, and the *Maximum k-Vertex Cover (Max k-VC)* problem[2], where the cardinality constraint $|S| = k$ is treated as a hard constraint and the goal is to find such S that covers as many edges as possible.

Both problems have been thoroughly studied in the approximation algorithms and hardness of approximation literature. Min VC admits a simple greedy 2-approximation algorithm[3], which has been known since the seventies (see e.g. [26]). The approximation ratio has subsequently been slightly improved [9, 47] and, currently, the best known approximation ratio in polynomial time is $(2 - 1/O(\sqrt{\log n}))$ [36]. There has also been a number of works on hardness of approximation of Vertex Cover [11, 34, 22, 41, 8, 39, 40]. The best known NP-hardness of approximation for Min VC, established in the recent works that resolve the (imperfect) 2-to-1 conjecture [39, 20, 21, 40], has a factor of $(\sqrt{2}-\varepsilon)$ for any $\varepsilon > 0$. Assuming the Unique Games Conjecture (UGC) [38], the inapproximability ratio can be improved to $(2 - \varepsilon)$ for any $\varepsilon > 0$ [41, 8], which is tight up to lower order terms.

Unlike Min VC, tight approximability results for Max k-VC are not known (even assuming UGC). In particular, on the algorithmic front, the best known efficient approximation algorithm due to Feige and Langberg [23] yields a $(0.75 + \delta)$-approximation for the problem, where $\delta > 0$ is a (small) constant. This was an improvement over an earlier 0.75-approximation algorithm of Ageev and Sviridenko [2], which in turn improved upon the simple greedy algorithm that yields $(1 - 1/e)$-approximation for the problem [35]. (See also [33, 32, 31] where improvements have been made for certain ranges of k and n.) On the hardness of approximation front, it is known that the problem is NP-hard to approximate to within $(1 + \delta)$ factor for some (small) $\delta > 0$ [49]. Moreover, it follows from a result of Austrin, Khot and Safra [7] that it is UG-hard to approximate the problem to within a factor of 0.944. (See Appendix A of the full version [44].) This leaves quite a large gap between the upper and lower bounds, even assuming the UGC.

Approximability is not the only aspect of Vertex Cover and its variants that has been thoroughly explored: its parameterized complexity is also a well-studied subject. Recall that an algorithm is said to be *fixed-parameter (FPT)* with respect to parameter k if it runs in time $f(k) \cdot \mathsf{poly}(n)$ for some function f, where n is the size of the input. An FPT algorithm (with running time $k^{O(k)} \cdot \mathsf{poly}(n)$) was first devised for Vertex Cover by Buss and Goldsmith [14]. Since then, many different FPT algorithms have been discovered for Vertex Cover; to the best of our knowledge, the fastest known algorithm is that of Chen, Kanj and Xia [17], which runs in $1.2738^k \cdot \mathsf{poly}(n)$ time.

Notice that an FPT algorithm for Vertex Cover can also be adapted to solve Min VC in FPT time parameterized by the optimal solution size, by running the Vertex Cover algorithm for $k = 1, 2, \ldots$ until it finds the size of the optimal solution. On the other hand, Max k-VC

[2] Max k-VC and Min k-VC (which will be introduced below) are sometimes referred to as the Max Partial Vertex Cover and Min Partial Vertex Cover respectively. However, we decide against calling them as such to avoid ambiguity since Partial Vertex Cover has also used to refer to a different variant of Vertex Cover (see e.g. [10]).

[3] Throughout this note, we use the convention that the approximation ratio is the worst case ratio between the cost of the output solution and the optimum. In other words, the approximation ratios for maximization problems will be at most one, whereas the approximation ratios for minimization problems will be at least one.

is unlikely to admit an FPT algorithm, as it is W[1]-hard [28]. Circumventing this hardness, Marx [46] designed an *FPT approximation scheme (FPT-AS)*, which is an FPT algorithm that can achieve approximation ratio $(1-\varepsilon)$ (or $(1+\varepsilon)$ for minimization problems) for any $\varepsilon > 0$, for Max k-VC. In particular, his algorithm runs in time $(k/\varepsilon)^{O(k^3/\varepsilon)} \cdot \mathsf{poly}(n)$. This should be contrasted with the aforementioned fact that Max k-VC does not admit a PTAS unless P = NP. Recently, the FPT-AS has been sped up by Gupta, Lee and Li [30, 29][4] to run in time $(k/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$.

FPT algorithms are intimately connected to the notion of kernel. A *kernelization algorithm* (or *kernel*) of a parameterized problem is a polynomial time algorithm that, given an instance (I, k), produces another instance (I', k') such that the size of the new instance $|I'|$ and the new parameter k' are both bounded by $g(k)$ for some function g. It is well known that a parameterized problem admits a kernel if and only if it admits FPT algorithms [15]. Once again, many kernels are known for Vertex Cover (see e.g. [1] and references therein). On the other hand, the W[1]-hardness of Max k-VC means that it does not admit a kernel unless W[1] = FPT.

Recently, there have been attempts to make the concept of kernelization compatible with approximation algorithms [24, 43]. In this note, we follow the notations defined by Lokshtanov et al. [43]. For our purpose, it suffices to define an *α-approximate kernel* for an parameterized optimization problem as a pair of polynomial time algorithms $\mathcal{A}$, the *reduction algorithm*, and $\mathcal{B}$, the *solution lifting algorithm*, such that (i) given an instance (I, k), $\mathcal{A}$ produces another instance (I', k') such that $|I'|, k'$ are bounded by $g(k)$ for some g and (ii) given an β-approximate solution s' for (I', k'), $\mathcal{B}$ produces a solution s of (I, k) such that s is an $(\alpha\beta)$-approximate solution[5] for (I, k). Akin to (exact) kernelization, Lokshtanov et al. [43] shows that a decidable parameterized optimization problem admits α-approximate kernel if and only if it admits an FPT α-approximation algorithm. (We refer interested readers to Section 2.1 of [43] for more details.) In light of Marx's algorithm for Max k-VC [46], this immediately implies that Max k-VC admits $(1-\varepsilon)$-approximate kernel for any $\varepsilon > 0$. Lokshtanov et al. [43] made this bound more specific, by showing that the insights from Marx's work can be turned into an $(1-\varepsilon)$-approximate kernel where the number of vertices in the new instance is at most $O(k^5/\varepsilon^2)$.

Minimum k-Vertex Cover. We will also consider the minimization variant of the Min k-VC, which we call *Minimum k-Vertex Cover (Min k-VC)*. The goal of this problem is to find a subset of k vertices that *minimizes* the number of edges covered. Note that this is *not* a natural relaxation of Vertex Cover and is in fact more closely related to edge expansion problems. (See [25] and discussion therein for more information.) The greedy algorithm that picks k vertices with minimum degrees yields a 2-approximation. Gandhi and Kortsarz [25] showed that this is likely tight: assuming the Small Set Expansion Conjecture [50], it is hard to approximate Min k-VC to within $(2-\varepsilon)$ factor for any $\varepsilon > 0$. As for its parameterized complexity, similar to Max k-VC, Min k-VC is W[1]-hard [28] and admits an FPT-AS with running time $(k/\varepsilon)^{O(k)}$ [30, 29].

[4] In fact, Gupta et al.gives an FPT-AS for Min k-VC; it is trivial to see that their algorithm works for Max k-VC as well.

[5] We use a similar convention here as our convention for approximation ratios. That is, $\alpha \leqslant 1$ for maximization problems and $\alpha \geqslant 1$ for minimization problems. Note that this is not the same as in [43] where $\alpha \geqslant 1$ in both cases; nevertheless, it is simple to see that these different conventions do not effect any of the results.

Weight vs Unweighted. All results stated above are for unweighted graphs. The natural extensions of Max k-VC (resp. Min k-VC) to edge-weighted graphs ask to find subsets of vertices of size k that maximizes (resp. minimizes) the total weight of the edges covered. To avoid confusion, we refer to these weighted variants explicitly as Weighed Max k-VC and Weight Min k-VC. Clearly, since these problems are more general than the unweighted ones, the lower bounds above (including inapproximability results and W[1]-hardness) applies immediately. It is also quite simple to check that all aforementioned polynomial time approximation algorithms for the unweighted case extends naturally to the weighted setting too. The FPT-ASes are slightly trickier, but Gupta et al. [30] provide an argument discretizing the weights and extend their FPT-ASes to the weighted case with similar time complexity. It is also possible to apply this argument to Lokshtanov et al.'s [43] approximate kernel, although it would result in a graph of $O(k^7/\varepsilon^4)$ vertices instead of $O(k^5/\varepsilon^2)$ for the unweighted case.

1.1 Our Results

For convenience, all our results stated below are for the weighted version of the problems, and moreover we allow self-loops in the input graph. This is the most general version of the problem and, hence, the algorithmic results below apply directly to the unweighted case (nd the weighted simple graph case. We also note that this choice is partly motivated by the fact that in some applications, such Gupta et al.'s [30, 29] algorithms for Minimum k-Cut, this full generality is needed. (Unfortunately, our result does not imply faster algorithms for Minimum k-Cut, as the bottlenecks of Gupta et al.'s approach is elsewhere[6].)

We remark that, while the algorithmic results apply directly to the more restricted version, the approximate kernel does *not*. This is because, in a more restricted version (e.g. unweighted) of the problems, the instance output by the reduction algorithm is also more restrictive (e.g. unweighted), meaning that one cannot simply use the approximate kernel for the more general version. Nevertheless, as we will point out below, our approximate kernel also extends to the unweighted setting (and simple graph setting), with a small loss in parameter.

Maximum k-Vertex Cover

Our first result is a faster FPT-AS for Max k-VC that runs in time $O(1/\varepsilon)^k \cdot \mathsf{poly}(n)$, which improves upon a $(k/\varepsilon)^k \cdot \mathsf{poly}(n)$-time FPT-AS due to Gupta, Lee and Li [29].

▶ **Theorem 1.** *For every $\varepsilon > 0$, there exists an $(1-\varepsilon)$-approximation algorithm for Weighted Max k-VC that runs in time $O(1/\varepsilon)^k \cdot \mathsf{poly}(n)$.*

Perhaps more importantly, our FPT-AS is simple and yields a new insight compared to the previous FPT-ASes [46, 30, 29]. In particular, our algorithm is just the following: restrict ourselves only to the $O(k/\varepsilon)$ vertices with maximum weighted degrees and use brute force to find a k-vertex subset among these vertices that cover edges with maximum total weight.

To demonstrate the differences to the previous algorithms, let us briefly sketch how they work here. The known FPT-ASes [46, 30, 29] all rely on a degree-based argument for the unweighted case due to Marx [46] who consider the following two cases:

[6] For their FPT approximation algorithm [30], the bottleneck is in the reduction from Min k-Cut to Laminar k-Cut which runs in time $2^{O(k^2)} \cdot \mathsf{poly}(n)$. For their $(1+\varepsilon)$-approximation algorithm [29], the bottleneck is in the dynamic programming step which takes $(k/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$ time.

1. The vertex with maximum degree have degree at least k^2/ε. In this case, one can simply take the k vertices with largest degree because the number of edges with both endpoints in the set is at most $\binom{k}{2}$, meaning that it only affects the number of edges covered by at most an ε factor and thus this is already an $(1-\varepsilon)$-approximation for the problem.
2. The vertex with maximum degree have degree at most k^2/ε. The key property in this case is that the number of edges covered by the optimal solution is at most k^3/ε, which is bounded by a function of k. Marx's algorithm then proceeds as follows: (i) guess the number of edges $\ell \leqslant k^3/\varepsilon$ in the optimal solution, (ii) guess (among the k^ℓ possibilities) which vertex (in the solution) that each edge is covered by, (iii) randomly color each edge in the input graph with one of ℓ colors and randomly color each vertex with one of k colors and (iv) finally, determine whether there are k vertices each of different color that covers edges with colors as guessed in Step (ii). Note that Step (iv) can be easily done in polynomial time. Since ℓ is bounded by k^3/ε, the algorithm succeeds with probability at least $k^{-O(k^3/\varepsilon)}$, which can be turned into a randomized algorithm with running time $k^{-O(k^3/\varepsilon)} \cdot \mathsf{poly}(n)$ that succeeds with high probability. Finally, it can be derandomized using standard techniques (see [3]).

The speed-up of Gupta et al. [30, 29] comes from the change in the second case. Roughly speaking, they show that more elaborated coloring techniques can be used, in conjunction with dynamic programming, to speed the second case up to $(k/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$.

Intuitively, our result shows that this case-based analysis is in fact not needed, as it suffices to consider the $O(k/\varepsilon)$ vertices with highest weighted degrees. Moreover, a nice feature about our algorithm is that it works naturally for the weighted case, whereas Gupta et al. needs to employ a discretization argument to deal with this case. (See Section 5.2 in the full version of [30].)

Another feature of our algorithm is that it immediately gives an approximate kernelization for the problem, by restricting to the subgraph induced by the $O(k/\varepsilon)$ vertices and adding self-loops with appropriate weights to compensate the edges from these vertices to the remaining vertices. This results in an $(1-\varepsilon)$-approximate kernelization of $O(k/\varepsilon)$ vertices for Max k-VC:

▶ **Lemma 2.** *For every $\varepsilon > 0$, Weighted Max k-VC admits an $(1-\varepsilon)$-approximate kernelization with $O(k/\varepsilon)$ vertices.*

As stated earlier, the above result is not directly comparable to Lokshtanov et al.'s [43] approximate kernel of $O(k^5/\varepsilon^2)$ vertices for the unweighted version of Max k-VC. Fortunately, our technique also gives an $O(k/\varepsilon^2)$-vertex approximate kernel for the unweighted case, which indeed improves upon Lokshtanov et al.'s result. (See the end of Section 3.2.)

Interestingly, the above approximate kernelization also has an application outside of parameterized complexity: using our approximate kernelization as a preprocessing step, we can directly apply Raghavendra and Tan's SDP-based algorithm for 2SAT with cardinality constraint [52] to give an 0.92-approximation algorithm for Max k-VC in polynomial time. This improves upon the aforementioned polynomial time approximation algorithm of Feige and Langberg [23] which yields $(0.75+\delta)$-approximation for some (small and unspecified) constant $\delta > 0$.

▶ **Corollary 3.** *There exists a polynomial time 0.92-approximation algorithm for Weighted Max k-VC.*

We note here that the approximation guarantee above is even better than the previous best known ratios for some special cases, such as in bipartite graph [4, 13] where the previous best known approximation ratio is 0.821 [13].

Minimum k-Vertex Cover

For the Weighted Min k-VC problem, we give a FPT-AS with similar running time of $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$ for the problem. Once again, this improves upon the $(k/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$-time algorithm of Gupta et al. [30, 29].

▶ **Theorem 4.** *For every $\varepsilon > 0$, there exists an $(1+\varepsilon)$-approximation algorithm for Weighted Min k-VC that runs in time $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$.*

We remark that this algorithm is different from the algorithm for Max k-VC and is instead based on a careful branch-and-bound approach. A natural question here is perhaps whether this difference is inherent. While it is unclear how to make this question precise, we provide an evidence that the two problems are indeed of different natures by showing that, in contrast to Max k-VC, a polynomial size approximate kernelization for Min k-VC for any factor less than two is unlikely to exist:

▶ **Lemma 5.** *Assuming the Strong Small Set Expansion Hypothesis (Conjecture 16) and $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, Weighted Min k-VC does not admit a polynomial size $(2-\varepsilon)$-approximate kernelization for any $\varepsilon \in (0, 1]$.*

The above result is under a variant of the Small Set Expansion Hypothesis [50]; please refer to Section 4.2 for the precise definition of the variant. We also note that the above lower bound also applies to the unweighted version; again please see Section 4.2 for more details.

2 Notations

Throughout this note, we think of an edge-weighted graph as a complete graph (self-loops included) where each edge is endowed with a non-negative weight. More specifically, an edge-weighted graph G consists of a vertex set V_G and a weight function $w_G : \binom{V_G}{\leqslant 2} \to \mathbb{R}_{\geqslant 0}$. (Note that, for a set U and a non-negative integer ℓ, we use $\binom{U}{\leqslant \ell}$ and $\binom{U}{\ell}$ to denote the collections of subsets of U of sizes at most ℓ and exactly ℓ respectively.) When the graph is clear from the context, we may drop the subscript G, and we sometimes use w_e to denote $w(e)$ for brevity. For each vertex $v \in V$, we use w-deg(v) to denote its weighted degree, i.e., w-deg$(v) = \sum_{e \in \binom{V_G}{\leqslant 2}, v \in e} w_e$. For a subset $S \subseteq V_G$, we write w-deg(S) to denote $\sum_{v \in S}$ w-deg(v). For subsets $S, T \subseteq V_G$, we use $E_G(S, T)$ to denote the total weight of edges with at least one endpoint in S and at least one endpoint in T; more specifically, $E_G(S, T) = \sum_{e \in \binom{V_G}{\leqslant 2}, e \cap S \neq \emptyset, e \cap T \neq \emptyset} w_G(e)$. Note that $E_G(S, S)$ is the total weight of the edges covered by S; for brevity, we use $E_G(S)$ as a shorthand for $E_G(S, S)$. Finally, we use $\mathrm{OPT}_{\text{Min k-VC}}(G, k)$ and $\mathrm{OPT}_{\text{Max k-VC}}(G, k)$ to denote the optimums of Min k-VC and Max k-VC respectively on the instance (G, k). More formally, $\mathrm{OPT}_{\text{Min k-VC}}(G, k) = \min_{S \in \binom{V_G}{k}} E_G(S)$ and $\mathrm{OPT}_{\text{Max k-VC}}(G, k) = \max_{S \in \binom{V_G}{k}} E_G(S)$.

3 Maximum k-Vertex Cover

We will now prove our results for Max k-VC. To do so, it will be convenient to order the vertices of the input graph V_G based on their weighted degree (ties broken arbitrarily), i.e., let $v_1, \ldots, v_n$ be the ordering of vertices in V_G such that w-deg$_G(v_1) \geqslant \cdots \geqslant \cdot$ w-deg$_G(v_n)$. Moreover, we use V_i to denote the set of i vertices with highest weighted degree, i.e., $V_i = \{v_1, \ldots, v_i\}$.

3.1 A Simple Observation and A Faster FPT-AS

Our main insight to the Weighted Max k-VC problem is that there is always an $(1-\varepsilon)$-approximate solution which is entirely contained in $V_{O(k/\varepsilon)}$, as stated more formally below.

▶ **Observation 6.** *For any $\varepsilon > 0$, let $n' = \min\{k + \lceil k/\varepsilon \rceil, n\}$. Then, there exists $S^* \subseteq V_{n'}$ of size k such that $E_G(S^*) \geqslant (1-\varepsilon) \cdot \mathrm{OPT}_{Max\text{ k-}VC}(G,k)$.*

Note that this implies Theorem 1: we can enumerate all k-vertex subsets of $V_{n'}$ and find an $(1-\varepsilon)$-approximation for Max k-VC in $\binom{|V_{n'}|}{k}\mathsf{poly}(n) = O(n'/k)^k\mathsf{poly}(n) = O(1/\varepsilon)^k\mathsf{poly}(n)$ time.

Before we present a formal proof of the observation, let us briefly give an (informal) intuition behind the proof. Let S_{OPT} be the optimal solution for (G,k). Our goal is to construct another set $S^* \subseteq V_{n'}$ such that $E_G(S^*)$ is roughly the same as $E_G(S_{\mathrm{OPT}})$. To do so, we will just replace each vertex in $S_{\mathrm{OPT}} \setminus V_{n'}$ by a vertex in $V_{n'} \setminus S_{\mathrm{OPT}}$. Intuitively, this should be good for the solution, as we are replacing one vertex with another vertex that has higher weighted degree. However, this argument does not yet work: we might "double count" edges with both endpoints coming from the new vertices. The key point here is that, while we will not be able to avoid this double counting completely, we will be able to pick new vertices such that the total weight of such doubled counted edges is small. This is just because the set $V_{n'}$ is so large that even if we pick a random k vertices from it, the probability that a given added edge is double counted is only $O(\varepsilon)$.

Proof of Observation 6. Note that, if $n' = n$, the statement is obviously true. Hence, we may assume that $n' = k + \lceil k/\varepsilon \rceil$. Let $S_{\mathrm{OPT}} \subseteq V_G$ denote any optimal solution, i.e., any subset of V_G of size k with $E_G(S_{\mathrm{OPT}}) = \mathrm{OPT}_{\text{Max k-VC}}(G,k)$. Let $S^{\mathrm{in}}_{\mathrm{OPT}} = S_{\mathrm{OPT}} \cap V_{n'}$, $S^{\mathrm{out}}_{\mathrm{OPT}} = S_{\mathrm{OPT}} \setminus V_{n'}$ and $U = V_{n'} \setminus S_{\mathrm{OPT}}$.

We construct $S \subseteq V_{n'}$ in randomly as follows. We randomly select a subset $U^* \subseteq U$ of $|S^{\mathrm{out}}_{\mathrm{OPT}}|$ vertices uniformly at random, and let $S = S^{\mathrm{in}}_{\mathrm{OPT}} \cup U^*$. Clearly, S is a subset of $V_{n'}$ of size k. We will show that the expected value of $E_G(S)$ is at least $(1-\varepsilon) \cdot \mathrm{OPT}_{\text{Max k-VC}}(G,k)$. This would imply that there exists $S^* \subseteq V_{n'}$ of size k such that $E_G(S^*) \geqslant (1-\varepsilon) \cdot \mathrm{OPT}_{\text{Max k-VC}}(G,k)$ as desired.

To bound $\mathbb{E}[E_G(S)]$, let us first rearrange $E_G(S)$ as follows.

$$E_G(S) = E_G(S^{\mathrm{in}}_{\mathrm{OPT}}) + E_G(U^*) - E_G(U^*, S^{\mathrm{in}}_{\mathrm{OPT}}). \tag{1}$$

Let $\rho = |S^{\mathrm{out}}_{\mathrm{OPT}}|/|U|$; note here that $\rho \leqslant k/(n'-k) \leqslant \varepsilon$. We can now bound $\mathbb{E}[E_G(U^*, S^{\mathrm{in}}_{\mathrm{OPT}})]$ by

$$\begin{aligned}\mathbb{E}[E_G(U^*, S^{\mathrm{in}}_{\mathrm{OPT}})] &= \sum_{u \in U} \sum_{v \in S^{\mathrm{in}}_{\mathrm{OPT}}} w_{\{u,v\}} \cdot \Pr[u \in U^*] \\ &= \rho \cdot \sum_{u \in U} \sum_{v \in S^{\mathrm{in}}_{\mathrm{OPT}}} w_{\{u,v\}} \leqslant \varepsilon \cdot E_G(S^{\mathrm{in}}_{\mathrm{OPT}})\end{aligned} \tag{2}$$

Moreover, $\mathbb{E}[E_G(U^*)]$ can be rearranged as

$$\begin{aligned}
\mathbb{E}[E_G(U^*)] &= \mathbb{E}\left[\sum_{u\in U^*}\left(\text{w-deg}(u) - \frac{1}{2}\sum_{v\in U^*\setminus\{u\}} w_{\{u,v\}}\right)\right] \\
&= \mathbb{E}\left[\sum_{u\in U}\left(\text{w-deg}(u)\cdot \mathbb{1}[u\in U^*] - \frac{1}{2}\sum_{v\in U\setminus\{u\}} w_{\{u,v\}}\cdot \mathbb{1}[u\in U^*\wedge v\in U^*]\right)\right] \\
&= \sum_{u\in U}\left(\text{w-deg}(u)\cdot \Pr[u\in U^*] - \frac{1}{2}\sum_{v\in U\setminus\{u\}} w_{\{u,v\}}\cdot \Pr[u\in U^*\wedge v\in U^*]\right) \\
&\geqslant \sum_{u\in U}\left(\text{w-deg}(u)\cdot \rho - \frac{1}{2}\sum_{v\in U\setminus\{u\}} w_{\{u,v\}}\cdot \rho^2\right) \\
&\geqslant \rho(1-\rho/2)\cdot\left(\sum_{u\in U}\text{w-deg}(u)\right) \\
&\geqslant \rho(1-\varepsilon)\cdot\left(\sum_{u\in U}\text{w-deg}(u)\right) \qquad (3)
\end{aligned}$$

where in the first inequality we use the fact that $\Pr[u\in U^*\wedge v\in U^*]\leqslant \Pr[u\in U^*]\Pr[v\in U^*]=\rho^2$.

Recall that the vertices are sorted in decreasing order of degrees; thus, for all $u\in U$, we have $\text{w-deg}(u)\geqslant \left(\sum_{v\in S^{\text{out}}_{\text{OPT}}}\text{w-deg}(v)\right)/|S^{\text{out}}_{\text{OPT}}|\geqslant E_G(S^{\text{out}}_{\text{OPT}})/|S^{\text{out}}_{\text{OPT}}|$. From this and (3), we arrive at

$$\mathbb{E}[E_G(U^*)]\geqslant \rho(1-\varepsilon)\cdot|U|\cdot\left(E_G(S^{\text{out}}_{\text{OPT}})/|S^{\text{out}}_{\text{OPT}}|\right) = (1-\varepsilon)\cdot E_G(S^{\text{out}}_{\text{OPT}}) \qquad (4)$$

Plugging (2) and (4) back into (1), we indeed have

$$\mathbb{E}[E_G(S)]\geqslant (1-\varepsilon)(E_G(S^{\text{in}}_{\text{OPT}})+E_G(S^{\text{out}}_{\text{OPT}}))\geqslant (1-\varepsilon)\cdot E_G(S_{\text{OPT}}),$$

which concludes the proof. ◀

3.2 An Approximate Kernel

Observation 6 also naturally gives an $(1-\varepsilon)$-approximate kernel for Weighted Max k-VC where the new instance has $O(k/\varepsilon)$ vertices, as stated below.

Proof of Lemma 2. The reduction algorithm $\mathcal{A}$ works by taking the graph induced on $V_{n'}$ (where $n'=\min\{k+\lceil k/\varepsilon\rceil, n\}$ as in Observation 6) and add appropriate weights to self-loops to compensate for edges going out of $V_{n'}$. More precisely, $\mathcal{A}$ outputs (G',k) where $V_{G'}=V_{n'}$ and $w'_G(\{u,v\})=w'_G(\{u,v\})$ for all $u\neq v\in V_{G'}$ and $w_{G'}(u)=w_G(u)+E_G(\{u\},V_G\setminus V_{n'})$ for all $u\in V_{G'}$.

The solution lifting algorithm $\mathcal{B}$ simply outputs the same solution as its get. It is obvious to see that $E_{G'}(S)=E_G(S)$. Hence, if $E_{G'}(S)\geqslant \alpha\cdot \text{OPT}_{\text{Max k-VC}}(G',k)$, then Observation 6 implies that $E_G(S)=E_{G'}(S)\geqslant \alpha(1-\varepsilon)\cdot\text{OPT}_{\text{Max k-VC}}(G,k)$. This means that $(\mathcal{A},\mathcal{B})$ is an $(1-\varepsilon)$-approximate kernel; moreover, it is obvious that the graph output by $\mathcal{A}$ has size $O(k/\varepsilon)$ as desired. ◀

As mentioned earlier, the above kernel does not directly work for the unweighted case. Let us sketch below how we can modify the above proof to work in this case, albeit with a slightly worse $O(k/\varepsilon^2)$ vertices in the reduced instance. We omit the full proof, which is a simple undergraduate-level exercise, and only describe the main ideas. We do this in two steps; we first modify the proof for weighted graphs without self-loops and then we proceed to unweighted graphs.

- Suppose that the graphs G and G' must not contain any self-loops. Then, instead of adding self-loops as above, $\mathcal{A}$ will add $n_{\text{padded}} = \lceil kn'/\varepsilon \rceil = O(k/\varepsilon^2)$ padded vertices and let the weight between each padded vertex and $u \in V_{n'}$ be $\frac{E_G(\{u\}, V_G \setminus V_{n'})}{n_{\text{padded}}}$. Once again, if we take a look at any set $S \subseteq V_{n'}$, we immediately have $E_G(S) = E_{G'}(S)$. The only additional argument needed is that these padded vertices has little effect on any solution. Indeed, it is simple to see that the weighted degree of each padded vertex is at most $(\varepsilon/k) \cdot \text{OPT}_{\text{Max k-VC}}(G, k)$. Thus, throwing these vertices away from any subset of size k affect the total weights of edges covered by at most $\varepsilon \cdot \text{OPT}_{\text{Max k-VC}}(G, k)$, which implies that this is an $(1 - 2\varepsilon)$-approximate kernel. Adjusting ε appropriately gives the $(1 - \varepsilon)$-approximate kernel with $O(k/\varepsilon^2)$ vertices.
- The above idea naturally adapts to the unweighted case. Instead of adding an edge from every $u \in V_{n'}$ to all the padded vertices, we just add $E_G(\{u\}, V_G \setminus V_{n'})$ edges from each $u \in V_{n'}$ to different padded vertices. These edges are added in a way that each padded vertices has roughly the same degree. It is simple to check that, if the degree of all vertices $u \in V_{n'}$ is at most say k/ε^2, then this works immediately (with the same proof as above). The only issue is when there are vertices with degree larger than k/ε^2. (In this case, the number of edges required to be added may even be larger than n_{padded}!) Nevertheless, this issue can also be easily resolved, by observing that, if any vertex in V_k has degree at least k/ε, then we can always take it in our solution while guaranteeing that the solution still remains within $\varepsilon \cdot \text{OPT}_{\text{Max k-VC}}(G, k)$ of the optimum. Hence, the reduction algorithm can first greedily pick these vertices and then use the padded argument as above; since no large degree vertex remains, the proof of the second step now works and we have the desired approximate kernel.

3.3 Raghavendra-Tan Algorithm and An Improved Approximation

We next describe how our approximate kernel can be used a preprocessing step for the aforementioned algorithm of Raghavendra and Tan [52] for Max 2SAT with cardinality constraint to obtain improved approximation for Weighted Max k-VC.

Recall that the (weighted) Max 2SAT with cardinality constraint is the following problem. Given a collection $\mathcal{C}$ of conjunctions of at most two literals (of variables $\{x_1, \ldots, x_n\}$) and their associated weights, find an assignment to $\{x_1, \ldots, x_n\}$ satisfying $x_1 + \cdots + x_n = k$ that maximizes the total weights of satisfied clauses in $\mathcal{C}$. Raghavendra and Tan [52] device an algorithm with approximation ratio strictly greater than 0.92 for the problem, as stated below.

▶ **Theorem 7** ([52]). *For some $\alpha > 0.92$, there exists an α-approximation algorithm for Max 2SAT with cardinality constraint that runs in time*[7] $n^{\mathsf{poly}(n/k)}$.

[7] The running time of the algorithm is not stated in this form in [52] as they are only concerned about the case where $k = \Omega(n)$, for which the running time is polynomial. To see that the running time is of the form $n^{\mathsf{poly}(n/k)}$, we note that their algorithm needs the variance guaranteed in their Theorem 5.1 to be at most $\mathsf{poly}(k/n)$. This means that they need the SDP solution to be $\mathsf{poly}(k/n)$-independence; to

It is not hard to see that the Weighted Max k-VC can be formulated as Max 2SAT with cardinality constraint: we create a variable x_i for each vertex v_i, and, for each $\{v_i, v_j\} \in \binom{V_G}{\leqslant 2}$, we create a clause $(v_i \vee v_j)$ with weight $w_{\{v_i,v_j\}}$. Obviously, any solution to Max 2SAT satisfying $x_1 + \cdots + x_n = k$ is also a solution of Max k-VC with the same cost. Of course, the only issue in applying Raghavendra and Tan's algorithm here is that its running time $n^{\mathsf{poly}(n/k)}$ is not polynomial when $k = o(n)$. Fortunately, our approximate kernel above precisely circumvents this issue, as the reduction algorithm produces an instance (G', k) where $|V_{G'}| \leqslant O(k/\varepsilon)$. Thus, we can now apply the algorithm and arrives at 0.92 approximation for Weight Max k-VC in polynomial time.

Proof of Corollary 3. Let α be the approximation ratio from Theorem 7 and let $\varepsilon > 0$ be a sufficiently small constant such that $\alpha(1-\varepsilon) \geqslant 0.92$. Let $\mathcal{A}$ be the reduction algorithm for the $(1-\varepsilon)$-approximate kernel as defined in the proof of Lemma 2.

For any instance (G, k) of Weight Max k-VC, we apply $\mathcal{A}$ to arrive at a reduced instance (G', k) where $|V_{G'}| \leqslant O(k/\varepsilon)$. We then formulate the instance (G', k) as an instance of Max 2SAT with cardinality constraint and apply the Raghavendra-Tan algorithm, which gives an α-approximate solution, i.e., a set $S \subseteq V_{G'}$ of size k such that $E_{G'}(S) \geqslant \alpha \cdot \mathrm{OPT}_{\text{Max k-VC}}(G', k) \geqslant \alpha(1-\varepsilon) \cdot \mathrm{OPT}_{\text{Max k-VC}}(G, k) \geqslant 0.92 \cdot \mathrm{OPT}_{\text{Max k-VC}}(G, k)$. Note that the Raghavendra-Tan algorithm runs in $k^{\mathsf{poly}(|V_{G'}|/k)} = k^{\mathsf{poly}(1/\varepsilon)}$ time. Hence, we have found a 0.92-approximate solution for (G, k) in polynomial time. ◀

4 Minimum k-Vertex Cover

4.1 A Faster FPT-AS

We now present our result on Weighted Min k-VC, starting with the faster FPT-AS (Theorem 4). It will be more convenient for us to work with a multicolored version of the problem, which we call *Multicolored Min k-VC*. In Multicolored Min k-VC, we are given G, k as before and also a coloring $\chi : V_G \to [k]$. A set $S \subseteq V_G$ is said to be *colorful* if every vertex in S is assigned a different color, i.e., $|\chi(S)| = |S|$. The goal of Multicolored Min k-VC is to find a colorful $S \subseteq V_G$ of size k that maximizes $E_G(S)$. We overload the notation $\mathrm{OPT}_{\text{Min k-VC}}$ and also use it to denote the optimum of Multicolored Min k-VC; that is, we let $\mathrm{OPT}_{\text{Min k-VC}}(G, k, \chi) = \min_{S \in \binom{V_G}{k}, |\chi(S)|=k} E_G(S)$.

The main theorem of this section is the following FPT-AS for Multicolored Min k-VC.

▶ **Theorem 8.** *For any $\varepsilon > 0$, there exists an $(1+\varepsilon)$-approximation algorithm for Multicolored Min k-VC that runs in time $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$.*

We note here that the above lemma immediately gives an FPT-AS for (uncolored) Weight Min k-VC with similar running time (i.e. Theorem 4) via standard color-coding technique [3]. Specifically, they show how to construct a family $\mathcal{F}$ of k-perfect hash functions from $V_G \to \{1, \ldots, k\}$ in $2^{O(k)} \cdot \mathsf{poly}(n)$ time. By running the FPT-AS from Theorem 8 on (G, k, χ) for all $\chi \in \mathcal{F}$ and take the best solution among the outputs, we arrive at the FPT-AS for (uncolored) Weight Min k-VC.

We now proceed to discuss the intuition behind Theorem 8. The algorithm consists of two parts: subgraph generation and dynamic programming. Roughly speaking, the subgraph generation part will, for each set of colors $C \subseteq [k]$, generate connected colorful subsets

find such a solution, the running time required is $n^{\mathsf{poly}(n/k)}$ (see Theorem 4.1 in that paper).

$T \subseteq V_G$ whose color is C and record the minimum $E_G(T)$ in the table cell DP[C]. The second part of the algorithm then uses a standard dynamic programming to find a colorful k-vertex S with minimum $E_G(S)$.

For the purpose of exposition, let us assume for the moment that our graph is unweighted. The subgraph generation part is the heart of the algorithm, and, if not implemented in a careful manner, will be too slow. For instance, the trivial implementation of this is as a recursive function that maintains a set of included vertices S_{INCLUDED} and a set of active vertices S_{ACTIVE}. This function then picks any vertex $u \in S_{\text{ACTIVE}}$ and tries to select at most k neighbors of u to add into S_{INCLUDED} and S_{ACTIVE}; the function then remove u from S_{ACTIVE} and recursively call itself on this new sets. (Note that in this step it also makes sure that the set S_{INCLUDED} remains colorful; otherwise, the recursive call is not made.) The function stops when S_{ACTIVE} is empty and update DP[C] to be the minimum between the current value and $E_G(S_{\text{INCLUDED}})$. As the reader may have already noticed, while this algorithm records (exactly) the correct answer into the table, it is very slow. In particular, if say we run this on a complete graph, then it will generates $n^{\Theta(k)}$ subgraphs.

The algorithm of Gupta, Lee and Li [30, 29], while not stated in this exact form, can be viewed as a more careful implementation of this approach. In particular, they use the observation of Marx [46] (that was also outlined outline in Section 1.1): for unweighted graphs, if the optimal solution has any vertex with degree at least $\binom{k}{2}/\varepsilon$, simply picking the k vertices with minimum degrees would already be an $(1+\varepsilon)$-approximate solution. In other words, one may assume that the graph has degree bounded by $\binom{k}{2}/\varepsilon = O(k^2/\varepsilon)$. When this is the case, the algorithm from the previous paragraph in fact runs in $O(k/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$ time; the reason is that the number of choices to be made when adding a vertex is only $O(k^2/\varepsilon)$ instead of n as before. Hence, the running time becomes $O(k^2/\varepsilon)^k \cdot \mathsf{poly}(n) = (k/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$.

To obtain further speed up, we observe that, if at most $\varepsilon/2$ fraction of neighbors of a vertex u lies in the optimal solution, then ignoring all of them completely while branching would change the number of covered edges by factor of no more than ε. (This is shown formally in the proof below.) In other words, instead of trying all subsets of at most k neighbors of u. We may only try subsets with at least $d\varepsilon/2$ (and at most k) neighbors of u where d is the degree of u. The point here is that, while there are still $\exp(d)$ branches, we are adding at least $d\varepsilon/2$ vertices. Hence, the "branching factor per vertex added" is small: namely, for $j \geqslant d\varepsilon/2$, the "branching factor per vertex added" is only $\binom{d}{j}^{1/j} \leqslant ed/j \leqslant O(1/\varepsilon)$. This indeed gives the running time of $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$. (Note that such branching may result in a connected component being separated; however, when this is the case, the number of edges between the generated parts must be small anyway.)

Let us now shift our discussion to the edge-weighted graph case. Once again, as we will show formally in the proof, throwing away the edges adjacent to u with total weight at most $(\varepsilon/2) \cdot \text{w-deg}(u)$ only affects the solution value by no more than ε factor. However, this observation alone is not enough; specifically, unlike the unweighted case, this does not guarantee that many vertices must be selected. As an example, if there is a vertex v where $w_{\{u,v\}} = 0.5 \cdot \text{w-deg}(u)$, then even the set $\{v\}$ should be consider when we branch. Nevertheless, it is once again possible to show that, we can select a collection $\mathcal{T}$ of representative subsets such that, for any set $S \subseteq V_G$ (the true optimal set), we can arrive in a subset in $\mathcal{T}$ by throwing away vertices whose edges to u are of total weight at most $(\varepsilon/2) \cdot \text{w-deg}(u)$. In other words, it is "safe" to just consider branching with subsets in $\mathcal{T}$ instead of all subsets. Again, the collection $\mathcal{T}$ will satisfy the property that the "branching factor per vertex added" is small; that is, for any j, the number of j-element subsets that belong to $\mathcal{T}$ is at most $O(1/\varepsilon)^j$. The existence and efficient construction of such $\mathcal{T}$ is stated below in a more general form.

Note that, in the context of subgraph generation algorithm, one should think of $\delta = \varepsilon/2$, $\ell = n - 1$ (all vertices except u itself) and $P = \text{w-deg}(u) - w_{\{u\}}$.

▶ **Lemma 9.** *Let $a_1, \ldots, a_\ell \geqslant 0$ be any non-negative real numbers, let $\delta > 0$ be any positive real number, and let $P = \sum_{i \in [\ell]} a_i$. Then, there exists a collection $\mathcal{T}$ of subsets of $[\ell]$ such that*

- **(i)** *For all $j \in [\ell]$, we have $\left|\mathcal{T} \cap \binom{[\ell]}{j}\right| \leqslant O(1/\delta)^j$, and,*
- **(ii)** *For any $S \subseteq [\ell]$, there exists $T \in \mathcal{T}$ such that $T \subseteq S$ and $\sum_{i \in (S \setminus T)} a_i \leqslant \delta \cdot P$.*

Moreover, for any $j \in [\ell]$, $\mathcal{T} \cap \binom{[\ell]}{\leqslant j}$ can be computed in $O(1/\delta)^{O(j)} \ell^{O(1)}$ time.

Proof. Let $\pi : [\ell] \to [\ell]$ be any permutation such that $a_{\pi(1)} \geqslant \cdots \geqslant a_{\pi(\ell)}$. For each $j \in [\ell]$, we construct $\mathcal{T} \cap \binom{[\ell]}{j}$ by taking all j-element subsets of $\{\pi(1), \ldots, \pi(\min\{j \cdot \lceil 1/\delta \rceil, \ell\})\}$. We have

$$\left|\mathcal{T} \cap \binom{[\ell]}{j}\right| \leqslant \binom{j \cdot \lceil 1/\delta \rceil}{j} \leqslant \left(\frac{ej \cdot \lceil 1/\delta \rceil}{j}\right)^j \leqslant O(1/\delta)^j.$$

Moreover, it is clear that the set $\mathcal{T} \cap \binom{[\ell]}{j}$ can be generated in time polynomial in the size of the set and ℓ, which is $O(1/\delta)^{O(j)} \ell^{O(1)}$ as desired.

Finally, we will prove ii. Consider any subset $S \subseteq [\ell]$ and suppose that its elements are $\pi(i_1), \ldots, \pi(i_m)$. We pick the set T as follows: let t be the largest index such that $i_t \leqslant t \cdot \lceil 1/\delta \rceil$ and let $T = \{\pi(i_1), \ldots, \pi(i_t)\}$. Since $i_t \leqslant t \cdot \lceil 1/\delta \rceil$, T is a t-element subset from $\{\pi(1), \ldots, \pi(\min\{t \cdot \lceil 1/\delta \rceil, \ell\})$ and hence T belongs to $\mathcal{T}$. To prove ii, observe that, by definition of t, we have $i_g > g \cdot \lceil 1/\delta \rceil$ for all $g > t$. This means that

$$\sum_{i \in (S \setminus T)} a_i = \sum_{g=t+1}^{m} a_{\pi(g)} \leqslant \sum_{g=t+1}^{m} \left(\frac{1}{\lceil 1/\delta \rceil} \sum_{i=(g-1)\cdot\lceil 1/\delta \rceil + 1}^{g \cdot \lceil 1/\delta \rceil} a_i \right) \leqslant \frac{1}{\lceil 1/\delta \rceil} \sum_{i \in [\ell]} a_i \leqslant \delta \cdot P,$$

which concludes the proof. ◀

With the above lemma ready, we proceed to the proof of Theorem 8.

Proof of Theorem 8. The proof is based on the ideas outlined above. For simplicity, we will describe the algorithm that computes an approximation for $\text{OPT}_{\text{Min k-VC}}(G, k, \chi)$ rather than a subset $S \subseteq V_G$, i.e., it will output a number between $\text{OPT}_{\text{Min k-VC}}(G, k, \chi)$ and $(1 + \varepsilon) \cdot \text{OPT}_{\text{Min k-VC}}(G, k, \chi)$. It is not hard to see that the algorithm can be turned to provide a desired set as well.

As stated above, the algorithm consists of two parts: the subgraph generation part, and the dynamic programming part. The subgraph generation algorithm, which is shown below as Algorithm 1, is very much the same as stated earlier: it takes as an input the sets S_{ACTIVE} and S_{INCLUDED} (in addition to (G, k, χ)). If there is no more active vertex in S_{ACTIVE}, then it just updates the table DP to reflect $E_G(S_{\text{INCLUDED}})$. Otherwise, it pick a vertex u and try to branch on every representative T from $\mathcal{T}$ from Lemma 9 where the $\{a_i\}$'s are defined as $a_v = w_{\{u,v\}}$ for all $v \neq \{u\}$ and $\delta = \varepsilon/2$.

The dynamic programming (main algorithm) proceeds in a rather straightforward manner: after initializing the table, the main algorithm calls the subgraph generation subroutine starting with each vertex. Then, it uses dynamic programming to updates the table DP to reflect the fact that the answer may consist of many connected components. Finally, it outputs $\text{DP}[\{1, \ldots, k\}]$. The pseudo-code for this is given below as Algorithm 2.

Algorithm 1

1: **procedure** SubgraphGen($G, k, \chi, S_{\text{active}}, S_{\text{included}}$)
2: **if** $S_{\text{active}} = \emptyset$ **then**
3: $\text{DP}[\chi(S_{\text{included}})] \leftarrow \min\{\text{DP}[\chi(S_{\text{included}})], E_G(S_{\text{included}})\}$
4: **else**
5: $u \leftarrow$ Any element of S_{active}
6: $S_{\text{active}} \leftarrow S_{\text{active}} \setminus \{u\}$
7: $\mathcal{T} \leftarrow$ Subsets generated by Lemma 9 for $a_v = w_{\{u,v\}}$ for all $v \neq u$ and $\delta = \varepsilon/2$.
8: **for** $T \subseteq \mathcal{T} \cap \binom{V_G \setminus \{u\}}{\leqslant k}$ **do**
9: **if** $T \cap S_{\text{included}} = \emptyset$ **and** $S_{\text{included}} \cup T$ is colorful **then**
10: SubgraphGen($G, k, \chi, S_{\text{active}} \cup T, S_{\text{included}} \cup T$)
11: **end procedure**

Algorithm 2

1: **procedure** Min_k-VC(G, k, χ)
2: **for** $C \subseteq [k]$ **do**
3: $\text{DP}[C] \leftarrow \infty$
4: **for** $u \in V_G$ **do**
5: SubgraphGen($G, k, \chi, \{u\}, \{u\}$)
6: **for** $C \subseteq [k]$ in increasing order of $|C|$ **do**
7: **for** $C' \subseteq C$ **do**
8: $\text{DP}[C] \leftarrow \min\{\text{DP}[C], \text{DP}[C'] + \text{DP}[C \setminus C']\}$
9: **return** $\text{DP}[[k]]$
10: **end procedure**

Running Time Analysis. We will show that the running time of the algorithm is indeed $O(1/\varepsilon)^{O(k)}$. It is obvious that the dynamic programming step takes only $2^{O(k)} \cdot \mathsf{poly}(n)$ time, and it is not hard to see that each call to SubgraphGen, without taking into account the time spent in the recursed calls (Step 10), takes only $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$ time (because the bottleneck is the generation of $\mathcal{T} \cap \binom{V_G \setminus \{u\}}{\leqslant k}$ and this takes only $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$ time as guaranteed by Lemma 9). Thus, it suffices for us to show that, for each $u \in V$, SubgraphGen($G, k, \chi, \{u\}, \{u\}$) only generates $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$ leaves in the recursion tree. (By *leaves*, we refer to calls SubgraphGen($G, k, \chi, S_{\text{active}}, S_{\text{included}}$) where $S_{\text{active}} = \emptyset$. Note that, if SubgraphGen($G, k, \chi, \emptyset, S_{\text{included}}$) is called multiple times for the same S_{included}, we count each call separately.) The proof is a formalization of the "branching factor per vertex added" idea outlined before the proof.

In fact, we will prove a more general statement: for all colorful subsets $S_{\text{active}} \subseteq S_{\text{included}}$, SubgraphGen($G, k, \chi, S_{\text{active}}, S_{\text{included}}$) results in at most $(C/\varepsilon)^{2k - |S_{\text{included}}| - |S_{\text{included}} \setminus S_{\text{active}}|}$ leaves for some $C > 0$. In particular, let $C' > 0$ be a constant such that Lemma 9 gives the bound $|\mathcal{T} \cap \binom{[\ell]}{j}| \leqslant (C'/\delta)^j$; we will prove the statement for $C = 2C' + 2$.

We prove by induction on decreasing order of $|S_{\text{included}}|$ and $|S_{\text{included}} \setminus S_{\text{active}}|$ respectively. In the base case where $|S_{\text{included}}| = k$, the statement is obviously true, since the condition in Line 9 ensures that no more subroutine is executed. In another base case where $|S_{\text{included}} \setminus S_{\text{active}}| = |S_{\text{included}}|$, the statement is also obviously true since, in this case, we simply have $S_{\text{active}} = \emptyset$.

For the inductive step, suppose that, for some $0 \leqslant i < k$ and $1 \leqslant j \leqslant i$, the statement holds for all colorful subsets $S_{\text{ACTIVE}} \subseteq S_{\text{INCLUDED}}$such that $|S_{\text{INCLUDED}}| > i$, or, $|S_{\text{INCLUDED}}| = i$ and $|S_{\text{ACTIVE}}| < j$. Now, consider any colorful subsets $S_{\text{ACTIVE}} \subseteq S_{\text{INCLUDED}}$ such that $|S_{\text{INCLUDED}}| = i$ and $|S_{\text{ACTIVE}}| = j$. We will argue below that SubgraphGen$(G, k, \chi, S_{\text{ACTIVE}}, S_{\text{INCLUDED}})$ results in at most $(C'/\varepsilon)^{2k-i-(i-j)}$ leaves.

To do so, first observe that (1) in every recursive call, $|S_{\text{INCLUDED}} \setminus S_{\text{ACTIVE}}|$ increases by one (namely u becomes inactive) and (2) for every $0 \leqslant t \leqslant k-i$, the number of recursive calls for which $|S_{\text{INCLUDED}}|$ increases by t is at most $|\mathcal{T} \cap \binom{V_G \setminus \{u\}}{t}| \leqslant (C'/\varepsilon)^t$. As a result, by the inductive hypothesis, the number of leaves generated by SubgraphGen$(G, k, \chi, S_{\text{ACTIVE}}, S_{\text{INCLUDED}})$ is at most

$$\begin{aligned}\sum_{t=0}^{k-i}(C'/\varepsilon)^t \cdot (C/\varepsilon)^{2k-(i+t)-(i-j+1)} &= (C/\varepsilon)^{2k-i-(i-j+1)} \cdot \left(\sum_{t=0}^{k-i}(C'/C)^t\right) \\ \text{(Since } C \geqslant 2C') &\leqslant (C/\varepsilon)^{2k-i-(i-j+1)} \cdot 2 \\ \text{(Since } C \geqslant 2) &\leqslant (C/\varepsilon)^{2k-i-(i-j)}\end{aligned}$$

as desired.

In conclusion, for all colorful $S_{\text{ACTIVE}} \subseteq S_{\text{INCLUDED}}$, SubgraphGen$(G, k, \chi, S_{\text{ACTIVE}}, S_{\text{INCLUDED}})$ generates at most $(C/\varepsilon)^{2k-|S_{\text{INCLUDED}}|-|S_{\text{INCLUDED}} \setminus S_{\text{ACTIVE}}|}$ leaves. As argued above, this implies that the running time of the algorithm is at most $O(1/\varepsilon)^{O(k)} \cdot \mathsf{poly}(n)$.

Approximation Guarantee Analysis. We will now show that the output lies between $\text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$ and $(1+\varepsilon) \cdot \text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$. For convenience, let us define DP* to be the value of table DP filled by SubgraphGen calls; that is, this is the table before Line 6 in Algorithm 2. Observe the following relationship between DP and DP*:

$$\text{DP}[C] = \min_{\text{Partition } P \text{ of } C} \sum_{C' \in P} \text{DP}^*[C']. \tag{5}$$

It is now rather simple to see that the output is at least $\text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$. To do so, observe that, for any $C \subseteq [k]$, DP$^*[C]$ is equal $E_G(S_C)$ for some colorful $S_C \subseteq V_G$ with $\chi(S_C) = C$. This, together with (5), implies that the output must be equal to $\sum_{C' \in P} E_G(S_{C'})$ for some partition P of $[k]$ and colorful $S_{C'}$'s such that $\chi(S_{C'}) = C'$. Observe that this value is at least $E_G\left(\bigcup_{C' \in P} S_{C'}\right)$, which is at least $\text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$ since $\bigcup_{C' \in P} S_{C'}$ is a colorful set of size k.

Next, we will show that the output (i.e. DP$[[k]]$) is at most $(1+\varepsilon) \cdot \text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$. The following proposition is at the heart of this proof:

▶ **Proposition 10.** *For any non-empty colorful subset $S \subseteq V_G$, there exists a non-empty $S^{rep} \subseteq S$ such that*

$$DP^*[\chi(S^{rep})] \leqslant E_G(S^{rep}) \text{ and } E_G(S^{rep}, S \setminus S^{rep}) \leqslant \delta \cdot \text{w-deg}(S^{rep}).$$

Proof of Proposition 10. Let v be any vertex in S. Let us consider the call SubgraphGen$(G,\ \chi, k, \{v\}, \{v\})$. Consider traversing the following single branch in every execution of Step 10: pick $T \in \mathcal{T}$ such that $T \subseteq (S \setminus S_{\text{INCLUDED}})$ and $\sum_{i \in (S \setminus S_{\text{INCLUDED}}) \setminus T} w_{\{u,i\}} \leqslant \delta \cdot \sum_{i \in V_G} w_{\{u,i\}} = \delta \cdot \text{w-deg}_G(u)$. (We remark that such T is guaranteed to exist by Lemma 9; if there are more than one such T's, just choose an arbitrary one.) Suppose that always choosing such branch ends in a call SubgraphGen$(G, k, \chi, \emptyset, S^{\text{rep}})$. We will show that S^{rep} satisfies the desired properties.

First of all, observe that the fact we always choose $T \subseteq S$ ensures that $S^{\text{rep}} \subseteq S$ and that, since SubgraphGen$(G, k, \chi, \emptyset, S^{\text{rep}})$ is executed, we indeed have $\text{DP}[\chi(S^{\text{rep}})] \leqslant E_G(S^{\text{rep}})$. Hence, we are only left to argue that $E_G(S^{\text{rep}}, S \setminus S^{\text{rep}}) \leqslant \delta \cdot \text{w-deg}(S^{\text{rep}})$. To see that this is the case, observe that the second property of the T's chosen implies that $\sum_{i \in S \setminus S^{\text{rep}}} w_{\{u,i\}} \leqslant \delta \cdot \text{w-deg}(u)$. Summing this inequality over all $u \in S^{\text{rep}}$ immediately yields $E_G(S^{\text{rep}}, S \setminus S^{\text{rep}}) \leqslant \delta \cdot \text{w-deg}(S^{\text{rep}})$. ◀

With Proposition 10 ready, we can now prove that $\text{DP}[[k]] \leqslant (1+\varepsilon) \cdot \text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$. Let $S_{\text{OPT}} \subseteq V_G$ denote an optimal solution to the problem, i.e., S_{OPT} is a colorful k-vertex subset such that $E_G(S_{\text{OPT}}) = \text{OPT}_{\text{Min k-VC}}(G, k, \chi)$. Let $S_1 = S_{\text{OPT}}$. For $i = 1, \ldots,$ if $S_i \neq \emptyset$, we apply Proposition 10 to find a non-empty subset $S_i^{\text{rep}} \subseteq S_i$ such that

$$\text{DP}^*[\chi(S_i^{\text{rep}})] \leqslant E_G(S_i^{\text{rep}}) \text{ and } E_G(S_i^{\text{rep}}, S_{i+1}) \leqslant \delta \cdot \text{w-deg}(S_i^{\text{rep}}). \tag{6}$$

where $S_{i+1} = S_i \setminus S_i^{\text{rep}}$.

Observe here that $\{S_i^{\text{rep}}\}_{i \geqslant 1}$ is a partition of S_{OPT}. Thus, from (5) and (6), we have

$$\text{DP}[[k]] \overset{(5)}{\leqslant} \sum_{i \geqslant 1} \text{DP}^*[\chi(S_i^{\text{rep}})] \overset{(6)}{\leqslant} \sum_{i \geqslant 1} E_G(S_i^{\text{rep}}). \tag{7}$$

On the other hand, observe that $E_G(S_i) = E_G(S_i^{\text{rep}}) + E_G(S_{i+1}) - E_G(S_i^{\text{rep}}, S_{i+1})$. Thus, we have

$$\begin{aligned} E_G(S_{\text{OPT}}) &= \sum_{i \geqslant 1} (E_G(S_i) - E_G(S_{i+1})) \\ &= \sum_{i \geqslant 1} E_G(S_i^{\text{rep}}) - \sum_{i \geqslant 1} E_G(S_i^{\text{rep}}, S_{i+1}) \\ &\overset{(6)}{\geqslant} \sum_{i \geqslant 1} E_G(S_i^{\text{rep}}) - \delta \cdot \sum_{i \geqslant 1} \text{w-deg}(S_i^{\text{rep}}) \\ &= \sum_{i \geqslant 1} E_G(S_i^{\text{rep}}) - \delta \cdot \text{w-deg}(S_{\text{OPT}}). \end{aligned} \tag{8}$$

Finally, from (7), (8) and $\text{w-deg}(S_{\text{OPT}}) \leqslant 2 \cdot E_G(S_{\text{OPT}})$, we have $\text{DP}[[k]] \leqslant (1 + 2\delta) \cdot E_G(S_{\text{OPT}}) = (1 + \varepsilon) \cdot \text{OPT}_{\text{Min } k\text{-VC}}(G, k, \chi)$ which concludes the proof. ◀

4.2 Non-Existence of Polynomial Size Approximate Kernel

The above FPT-AS and the equivalence between existence of FPT approximation algorithm and approximate kernel [43] immediately implies that there exists an $(1-\varepsilon)$-approximate kernel for Weighted Min k-VC. However, this naive approach results in an approximate kernel of size $O(1/\varepsilon)^{O(k)}$. A natural question is whether there exists a polynomial-size approximate kernel for Weighted Min k-VC (similar to Weighted Max k-VC). In this section, we show that the answer to this question is likely a negative, assuming a variant of the Small Set Expansion Conjecture.

Our proof follows the framework of Lokshtanov et al. [43]. Let us recall that an equivalence relation R over strings on a finite alphabet Σ is said to be *polynomial* if (i) whether $x \sim y$ can be checked in $\mathsf{poly}(|x|+|y|)$ time and (ii) for every $n \in \mathbb{N}$, Σ^n has at most $\mathsf{poly}(n)$ equivalence classes. The framework of Lokshtanov et al. uses the notion of α-gap cross composition, as defined below. (This is based on the cross composition in the exact settings from [12].)

▶ **Definition 11** (α-gap cross composition [43])**.** Let L be a language and Π be a parameterized minimization problem. We say that L *α-gap cross composes* into Π (for $\alpha \leqslant 1$), if there is a polynomial equivalence relation R and an algorithm which, given strings $x_1, \cdots, x_t$ from the same equivalence class of R, computes an instance (y, k) of Π and $r \in \mathbb{R}$, in time $\mathsf{poly}(\sum_{i=1}^{t} |x_i|)$ such that the following holds:

(i) (Completeness) $\mathrm{OPT}_\Pi(y, k) \leqslant r$ if $x_i \in L$ for some $1 \leqslant i \leqslant t$,
(ii) (Soundness) $\mathrm{OPT}_\Pi(y, k) > r\alpha$ if $x_i \notin L$ for all $i \in [t]$, and,
(iii) k is bounded by a polynomial in $\log t + \max_{1 \leqslant i \leqslant t} |x_i|$.

A parameterized optimization problem is said to be *nice* if, given a solution to the problem, its cost can be computed in polynomial time. (Clearly, Weighted Min k-VC is nice.) The main tool from [43] is that any problem that α-gap cross composes to a nice parameterized optimization problem Π must be in $\mathsf{coNP/poly}$ if Π has α-approximate kernel[8]. In other words, if an NP-hard language α-gap cross composes to Π, then Π does not have α-approximate kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

▶ **Lemma 12** ([43])**.** *Let L be a language and Π be a nice parameterized optimization problem. If L α-gap cross composes to Π, and Π has a polynomial size α-approximate kernel, then $L \in \mathsf{coNP/poly}$.*

As stated earlier, our lower bound will be based on the Small Set Expansion Hypothesis (SSEH) [50]. To state the hypothesis, let us first recall the definition of edge expansion; for a graph G, the *edge expansion* of a subset of vertices $S \subseteq V_G$ is defined as $\Phi(S) := \frac{E_G(S, V_G \setminus S)}{\text{w-deg}(S)}$. Roughly speaking, SSEH, which was proposed in [50], states that it is NP-hard to determine whether (completeness) there is a subset of a specified size with very small edge expansion or (soundness) every subset of a specified size has edge expansion close to one. This is formalized below.

▶ **Definition 13** ($\mathsf{SSE}(\delta, \eta)$)**.** Given an unweighted regular graph G, distinguish between:

- (Completeness) There exists $S \subseteq V_G$ of size $\delta|V_G|$ such that $\Phi(S) \leqslant \eta$.
- (Soundness) For every $S \subseteq V_G$ of size $\delta|V_G|$, $\Phi(S) > 1 - \eta$.

▶ **Conjecture 14** (Small Set Expansion Hypothesis [50])**.** *For every $\eta > 0$, there exists $\delta = \delta(\eta) > 0$ such that SSE(δ, η) is NP-hard.*

Before we state the variant of SSEH that we will use, let us demonstrate why we need to strengthen the hypothesis. To do so, let us consider the $(2 - \varepsilon)$-factor hardness of approximation of Min k-VC as proved in [25], which our construction will be based on. The reduction takes in an input G to SSE(δ, η) and simply just outputs (G, k) where $k = \delta|V_G|$. The point is that, in a d-regular graph, a set S covers exactly $d(1 + \Phi(S))|S|/2$ edges. This means that, in the completeness case, there is a set S of size k that covers only $d(1 + \eta)k/2$ edges, whereas, in the soundness case, any set S of size k covers at least $d(2 - \eta)k/2$ edges. By selecting η sufficiently small, the ratio between the two cases is at least $(2 - \varepsilon)$, and hence [25] arrives at their $(2 - \varepsilon)$-factor inapproximability result.

Now, our cross composition is similar to this, except that we need to be to handle multiple instances at once. More specifically, given instance $G_1, \ldots, G_t$ of SSE(δ, η) where all $G_1, \ldots, G_t$ are d-regular for some d and $|V_{G_1}| = \cdots = |V_{G_t}|$, we want to produce an instance (G^*, k) where G^* is the disjoint union of $G_1, \ldots, G_t$ and $k = \delta|V|$. Once again, the

[8] We note that the result of [43] works even with a weaker notion than α-approximate kernel called *α-approximate compression*; see Definition 5.5 and Theorem 5.9 of [43] for more details.

completeness case works exactly as before. The issue lies in the soundness case: even though we know that every $S_i \subseteq V_{G_i}$ of size k has expansion close to one, it is possible that there exists $S_i \subseteq V_{G_i}$ of size much smaller than k that has small expansion. For instance, it might even be that $G_1, \ldots, G_t$ each contains a connected component of size k/t. In this case, we can take the union of these components and arrive at a set of size k that covers $dk/2$ edges, which is even smaller than the completeness case! In other words, for the composition to work, we want the soundness of SSEH to consider not only S's of size k, but also S's of size *at most* k. With this in mind, we can formalize our strengthened hypothesis as follows.

▶ **Definition 15** (Strong-SSE(δ, η))**.** Given an unweighted regular graph G, distinguish between:

- (Completeness) There exists $S \subseteq V_G$ of size $\delta|V_G|$ such that $\Phi(S) \leqslant \eta$.
- (Soundness) For every $S \subseteq V_G$ of size at most $\delta|V_G|$, $\Phi(S) > 1 - \eta$.

▶ **Conjecture 16** (Strong Small Set Expansion Hypothesis)**.** *For every $\eta > 0$, there exists $\delta = \delta(\eta) > 0$ such that Strong-SSE(δ, η) is NP-hard.*

We remark that it is known that a strengthening of SSEH where the soundness case is required for all S of size in $[\beta\delta|V|, \delta|V|]$ for any $\beta > 0$ is known to be equivalent to the original SSEH. (See Appendix A.2 of the full version of [51] for a simple proof.) This is closely related to what we want above, except that we need this to holds even for $|S| = o(|V|)$. To the best of our knowledge, the Strong SSEH as stated above is not known to be equivalent to the original SSEH.

Proof of Lemma 5. Let ε be any number that lies in $(0, 1]$. Let η be $\varepsilon/2$, and let $\delta = \delta(\eta) > 0$ be as guaranteed by Conjecture 14. We will show that Strong-SSE(δ, η) $(2 - \varepsilon)$-gap cross composes[9] into Min k-VC, which together with Lemma 12 immediately implies the statement in the lemma.

We define an equivalence relation R on instances of Strong-SSE(δ, η) by $G \sim G'$ iff $|V_G| = |V_{G'}|$ and w-deg(G) = w-deg(G'). It is obvious that R is polynomial. Given t instances $G_1, \ldots, G_t$ from the same equivalence class of R where $n = |V_{G_1}| = \cdots = |V_{G_t}|$ and $d =$ w-deg$(G_1) = \cdots =$ w-deg(G_t), we create an instance (G^*, k) of Min k-VC by letting G^* be the (disjoint) union of $G_1, \ldots, G_t$, $k = \delta n$, and $r = d\delta n(1 + \eta)/2$. We next argue the completeness and soundness of the composition.

Completeness. Suppose that, for some $i \in [t]$, there exists $S \subseteq V_{G_i}$ of size δn such that $\Phi_{G_i}(S) \leqslant \eta$. Then, the number of edges covered by S (in both G_i and G^*) is $d\delta n(1 + \Phi(S))/2 \leqslant d\delta n(1 + \eta)/2$. In other words, $\mathrm{OPT}_{\text{Max } k\text{-VC}}(G^*, k) \leqslant r$ as desired.

Soundness. Suppose that, for all $i \in [t]$ and $S \subseteq V_{G_i}$ of size at most δn, we have $\Phi_{G_i}(S) > (1 - \eta)$. Consider any set $S^* \subseteq V_{G^*}$ of size δn. Let S_i denote $S^* \cap V_{G_i}$. Observe that the number of edges covered by S^* is

$$\sum_{i \in [t]} d|S_i|(1 + \Phi_{G_i}(S_i))/2 \geqslant \sum_{i \in [t]} d|S_i|(2 - \eta)/2 = d\delta n(2 - \eta)/2 \geqslant (2 - \varepsilon)r,$$

[9] Note that strictly speaking Strong-SSE(δ, η) is not a language, but rather a promise problem (cf. [27]). Nevertheless, the notion of gap cross composes extends naturally to promise problems; the only changes are that in the yes case $x_i \in L$ should be changed to $x_i \in L_{\text{YES}}$ and in the no case $x_i \notin L$ should be changed to $x_i \in L_{\text{NO}}$. The result in Lemma 12 also holds for this case; for instance, see Lemma 5.11 and Theorem 5.12 of [43], where the gap cross composition also starts from a promise problem (Gap-Longest-Path).

where the first inequality comes from our assumption and the second comes from our choice of η. Thus, we have $\mathrm{OPT}_{\text{Max } k\text{-VC}}(G^*, k) > (2-\varepsilon)r$ as desired. ◀

We note here that the above proof produces G^* that is unweighted. As a result, the lower bound also applies for Unweighted Min k-VC.

5 Concluding Remarks

Let us make a few brief remarks regarding the tightness of running times of our algorithms.

- The W[1]-hardness proofs of Max k-VC and Min k-VC in [28] also implies that, even in the unweighted case, if we can approximate the problems to within $(1-1/n^2)$ and $(1+1/n^2)$ factors respectively, then we can solve the k-Clique problem with only polynomial overhead in running time. This implies the following lower bounds:
  1. Unless W[1] = FPT, there is no FPT-AS for Max k-VC and Min k-VC with running time $\exp(f(k)\cdot o(\log(1/\varepsilon)))\cdot \mathsf{poly}(n)$ for any function f (because this would give an FPT time algorithm for k-Clique when plugging in $\varepsilon = 1/n^2$).
  2. Unless k-Clique can be solved in $g(k)\cdot n^{o(k)}$ time for some function g, there is no FPT-AS for Max k-VC and Min k-VC with running time $O(1/\varepsilon)^{o(k)}\cdot\mathsf{poly}(n)$.
- For Max k-VC, the reduction that proves $(1+\delta)$-factor NP-hardness of approximation [49] is in fact a linear size reduction from the gap version of 3SAT. As a result, assuming the Gap Exponential Time Hypothesis (Gap-ETH)[10], there is no FPT-AS that runs in time $f(1/\varepsilon)^{o(k)}\cdot\mathsf{poly}(n)$ for any function f. Under the weaker ETH, a lower bound of the form $f(1/\varepsilon)^{o(k/\mathsf{poly}\log k)}\cdot\mathsf{poly}(n)$ for any f can be achieved via nearly linear size PCP [18]. (Note that we do not know any lower bound of this form for Min k-VC; in particular, it is not known whether Min k-VC is NP-hard to approximate even for a factor of 1.0001.)

An interesting remaining open question is to close the gap between the (polynomial time) approximation algorithms and hardness of approximation for Max k-VC. On the algorithmic front, we note that Austrin et al. [6] further exploited the techniques developed by Raghavendra and Tan [52] to achieve several improvements. Most importantly, they show that, for Max 2SAT with cardinality constraint, if the cardinality constraint is $x_1+\cdots+x_n = n/2$ (i.e. $k = n/2$), then an 0.94-approximation can be achieved in polynomial time. (In particular, the ratio here is the same as the ratio of the Lewin-Livnat-Zwick algorithm for Max 2SAT without cardinality constraint [42]; see also [5, 53]. Note that this ratio is still different from the hardness from [7].) This specific case is often referred to as *Max Bisection 2SAT*. Unfortunately, the algorithm does not naturally[11] extend to the case where $k \neq n/2$ and hence it is unclear how to employ this algorithm for Max k-VC.

On the hardness of approximation front, we remark that the hardness that follows from [7] holds even for the *perfect completeness* case. That is, even when there is a vertex cover of size k, it is still hard to find k vertices that cover 0.944 fraction of the edges. (See Appendix A of the full version [44].) Interestingly, there is an evidence that this perfect completeness case is easier: Feige and Langberg [23] shows that their algorithm achieves 0.8-approximation in this case, which is better than $(0.75+\delta)$-approximation that their algorithm yields in the

[10] Gap-ETH states that there is no $2^{o(n)}$-time algorithm that can distinguish between a fully satisfiable 3CNF formula and one which is not even 0.999-satisfiable [19, 45].

[11] In particular, the rounding algorithm involves scaling the bias of the variables (see Section 6 of [6]). For *Max Bisection 2SAT*, the sum of the bias is zero and hence scaling retains the sum. However, when the sum is non-zero, scaling changes the sum and hence the rounding algorithm produces a subset of size not equal to k.

general case. In fact, we can even get 0.94-approximation in this case as follows. First, we follow the kernelization for Vertex Cover [16] based on the Nemhauser-Trotter theorem [48]: on input graph (G, k), this gives a partition $V_0, V_{1/2}, V_1$ such that there exists a vertex cover S of size k such that $V_1 \subseteq S \subseteq V_{1/2} \cup V_1$. Moreover, the Nemhauser-Trotter theorem also ensures that $|V_{1/2}| = 2 \cdot (k - |V_1|)$. This means that we can restrict ourselves to the graph induced by $V_{1/2}$ and applies the aforementioned Max Bisection 2SAT from [6]. This indeed gives us a 0.94-approximation as desired. These suggest that it might be that the perfect completeness case is easier to approximate; thus, it would be interesting to see whether there is any way to construct harder instances with imperfect completeness.

References

1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *ALENEX*, pages 62–69, 2004.

2 Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

4 Nicola Apollonio and Bruno Simeone. The maximum vertex coverage problem on bipartite graphs. *Discrete Applied Mathematics*, 165:37–48, 2014.

5 Per Austrin. Balanced Max 2-Sat might not be the hardest. In *STOC*, pages 189–197, 2007.

6 Per Austrin, Siavosh Benabbas, and Konstantinos Georgiou. Better balance by being biased: A 0.8776-approximation for max bisection. In *SODA*, pages 277–294, 2013.

7 Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. *Theory of Computing*, 7(1):27–43, 2011.

8 Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *FOCS*, pages 453–462, 2009.

9 R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 27 – 45. North-Holland, 1985.

10 Reuven Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *J. Algorithms*, 39(2):137–144, 2001.

11 Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps, and nonapproximability-towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.

12 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

13 Édouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Georgios Stamoulis. Purely combinatorial approximation algorithms for maximum k-vertex cover in bipartite graphs. *Discrete Optimization*, 27:26–56, 2018.

14 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

15 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Logic*, 84(1):119–138, 1997.

16 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

17 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

18 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

19 Irit Dinur. Mildly exponential reduction from gap 3sat to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016.

20 Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. Towards a proof of the 2-to-1 games conjecture? *ECCC*, 23:198, 2016.

21 Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. On non-optimally expanding sets in grassmann graphs. *ECCC*, 24:94, 2017.

22 Irit Dinur and Shmuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

23 Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *J. Algorithms*, 41(2):174–211, 2001.

24 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93:30–40, 2018.

25 Rajiv Gandhi and Guy Kortsarz. On set expansion problems and the small set expansion conjecture. *Discrete Applied Mathematics*, 194:93–101, 2015.

26 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

27 Oded Goldreich. On promise problems: A survey. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, pages 254–290, 2006.

28 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41(3):501–520, 2007.

29 Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for k-cut. In *FOCS*, 2018. To appear.

30 Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for k-cut. In *SODA*, pages 2821–2837, 2018.

31 Eran Halperin and Uri Zwick. A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. *Random Struct. Algorithms*, 20(3):382–402, 2002.

32 Qiaoming Han, Yinyu Ye, Hantao Zhang, and Jiawei Zhang. On approximation of max-vertex-cover. *European Journal of Operational Research*, 143(2):342–355, 2002.

33 Qiaoming Han, Yinyu Ye, and Jiawei Zhang. An improved rounding method and semidefinite programming relaxation for graph partition. *Math. Program.*, 92(3):509–535, 2002.

34 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

35 Dorit S. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

36 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009.

37 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

38 Subhash Khot. On the power of unique 2-prover 1-round games. In *CCC*, page 25, 2002.

39 Subhash Khot, Dor Minzer, and Muli Safra. On independent sets, 2-to-2 games, and grassmann graphs. In *STOC*, pages 576–589, 2017.

40 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. *ECCC*, 25:6, 2018.

41 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

42 Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-sat and MAX DI-CUT problems. In *Integer Programming and Combinatorial Optimization,*

9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings, pages 67–82, 2002.

43 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *STOC*, pages 224–237, 2017.

44 Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. *arXiv preprint arXiv:1810.03792*, 2018.

45 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense csps. In *ICALP*, pages 78:1–78:15, 2017.

46 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

47 Burkhard Monien and Ewald Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Inf.*, 22(1):115–123, 1985.

48 George L. Nemhauser and Leslie E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Math. Program.*, 6(1):48–61, 1974.

49 Erez Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4:133–157, 1994.

50 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *STOC*, pages 755–764, 2010.

51 Prasad Raghavendra, David Steurer, and Madhur Tulsiani. Reductions between expansion problems. In *CCC*, pages 64–73, 2012.

52 Prasad Raghavendra and Ning Tan. Approximating CSPs with global cardinality constraints using SDP hierarchies. In *SODA*, pages 373–387, 2012.

53 Henrik Sjögren. Rigorous analysis of approximation algorithms for MAX 2-CSP. Master's thesis, KTH Royal Institute of Technology, 2009.

Simple Contention Resolution via Multiplicative Weight Updates

Yi-Jun Chang
University of Michigan
cyijun@umich.edu

Wenyu Jin
University of Michigan
wyjin@umich.edu

Seth Pettie[1]
University of Michigan
pettie@umich.edu

Abstract

We consider the classic *contention resolution* problem, in which devices conspire to share some common resource, for which they each need temporary and exclusive access. To ground the discussion, suppose (identical) devices wake up at various times, and must send a single *packet* over a shared *multiple-access channel*. In each time step they may attempt to send their packet; they receive ternary feedback $\{0, 1, 2^+\}$ from the channel, 0 indicating *silence* (no one attempted transmission), 1 indicating *success* (one device successfully transmitted), and 2^+ indicating *noise*. We prove that a simple strategy suffices to achieve a channel utilization rate of $1/e - O(\epsilon)$, for any $\epsilon > 0$. In each step, device i attempts to send its packet with probability p_i, then applies a rudimentary multiplicative weight-type update to p_i.

$$p_i \leftarrow \begin{cases} p_i \cdot e^{\epsilon} & \text{upon hearing silence (0)} \\ p_i & \text{upon hearing success (1)} \\ p_i \cdot e^{-\epsilon/(e-2)} & \text{upon hearing noise } (2^+) \end{cases}$$

This scheme works well even if the introduction of devices/packets is adversarial, and even if the adversary can *jam* time slots (make noise) at will. We prove that if the adversary jams J time slots, then this scheme will achieve channel utilization $1/e - \epsilon$, excluding $O(J)$ wasted slots. Results similar to these (Bender, Fineman, Gilbert, Young, SODA 2016) were already achieved, but with a lower constant efficiency (less than 0.05) and a more complex algorithm.

2012 ACM Subject Classification Networks → Network protocols, Mathematics of computing → Probabilistic algorithms

Keywords and phrases Contention resolution, multiplicative weight update method

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.16

1 Introduction

Suppose n identical devices have packets that they wish to transmit over a shared multiple access channel. For simplicity we assume that time is divided into discrete time slots and that the devices are synchronized. In each time slot they decide whether to attempt to transmit their packet or remain idle. In order to succeed the devices must monopolize the

[1] Supported by NSF grants CCF-1514383, CCF-1637546, and CCF-1815316.

channel for one time slot: if two or more devices transmit there is *noise* and if zero devices transmit there is *silence*. We assume that after each time step, all devices receive ternary feedback $\{0, 1, 2^+\}$ from the channel indicating how many devices attempted to transmit their packets. The reader should remember that the problem we are considering here is *abstract* contention resolution. The terms *packet*, *channel*, *noise*, etc. are merely meant to keep an easily visualized instance of the problem in mind.

The traditional way to solve this contention resolution problem is via *exponential backoff* [22]. Each device i holds a parameter p_i, initialized to some constant, say $1/2$. In each time step it executes the following protocol.[2]

Binary Exponential Backoff:

Device i $\begin{cases} \text{remains silent with probability } 1 - p_i \\ \text{transmits with probability } p_i\text{; if unsuccessful, set } p_i \leftarrow p_i/2. \end{cases}$

Although binary exponential backoff is empirically useful in many applications [22, 20, 18, 27, 19, 24], it has numerous shortcomings. Even if packets are injected into the system according to a Poisson distribution with some low expectation $\lambda > 0$ (i.e., a plausible and *non*-adversarial input distribution), binary exponential backoff will *eventually* become deadlocked and no more packets will ever be successfully sent [1, 4]. When all n packets are injected simultaneously, binary exponential backoff requires $n \log n$ steps to transmit all of them, and each device attempts to transmit its packet $\Theta(\log n)$ times [4], whereas $O(n)$ and $O(1)$ are optimal in this situation.

Recent research has tried to fix all the deficiencies of exponential backoff, and along many metrics this research has been quite successful. Bender et al. [4] studied the behavior of backoff-type protocols when all n packets arrive simultaneously, and proved that $O(n \log \log n / \log \log \log n)$ time is necessary and sufficient for *monotone* protocols (p_i decreases over time) whereas $O(n)$ time is possible with a non-monotone protocol. In the case that a *jammer* can jam slots at will, it is possible to achieve a (small) constant throughput on the unjammed slots [6, 3], even when the adversary controls the injection rate of new packets. In the [6] protocol each device makes $O(\log^2(n + J))$ transmissions, on average, where J is the number of jammed slots.

Bender, Kopelowitz, Pettie, and Young [7] considered a model motivated by battery-powered devices in which *both* transmitting and listening to the channel cost one unit of energy. They proved that constant channel utilization could be achieved with $O(\log(\log^* n))$ energy per device when an adversary controls the packet insertions (but cannot jam). The bound $O(\log(\log^* n))$ was later proved to be optimal [10].

Unfortunately, these recent advances come nowhere close to the minimalism and elegance of binary exponential backoff. In this work we design a contention resolution protocol that matches and substantially improves the main result of [6], while at the same time achieving something close to the simplicity of binary exponential backoff. Like backoff, our algorithm keeps a single numerical parameter (p_i) and is otherwise *stateless*: it keeps no information on its previous actions or the history of the channel.

[2] Exponential backoff comes in several more-or-less equivalent varieties. In the *windowed* version each device partitions its time in the system into consecutive *windows* $W_1, W_2, W_3, \ldots$, $|W_j| = 2^j$, and attempts to transmit at a uniformly chosen time slot in each window, until successful. In the *homogeneous* version, any device i in the system for t steps transmits with probability $p_i = 1/t$. The version of exponential backoff presented here requires the devices to keep track of less information.

Organization

In Section 2 we introduce our contention resolution protocol and analyze some parts of it. In Section 3 we design an unusual (continuous, real valued) potential function, and use it to to argue that the channel utilization of our protocol can get arbitrarily close to $1/e$. In Section 4 we give a more thorough literature survey on contention resolution and multiple access channels. We conclude in Section 5 with some observations and open problems.

2 Contention Resolution

If the devices have no distinguishing features to break symmetry, but they know what 'n' is, then a reasonable strategy is for everyone to transmit with probability $p = 1/n$, decrementing 'n' every time a packet is transmitted successfully. Observe that they succeed with probability $p_{\text{suc}} = \binom{n}{1}p(1-p)^{n-1} > e^{-1}$, and that $\lim_{n\to\infty} p_{\text{suc}} = 1/e$. More generally, the number of devices that transmit is, in the limit, a Poisson-distributed random variable:

$$\lim_{n\to\infty} \Pr(t \text{ devices transmit}) = \binom{n}{t}p^t(1-p)^{n-t} = e^{-1}/t!$$

Of course, for any finite n the distribution is merely *almost*-Poisson. In order to simplify things, we begin by considering an algorithm that creates channel feedback consistent with a number of transmitters that is Poisson-distributed. Each device i holds a variable p_i which it uses to determine its behavior according to the *Transmission Rule*.

Transmission Rule:

$$\text{Device } i \begin{cases} \text{remains silent with probability } e^{-p_i} \\ \text{transmits its packet with probability } p_i e^{-p_i} \\ \text{makes noise with probability } 1-(1+p_i)e^{-p_i} \end{cases}$$

If device i successfully transmits its packet, it halts.

We will later argue that "*making noise*" (even if the devices were capable of this) is unnecessary, and that the algorithm is improved if we simply transmit with probability $1 - e^{-p_i}$.

If the number of devices in the system is n, the probability of the three channel feedbacks (silence, success, and noise) is exactly:

$$\begin{aligned} p_{\text{sil}} &= \prod_{i\in[n]} e^{-p_i} = e^{-c} \\ p_{\text{suc}} &= \sum_{i\in[n]} p_i e^{-p_i} \cdot \prod_{j\in[n]\setminus\{i\}} e^{-p_j} = ce^{-c} \\ p_{\text{noi}} &= 1-(1+c)e^{-c} \end{aligned}$$

where c measures the *aggregate contention* in the system

$$c = \sum_{i\in[n]} p_i$$

The probability of success is maximized when $c = 1$. We would like to design an update rule such that c tends to move toward 1 whenever it is too small or too large. Observe that

when $c = 1$, the probability of hearing silence and noise are $1/e$ and $(e-2)/e$, respectively. In order for the update rule to be unbiased at $c = 1$, we must respond to noise and silence proportionately. Assuming device i has not successfully transmitted its packet, it applies the Update Rule to change p_i.

Update Rule:

$$p_i \leftarrow \begin{cases} p_i \cdot e^{\epsilon} & \text{upon hearing silence} \\ p_i & \text{upon hearing success} \\ p_i \cdot e^{-\epsilon/(e-2)} & \text{upon hearing noise} \end{cases}$$

Here the *step size* $\epsilon > 0$ is the only parameter of the algorithm. Since probabilities are updated multiplicatively, it is natural to measure the contention c on a logarithmic scale, so we define

$$\gamma = \ln(c)$$

In the absence of packet arrivals/departures, γ evolves according to a random walk on the reals that has a certain positive *attraction* towards the origin.[3] Observe that if γ and γ' are the values before and after an update, $\gamma' \in \{\gamma - \frac{\epsilon}{e-2}, \gamma, \gamma + \epsilon\}$.[4] We define the *attraction at* γ to be the expectation of $\gamma' - \gamma$, expressed in units of the step size ϵ.

$$\mathrm{attr}(\gamma) = p_{\mathrm{sil}}(\gamma) - \tfrac{1}{e-2} \cdot p_{\mathrm{noi}}(\gamma) = e^{-e^{\gamma}} - \tfrac{1}{e-2} \cdot (1 - (1+e^{\gamma})e^{-e^{\gamma}}).$$

In other words, $\mathrm{E}[\gamma'] = \gamma + \epsilon \cdot \mathrm{attr}(\gamma)$. Observe that because of the different step sizes, $\mathrm{attr}(\gamma)$ is asymptotic to 1 as $\gamma \to -\infty$ and asymptotic to $-1/(e-2)$ as $\gamma \to \infty$. We do not deal with the actual expression for $\mathrm{attr}(\gamma)$, but with a piecewise-linear approximation. See Figure 1.

▶ **Approximation 1.** *Define* $\widetilde{\mathrm{attr}}(\gamma)$ *as follows.*

$$\widetilde{\mathrm{attr}}(\gamma) = \begin{cases} 3/5 & \gamma < -1 \\ -(3/5)\gamma & \gamma \in [-1, 1] \\ -3/5 & \gamma > 1 \end{cases}$$

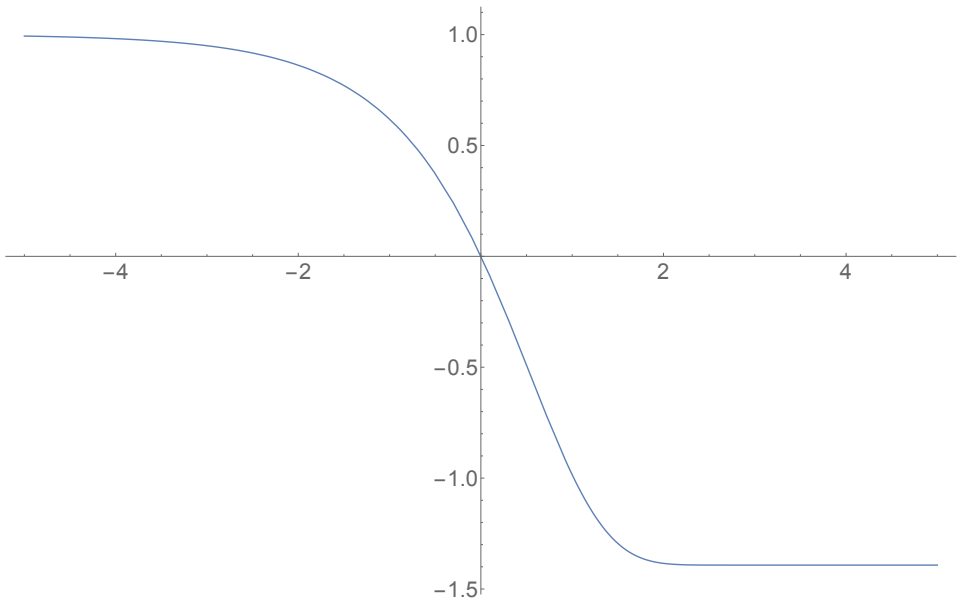
Then $\mathrm{attr}(0) = \widetilde{\mathrm{attr}}(0) = 0$ *and* $\mathrm{attr}(\gamma)/\widetilde{\mathrm{attr}}(\gamma) \geq 1$ *when* $\gamma \neq 0$.

2.1 Interlude: Homesick Random Walks

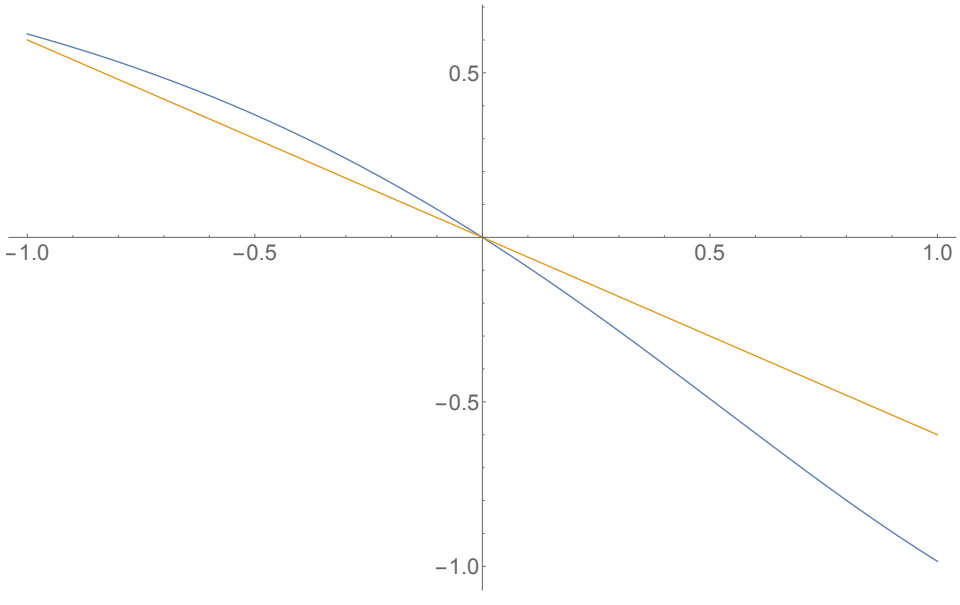
In order to build some intuition for how γ evolves, it is useful to think about what the stationary distribution of a *simplified* random walk looks like when the walk exhibits an attraction towards the origin. Consider a random walk on the integers $[-\delta^{-1}, \ldots, \delta^{-1}]$, $\delta > 0$, with the following transition probabilities. If the token is at $\pm i$ at step t, at step $t+1$ it moves toward 0 (i.e., to $\pm(i-1)$) with probability $(1 + i\delta)/2$ and away from 0 (i.e., to $\pm(i+1)$) with probability $(1 - i\delta)/2$. When it is at 0 it moves to -1 and 1 with equal probability. Let

[3] Interestingly, this attraction is qualitatively different in the positive and negative halves of the γ-axis, though it is numerically similar. Observe that when $\gamma > 0$ is large, the probability of hearing silence $p_{\mathrm{sil}} = e^{-c}$ is exponentially small in c, but when $\gamma < 0$ is small, the probability of hearing noise $p_{\mathrm{noi}} = 1 - (1+c)e^{-c} \approx c^2$ is quadratic in c.

[4] Actually, in the event of a successful transmission, γ will be reduced by some amount after one device withdraws from the system; we do not take that effect into account when calculating *attraction*.



(A)



(B)

Figure 1 (A) The attraction function $\mathrm{attr}(\gamma)$ is monotone decreasing in γ. (B) In the interval $[-1,1]$, $\widetilde{\mathrm{attr}}(\gamma) = -\frac{3}{5}\gamma$ is a conservative approximation in the sense that $\mathrm{attr}(\gamma)/\widetilde{\mathrm{attr}}(\gamma) > 1$.

$\pi(i)$ be the probability of being at either i or $-i$ under the stationary distribution. Then π satisfies the following equations.

$$\pi(0) = \pi(1) \cdot \frac{1+\delta}{2}$$
$$\pi(i-1) \cdot \frac{1-(i-1)\delta}{2} = \pi(i) \cdot \frac{1+i\delta}{2}$$

and hence

$$\pi(i) = 2\pi(0) \cdot \prod_{j=1}^{i} \frac{1-(j-1)\delta}{1+j\delta} = 2\pi(0) \cdot \prod_{j=0}^{i-1} \frac{1-j\delta}{1+(i-j)\delta} > 2\pi(0)(1-i\delta)^i \approx 2\pi(0)e^{-i^2\delta}.$$

In other words, a constant fraction of the mass of π is in the interval $[-\sqrt{\delta^{-1}}, \sqrt{\delta^{-1}}]$.

2.2 The Efficiency Curve

The back-of-the-envelope calculations above *suggest* that in its stationary distribution, the random walk generated by our contention resolution protocol puts a constant fraction of the probability mass in the real interval $[-\sqrt{\epsilon}, \sqrt{\epsilon}]$. Given that the efficiency of the algorithm is $1/e$ at $\gamma = 0$, it is natural to ask how the efficiency *degrades* as γ deviates from optimum. The overall efficiency of the algorithm will be determined by its behavior at the extremes of $\gamma \in [-\sqrt{\epsilon}, \sqrt{\epsilon}]$.

Recall that $p_{\text{suc}}(\gamma) = e^{\gamma}e^{-e^{\gamma}}$ is the probability of success as a function of $\gamma = \ln(c)$. By taking the first few terms of the Taylor expansion of p_{suc} at $\gamma = 0$, we have the following approximation. See Figure 2.

▶ **Approximation 2.**

$$p_{\text{suc}}(\gamma) = \frac{1}{e} - \frac{\gamma^2}{2e} - \frac{\gamma^3}{6e} + O(\gamma^4) > \frac{1}{e} - \gamma^2\left(\frac{1}{2e} + \frac{1}{6e}\right) > 1/e - \gamma^2/4.$$

To recap, we expect that γ will spend a constant fraction of its time in $[-\sqrt{\epsilon}, \sqrt{\epsilon}]$, and in this interval the expected channel utilization of the algorithm is at least $1/e - \epsilon/4$.

3 Amortized Analysis

Our goal is to show that the channel utilization of the algorithm is $1/e - O(\epsilon)$ by analyzing the *expected* change in a certain potential function Φ. Let Φ_t be the potential after time slot t and $n_t \geq 0$ be the number of packets inserted into the system just before time slot t begins. We intend to show that

$$\mathrm{E}[\Phi_t - \Phi_{t-1}] \leq -(1 - O(\epsilon)) + (e + O(\epsilon)) \cdot n_t.$$

In other words, each new packet carries with it $e + O(\epsilon)$ units of potential, and the combined effect of the Transmission & Update Rules reduces the potential by $1 - O(\epsilon)$ in expectation, thereby "paying for" this slot in a probabilistic sense. As a consequence, the channel utilization is $(1 - O(\epsilon))/(e + O(\epsilon)) = 1/e - O(\epsilon)$; the formal definition of channel utilization and its analysis will be presented in Section 3.4. For this analysis to work, it is important that newly injected devices initialize p_i properly.

Initialization Rule:

Upon activation, device i sets $p_i \leftarrow \epsilon^2$.

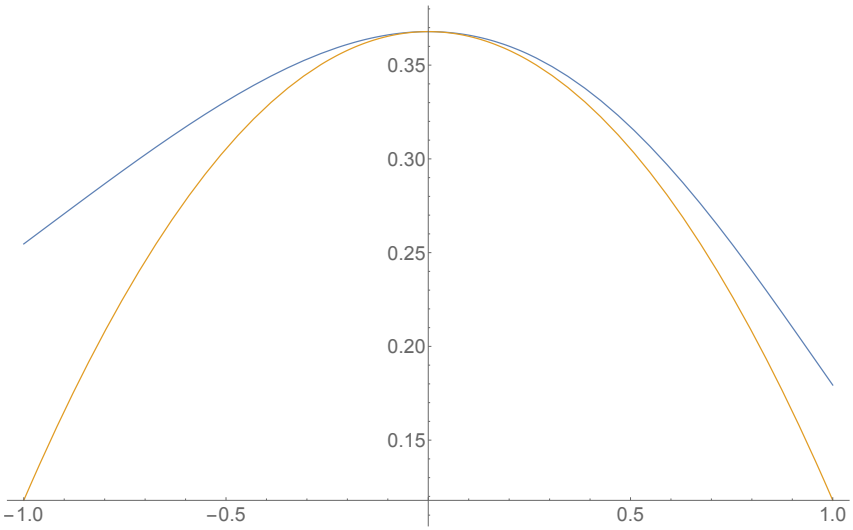


Figure 2 Top curve: the probability of successful transmission, as a function of γ, is $e^{\gamma}e^{-e^{\gamma}}$. Bottom curve: it is lower bounded by $1/e - \gamma^2/4$.

3.1 The Potential Function

The potential function Φ has three components.

$$\Phi = A(n) + B(\gamma) + C.$$

A depends only on n, the number of active devices still in the system, B depends on the contention γ, and C depends on the *relative* magnitude of the variables (p_i). The main term is

$$A(n) = en.$$

If γ is in the "efficient" range $[-\sqrt{\epsilon}, \sqrt{\epsilon}]$, then by Approximation 2 the expected change in A is $p_{\text{suc}}(\gamma) \cdot (-e) = -(1 - O(\epsilon))$, which pays for the time slot.

When $\gamma \in (-\infty, -\sqrt{\epsilon}) \cup (\sqrt{\epsilon}, \infty)$ we make up for the loss in efficiency by showing the expected contention becomes *closer* to optimum in the next time step; this is where the $B(\gamma)$ term comes into play. We define B to be the unique continuous function with $B(0) = 0$ and derivative

$$B'(\gamma) = \begin{cases} -\dfrac{5}{3\epsilon} & \text{when } \gamma < -1 \\ \dfrac{5}{3\epsilon}\gamma & \text{when } \gamma \in [-1, 1] \\ \dfrac{5}{3\epsilon} & \text{when } \gamma > 1 \end{cases}$$

In other words, when $\gamma \in [-1, 1]$, $B(\gamma) = \frac{5}{6\epsilon}\gamma^2$; see Figure 3.

Recall that $\mathrm{E}[\gamma'] = \gamma + \epsilon \cdot \text{attr}(\gamma)$, and by Approximation 1, $\text{attr}(\gamma)/\widetilde{\text{attr}}(\gamma) \geq 1$ when $\gamma \neq 0$. Therefore, whenever γ, γ' are in the "far off" range $(-\infty, -1] \cup [1, \infty)$, the expected change in B is smaller than $B'(\gamma) \cdot \epsilon \cdot \widetilde{\text{attr}}(\gamma) \leq -1$. When $\gamma \in [-1, 1]$ is close to the origin, it is also possible to show that the combined change in $A + B$ is less than $-(1 - O(\epsilon))$ in expectation. The formal analysis is in Section 3.2.

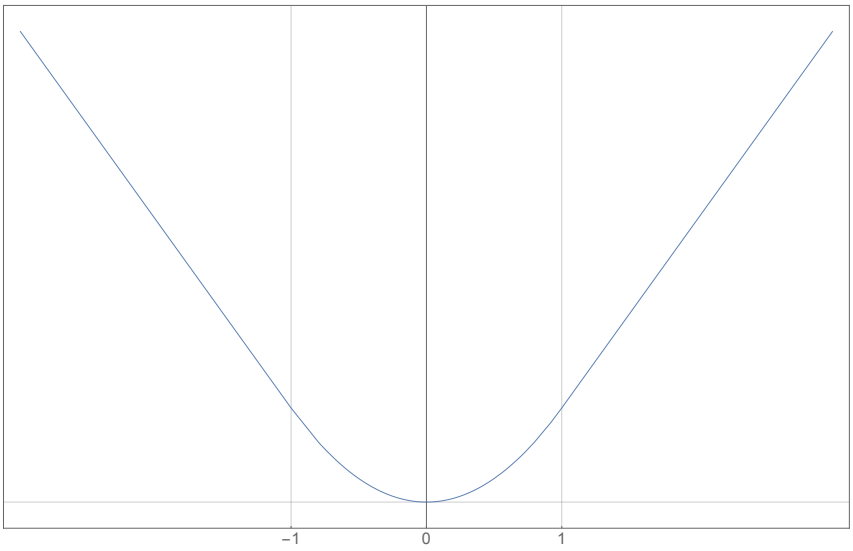


Figure 3 A schematic depiction of B. It is linear in the ranges $(-\infty, 1]$ and $[1, \infty)$, and quadratic in $[-1, 1]$.

In view of the above, under most of the circumstances, the expected change in A and B suffices to pay for each time slot. Occasionally, one packet p_i contributes the lion's share of the aggregate contention c, and when packet i is transmitted, $\gamma = \ln(c)$ drops sharply, increasing $B(\gamma)$. The third component C of Φ compensates for this effect:

$$C = \frac{5}{3\epsilon}(\gamma - \ln p_{\min}), \text{ where } p_{\min} = \min_i p_i$$

Observe that C remains unchanged by the Update Rule since γ and $\ln p_{\min}$ are increased/decreased by the same amount; but the value of C might change when the packet of the smallest transmission probability is successfully transmitted.

3.2 Analysis

We analyze the effect of one time slot on Φ by considering three actions sequentially (i) the increase in $\Phi = A + B + C$ caused by the insertion of new packets, (ii) the expected increase in $A + B$ caused by executing the Transmission & Update Rules, and (iii) in the event of a successful transmission from device i, the increase in $B + C$ caused by subtracting p_i from the aggregate contention. Part (i) is considered in Lemma 1; parts (ii) and (iii) are considered in Lemma 2.

In the analysis below, we work under the following assumption (*).

$$p_{\min} \leq \epsilon^2 \tag{*}$$

Assumption (*) often fails to hold when the number of active devices in the system is less than ϵ^{-2}. Section 3.4 justifies Assumption (*) by showing that it suffices to bound the long term channel utilization of our contention resolution protocol.

▶ **Lemma 1.** *Suppose all new packets follow the Initialization Rule and that Assumption (*) holds. Inserting m packets increases Φ by $(e + O(\epsilon))m$.*

Proof. We consider the contribution of each packet insertion individually. $A(n) = en$ clearly increases by e. If $\gamma \in [-1, \infty)$, C increases by $\frac{5}{3\epsilon}\ln(\frac{e^\gamma+\epsilon^2}{e^\gamma}) < \frac{5}{3\epsilon}(\epsilon^2/e^\gamma) = O(\epsilon)$ and B also increases by $O(\epsilon)$. (If $\gamma \in [-1, 0]$ then B is actually reduced; this only helps us.) If $\gamma \in (-\infty, -1]$, B is reduced by $\frac{5}{3\epsilon}(\ln(\frac{e^\gamma+\epsilon^2}{e^\gamma}))$ and C is increased by precisely the same amount. (Note that when $\gamma \ll -1$, the positive and negative changes to C and B can be very large.) ◀

▶ **Lemma 2.** *In each time step, the expected change in $A + B$ is $-1 + O(\epsilon)$. In the event of a successful transmission, the worst case change to $B + C$ is at most zero.*

Proof. Let γ and γ' be the values before and after applying the Update Rule in this time step. We consider three cases. Case 1 is when $B(\gamma)$ and $B(\gamma')$ are both on the *linear* parts, when $\gamma, \gamma' \in (-\infty, -1] \cup [1, \infty)$. Case 2 is when $B(\gamma)$ is on the quadratic part of B. Case 3 is when $B(\gamma)$ is on the linear parts of B but $B(\gamma')$ has a chance to be on the quadratic part of B.

Case 1: $\gamma \in (-\infty, -(1+\epsilon)] \cup [(1 + \frac{\epsilon}{e-2}, \infty)$. $B(\gamma)$ and $B(\gamma')$ are both guaranteed to be on the linear parts of B. The expected change[5] in B is therefore at most

$$\begin{aligned} B'(\gamma) \cdot (\mathrm{E}[\gamma'] - \gamma) &= B'(\gamma) \cdot (\epsilon \cdot \mathrm{attr}(\gamma)) \\ &\le \left(\mathrm{sign}(\gamma)\frac{5}{3\epsilon}\right) \cdot \left(\epsilon \cdot \widetilde{\mathrm{attr}}(\gamma)\right) \\ &\le \mathrm{sign}(\gamma) \cdot \frac{5}{3} \cdot \left(-\mathrm{sign}(\gamma)\frac{3}{5}\right) \le -1. \end{aligned}$$

In this range we do not count on successful transmissions; if they do occur, this reduces A even further.

Case 2: $\gamma \in [-1, 1]$. Here $B(\gamma) = \frac{5}{6\epsilon}\gamma^2$ behaves as a quadratic function. The expected change in $A + B$ is at most

$$p_{\mathrm{suc}}(\gamma)(-e) + \frac{5}{6\epsilon}\left[p_{\mathrm{sil}}(\gamma)\left(-\gamma^2 + (\gamma+\epsilon)^2\right) + p_{\mathrm{noi}}(\gamma)\left(-\gamma^2 + \left(\gamma - \frac{\epsilon}{e-2}\right)^2\right)\right]$$

Cancelling the γ^2 terms, we have

$$= p_{\mathrm{suc}}(\gamma)(-e) + \frac{5}{6\epsilon}\left[2\gamma\epsilon\left(p_{\mathrm{sil}}(\gamma) - \frac{1}{e-2}\cdot p_{\mathrm{noi}}(\gamma)\right) + \epsilon^2\left(p_{\mathrm{sil}}(\gamma) + \frac{1}{(e-2)^2}\cdot p_{\mathrm{noi}}(\gamma)\right)\right]$$

Observe that the term following $2\gamma\epsilon$ is exactly the definition of the *attraction* at γ, i.e., $\mathrm{attr}(\gamma)$. Because $e - 2 < 1$, the term following ϵ^2 is maximized over $\gamma \in [-1, 1]$ when $p_{\mathrm{noi}}(\gamma)$ is maximized. At $\gamma = 1$, $p_{\mathrm{sil}}(\gamma) + (e-2)^{-2}p_{\mathrm{noi}}(\gamma) < 1.53 < 8/5$. Simplifying, we have

$$< p_{\mathrm{suc}}(\gamma)(-e) + \frac{5}{6\epsilon}\left[2\gamma\epsilon \cdot \mathrm{attr}(\gamma) + \frac{8}{5}\epsilon^2\right]$$

[5] Here we are only considering the effect of the Update Rule on γ; decreases in γ due to successful transmission are considered when we analyze the effect on $B + C$.

Applying Approximations 1 and 2 (which state $p_{\text{suc}}(\gamma) > \frac{1}{e} - \frac{\gamma^2}{4}$ and $\gamma \cdot \text{attr}(\gamma) \leq -\frac{3}{5}\gamma^2$) and cancelling an ϵ factor, we have

$$\begin{aligned} &< \left(\frac{1}{e} - \frac{\gamma^2}{4}\right)(-e) + \frac{5}{6}\left[-2 \cdot \frac{3}{5}\gamma^2 + \frac{8}{5}\epsilon\right] \\ &= -1 + \gamma^2\left[\frac{e}{4} - 1\right] + \frac{4}{3}\epsilon \\ &\leq -1 + \frac{4}{3}\epsilon \end{aligned}$$

In other words, in each time step we lose at least $1 - O(\epsilon)$ units of potential in expectation, independent of γ.

Case 3: The remaining case covers the transition between the linear and quadratic parts, when $\gamma \in (-(1+\epsilon), -1) \cup (1, 1 + \frac{\epsilon}{e-2})$. The case 1 analysis applies here, up to a $(1 - O(\epsilon))$-factor since the slope of B between γ and γ' is either in the narrow interval $[-\frac{5}{3\epsilon}, -\frac{5}{3\epsilon}(1-\epsilon))$ or $(\frac{5}{3\epsilon}(1 - \frac{\epsilon}{e-2}), \frac{5}{3\epsilon}]$. This $O(\epsilon)$ loss is more than compensated for by the expected change in A, which is at most

$$p_{\text{suc}}(\gamma)(-e) \leq \left(1 - \frac{(1+\epsilon/(e-2))^2}{4}\right)(-e) = -\frac{e}{4} + O(\epsilon) \ll -\Theta(\epsilon).$$

This concludes our analysis of the change in $A + B$ caused by the Update Rule.

If device i successfully transmits its packet, the new γ is $\gamma'' = \ln(e^\gamma - p_i)$. The term C *decreases* by at least $\frac{5}{3\epsilon}(\gamma - \gamma'')$. Note that C decreases even more if the successful transmission causes $p_{\min}$ to increase. Because the derivative of B is always at least $-\frac{5}{3\epsilon}$, B *increases* by at most $\frac{5}{3\epsilon}(\gamma - \gamma'')$. Thus, the change in $B + C$ due to successful transmission is always at most zero. ◀

3.3 Variants and Extensions

Jamming

Our analysis easily extends to handle an adversarial *jammer*. In any time step, the jammer can make noise during the time slot; no packets are sent successfully and all active devices receive channel feedback 2^+ (noise). If they are following the Update Rule, then γ is reduced by $\epsilon/(e-2)$, and the increase in $B(\gamma)$ is *at most* $\frac{5}{3\epsilon} \cdot \frac{\epsilon}{e-2} < 2.33$. We charge the jammer $3.33 \cdot J$ for jamming a total of J slots: $1 \cdot J$ pays for the jammed slots and $2.33 \cdot J$ pays for the increase in potential. In other words, we expect the efficiency of our algorithm to be completely unchanged, if we ignore $3.33 \cdot J$ wasted time slots.

A Simpler Transmission Rule

Recall that in order to effect channel feedback consistent with a *precisely* Poisson distribution, the Transmission Rule allowed device i to "make noise" in a time slot (as if ≥ 2 devices were transmitting) with small probability $1 - (1 + p_i)e^{-p_i}$. Intuitively this is unwise. From a device's perspective, it is always better to attempt to transmit its packet rather than make noise. We show that from a system-wide perspective, the efficiency of Transmission Rule* is better than Transmission Rule.

Transmission Rule*:

$$\text{Device } i \begin{cases} \text{remains silent with probability } e^{-p_i} \approx 1 - p_i \\ \text{transmits its packet with probability } 1 - e^{-p_i} \approx p_i \end{cases}$$

If device i successfully transmits its packet, it halts.

▶ **Lemma 3.** *Let Φ be the current potential at the beginning of a time step. Let Φ' (resp., $\Phi^\star$) be the potential after applying Update Rule and Transmission Rule (resp., Transmission Rule*) in this time step. Then $\Phi' \geq \Phi^\star$.*

Proof. The only situation the two protocols differ in their behavior is when all devices remain silent, except for one, which chooses to make noise (Transmission Rule) or transmit its packet (Transmission Rule*). Observe that in this situation, following Transmission Rule* decreases Φ by at least e.[6] On the other hand, following Transmission Rule reduces Φ by at most $\frac{5}{3(e-2)}$ (when $\gamma > 1$), which is smaller than e. Thus, we must have $\Phi' \geq \Phi^\star$. ◀

3.4 Channel Utilization

One unfortunate aspect of our potential function is that it does not perform very well when the number of packets in the system is very small. For example, if there are a constant number of packets and γ is close to 0, then inserting a new packet with $p_i = \epsilon^2$ will likely *increase* C by $\Omega(\epsilon^{-1} \ln \epsilon^{-1})$, not $O(\epsilon)$ like we would hope. It turns out that in order to guarantee channel utilization of $1/e - O(\epsilon)$ over the long term, it is *not necessary* that the system be this efficient when number of active packets drops below a certain threshold, e.g., $O(\text{poly}(\epsilon^{-1}))$. Indeed, if the number of active packets is small, this is proof that the protocol is already functioning at the maximum possible efficiency (successful transmission rate = packet injection rate). Theorem 5 captures this intuition more formally. We first define a class of adversaries that strikes a nice balance between allowing essentially *arbitrary* adversarial behavior and adhering to some long-term average injection rate. This definition is more permissive than (λ, T)-adversaries [4].

▶ **Definition 4.** A *λ-adversary* injects packets and jams time slots indefinitely, under the constraint that $N_t < \lambda(t - \alpha J_t)$, for infinitely many values of t, where N_t and J_t are the total number of packets inserted and slots jammed by time t. Note that in our case, $\alpha = 3.33$.

In other words, if we delete αJ_t wasted slots from consideration, the adversary inserts λ packets per slot, on average, over the time period $[1, t]$. This condition is only required to hold infinitely often, which means the adversary is nearly always unconstrained.

Let us normalize the constants implicit in Lemmas 1, 2, and 3 so that whenever Assumption (*) holds, every packet insertion increases Φ by at most e, and every time step reduces Φ by at least $1 - \hat{\epsilon}$ in expectation, where $\hat{\epsilon} = \Theta(\epsilon)$ depends on ϵ.

▶ **Theorem 5.** *Suppose the packet-injection and channel jamming is controlled by a λ-adversary, with $\lambda + \epsilon < \frac{1-\hat{\epsilon}}{e}$. If the devices adhere to the Initialization, Transmission*, and Update Rules, then for infinitely many time slots, the number of active devices in the system will be less than ϵ^{-3}.*

[6] A decreases by e; B is unchanged by Update Rule, and the effect on $B + C$ caused by reducing γ is non-positive.

In other words, for infinitely many time slots, the channel utilization is optimal (up to an *additive* ϵ^{-3}).

Proof. We partition time into consecutive *epochs*, alternating between periods when there are at most ϵ^{-3} active devices and periods when there are greater than ϵ^{-3} active devices. We are not concerned with epochs of the first type. Suppose an epoch of the second type begins at time slot t_0. At this moment we evaluate the potential Φ of the system, with one minor change. In the definition of C, let

$$p_{\min} = \min\{\epsilon^2,\ \min_i p_i\}.$$

We argue that our previous analysis also applies when $p_{\min}$ is redefined in this way. We only need to consider the situation where we hear silence, which would ordinarily make $p_{\min}$ greater than ϵ^2, but it is forced to remain at ϵ^2. Since the epoch has not ended, the contention is $c \geq n\epsilon^2 \geq \epsilon^{-1}$. The probability of hearing silence is $e^{-c} \leq e^{-1/\epsilon}$ and this causes an extra increase in C-potential of $5/3$. On the other hand, the probability of seeing a successful transmission is ce^{-c}, and if this occurs, we see a reduction in potential of $e > 5/3$. The net expected effect of these two phenomena is negative. (Recall that our previous analysis did not take successful transmission into account when c was this large, so we are not double-counting this effect.)

Let Φ_0 be the initial potential endowment at time t_0.[7] Let t_1 be a time sufficiently far in the future when the adversary hits average insertion rate at most $\lambda = \frac{1-\hat{\epsilon}}{e} - \epsilon$. The number of packets inserted during the interval $[t_0, t_1]$ is at most the number of packets inserted by t_1, which is at most λt_1, and so the increase in potential due to packet insertion during the interval $[t_0, t_1]$ is always at most $e\lambda t_1$ (Lemma 1). In the interval $[t_0, t_1]$, the *expected* drop in potential is $(1-\hat{\epsilon})(t_1 - t_0 + 1)$ (Lemma 2).

We choose t_1 to be sufficiently large so that the expected net change in potential is $e\lambda t_1 - (1-\hat{\epsilon})(t_1 - t_0) < -\epsilon t_1$, and $-\epsilon t_1 + \Phi_0 < -\epsilon t_1/2$. Of course, if Φ ever reaches zero the epoch surely has ended. Seeing such a large deviation from the expectation is unlikely.

Let X_i be the potential drop at time step t_i (without taking into account the potential increase due to packet insertion), and let $X = \sum_{i=t_0}^{t_1} X_i$. The probability that the epoch has *not* ended by time t_1 is at most $\Pr[X \geq -(\Phi_0 + e\lambda t_1)]$. Note that $-(\Phi_0 + e\lambda t_1) \geq \mathrm{E}[X] + \epsilon t_1/2$ by our choice of t_1. By Azuma's inequality, this occurs with probability $\exp\left(-\Omega(\frac{(\epsilon t_1/2)^2}{t_1 - t_0 + 1})\right) = \exp(-\Omega(\epsilon^2 t_1))$.[8]

In the unlikely event that the epoch has not ended by time t_1, we can do the analysis with a sufficiently distant point $t_2 > t_1$ in the future. Thus, with probability 1 every epoch with $n > \epsilon^{-3}$ eventually ends. ◀

Theorem 5 establishes the main result of [6] but in a stronger form. In their protocol the efficiency is some constant much smaller than $1/e$. If there are n device injections, the protocol of [6] guarantees that the devices make $O(\log^2(n+J))$ transmission attempts each, on average. Our protocol also improves this aspect of [6], by showing that the number of transmission attempts is independent of n and J.

---

[7] Typically Φ_0 will be $\Theta(\epsilon^{-3})$ but we do not require this.

[8] Note that $|X_i|$ can be upper bounded by a universal Lipchitz constant. In each time step, the term A can only be decreased by at most e; the absolute change of B is at most $\frac{5}{3(e-2)}$; the extra increase in the component C in the modified potential is at most $\frac{5}{3}$.

▶ **Theorem 6.** *If the devices adhere to the Initialization, Transmission*, and Update Rules, the average number of transmission attempts per device is $e + O(\epsilon)$, under any adversarial strategy.*

Proof. The analysis is similar, except that the *expected* cost of a slot is now less than $c = e^\gamma$ rather than 1.[9] We redefine the potential Φ to be

$$\Phi = en + \frac{5}{3\epsilon} \cdot \max\{c, 1\}$$

An insertion increases n by 1 and c by ϵ^2, so the cost per insertion is at most $e + O(\epsilon)$. When $\gamma \in [0, \infty)$, the expected change in Φ caused by applying the Update Rule is

$$\begin{aligned} &p_{\text{suc}}(\gamma)(-e) + \frac{5}{3\epsilon}\left[p_{\text{sil}}(\gamma)(-c + ce^{\epsilon}) + p_{\text{noi}}(\gamma)(-c + ce^{-\epsilon/(e-2)})\right] \\ &= p_{\text{suc}}(\gamma)(-e) + \frac{5c}{3\epsilon}\left[p_{\text{sil}}(\gamma)(e^{\epsilon} - 1) + p_{\text{noi}}(\gamma)(e^{-\epsilon/(e-2)} - 1)\right] \end{aligned}$$

We apply the approximation $e^\epsilon \le 1 + \epsilon + \epsilon^2$ obtained from the Taylor expansion of e^x, yielding:

$$\begin{aligned} &\le p_{\text{suc}}(\gamma)(-e) + \frac{5c}{3\epsilon}\left[\epsilon \cdot p_{\text{sil}}(\gamma) - \frac{\epsilon}{e-2} p_{\text{noi}}(\gamma) + 2\epsilon^2\right] \\ &= p_{\text{suc}}(\gamma)(-e) + \frac{5c}{3}\Big[\text{attr}(\gamma) + 2\epsilon\Big] \qquad (**) \end{aligned}$$

We bound (**) depending on γ. When $\gamma \in [1, \infty)$, $\text{attr}(\gamma) + 2\epsilon < -3/5$, in which case (**) is

$$< \frac{5c}{3}\left(-\frac{3}{5}\right) = -c$$

and the slot is paid for, in a probabilistic sense. If $\gamma \in [0, 1]$, we bound (**) as

$$\begin{aligned} &< ce^c(-e) + \frac{5c}{3}\left(-(3/5)\ln c + 2\epsilon\right) \\ &= -c\left(e^{1-c} + \frac{5}{3}\ln c\right) + O(\epsilon) \\ &\le -c + O(\epsilon) \end{aligned}$$

When $\gamma \in (-\infty, 0]$, the Update Rule (alone) has no effect on Φ; only a successful packet transmission can decrease Φ. Let s be the number of transmitters in a given time slot. When $s = 0$ the cost is zero, so we can consider what the distribution on s looks like, normalized by the event that $s \ge 1$. The event $s = 1$ is good (it costs 1 and decreases Φ by e) and the events when $s \ge 2$ are bad (they cost s and leave Φ unchanged). Within the range $(-\infty, 0]$, the worst distribution on s occurs when $\gamma = 0$, simultaneously minimizing $\Pr(s = 1 | s \ge 1)$ and maximizing $\Pr(s = r | s \ge 1)$ for all $r \ge 2$. The efficiency at $\gamma = 0$ was already handled in the case $\gamma \in [0, 1]$ above. ◀

[9] $c = e^\gamma$ is an *upper bound* on the expected number of packets that transmit in this time step.

4 Related Work

The "new" idea in this work is to create a protocol that is optimal, in a sense, in its lowest energy configuration, by taking inspiration from the *multiplicative weight update* meta-algorithm [2]. Of course, there is nothing new under the sun, and even in the area of backoff-type protocols, updating parameters in response to channel feedback is quite common. Coming from a systems perspective, researches have evaluated variants of exponential backoff that use exponential increase/exponential decrease heuristics [28], multiplicative-increase/linear-decrease [8, 16], additive-increase/multiplicative-decrease [21], and a mixture of linear or multiplicative increase/linear decrease [11].[10] In the theoretical literature, Awerbuch et al. [3] used a multiplicative-weight-type update rule to achieve a (*very* small) constant rate of efficiency, in a model in which a jammer can jam up to a $(1-\epsilon)$-fraction of the slots. To our knowledge, no prior work has analyzed MWU-type contention resolution protocols in both a rigorous and numerically precise fashion.

We have shown that our protocol is *stable* for long-term injection rates approaching $1/e$. The stability of binary exponential backoff (BEB) and its variants has been studied extensively. Aldous [1] showed that for any constant Poisson injection rate $\lambda > 0$, BEB is unstable. Improving this, Bender et al. [4] proved that BEB is unstable at rate $\Omega(\log\log n/\log n)$[11] and stable at rate $O(1/\log n)$. See [15, 17] for other results on the stability of BEB. The failure of BEB to achieve stability even under constant injection rates motivated the development of more complex stable protocols [6, 3, 7]. Unlike BEB, these protocols (like ours) require that the channel feedback differentiate silence and noise.

Although the "$1/e$" threshold of our algorithm is optimal for *stateless* algorithms,[12], it is known that $1/e$ can be beaten, assuming the arrival times of packets are *Poisson-distributed*. The most efficient algorithms of Mosely and Humblet [25] and Tsybakov and Mikhailov [29] (slightly improving [9, 12]) are stable under arrival rates up to ≈ 0.48776. The best known upper bound on contention resolution (in Poisson-distributed injections, which also applies to adversarial injections) is 0.5874 [23]. The assumption of *ternary* feedback is essential here for both the upper and lower bounds. Goldberg et al. [13] have shown that if only transmitters receive feedback from the channel, then no protocol is stable at injections rates above 0.42. Pippenger [26] showed that if the channel reports the *exact* number of transmitters, that a batch of n synchronized devices can solve contention resolution in $n + o(n)$ time slots, i.e., achieving efficiency $1 - o(1)$.

Bender et al. [5] considered variants of BEB that are efficient with heterogeneous packet sizes (as opposed to unit-size packets). Goldberg et al. [14] designed a protocol in which the expected *delay* per packet is $O(1)$, assuming Poisson-injection at rates less than $1/e$.

5 Conclusions

In this work we proved that a simple and natural contention resolution protocol achieves channel utilization arbitrarily close to $1/e$, which is also resilient to a jammer that can jam a constant fraction of the slots. The "$1/e$" threshold of our algorithm cannot be improved by a *stateless* algorithm, and so in this sense its efficiency cannot be improved without a measurable increase in algorithmic complexity. We are confident that the protocols [25, 29]

[10] In these works, 'multiplicative' and 'exponential' are used interchangeably; 'additive' and 'linear' are used interchangeably.
[11] I.e., the number of packets that arrive at slot n is Poisson-distributed with expectation $\Omega(\log\log n/\log n)$.
[12] (meaning every device executes the same algorithm in each time step)

with efficiency 0.48776 for Poisson injections can be successfully adapted to adversarial injections using the same multiplicative weight update machinery developed here.

Although our protocol is very efficient in terms of transmission attempts ($e + O(\epsilon)$ vs. the $O(\log^2(n + J))$ of [6]) it does require that the devices listen for channel feedback in every step. In [7], "energy" is defined to be the number of slots spent accessing/listening to the channel. Is it possible to simultaeneously achieve energy cost $\text{poly}(\epsilon^{-1}, \log T)$[13] and $1/e - O(\epsilon)$ channel utilization?

References

1. D. J. Aldous. Ultimate instability of exponential back-off protocol for acknowledgment-based transmission control of random access communication channels. *IEEE Trans. Information Theory*, 33(2):219–223, 1987. doi:10.1109/TIT.1987.1057295.
2. S. Arora, E. Hazan, and S. Kale. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.
3. B. Awerbuch, A. W. Richa, and C. Scheideler. A jamming-resistant MAC protocol for single-hop wireless networks. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 45–54, 2008. doi:10.1145/1400751.1400759.
4. M. A. Bender, M. Farach-Colton, S. He, B. C. Kuszmaul, and C. E. Leiserson. Adversarial contention resolution for simple channels. In *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 325–332, 2005. doi:10.1145/1073970.1074023.
5. M. A. Bender, J. T. Fineman, and S. Gilbert. Contention Resolution with Heterogeneous Job Sizes. In *Proceedings 14th Annual European Symposium on Algorithms (ESA)*, pages 112–123, 2006. doi:10.1007/11841036_13.
6. M. A. Bender, J. T. Fineman, S. Gilbert, and M. Young. How to Scale Exponential Backoff: Constant Throughput, Polylog Access Attempts, and Robustness. In *Proceedings 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 636–654, 2016. doi:10.1137/1.9781611974331.ch47.
7. M. A. Bender, T. Kopelowitz, S. Pettie, and M. Young. Contention resolution with log-logstar channel accesses. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 499–508, 2016. doi:10.1145/2897518.2897655.
8. V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LAN's. In *Proceedings of the ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications*, pages 212–225, 1994. doi:10.1145/190314.190334.
9. J. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Trans. Information Theory*, 25(5):505–515, 1979. doi:10.1109/TIT.1979.1056093.
10. Y.-J. Chang, T. Kopelowitz, S. Pettie, R. Wang, and W. Zhan. Exponential separations in the energy complexity of leader election. In *Proceedings 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 771–783, 2017. doi:10.1145/3055399.3055481.
11. J. Deng, P. K. Varshney, and Z. Haas. A New Backoff Algorithm for the IEEE 802.11 Distributed Coordination Function. In *In Communication Networks and Distributed Systems Modeling and Simulation*, pages 215–225, 2004.

[13] I.e., if a packet is transmitted at time slot T, it listens in at most $\text{poly}(\epsilon^{-1}, \log T)$ slots.

12 R. G. Gallager. Conflict resolution in random access broadcast networks. In *Proceedings AFOSR Workshop on Communications Theory Applications, Provincetown, MA, Sept 17–20*, pages 74–76, 1978.

13 L. A. Goldberg, M. Jerrum, S. Kannan, and M. Paterson. A bound on the capacity of backoff and acknowledgment-based protocols. *SIAM J. Comput.*, 33(2):313–331, 2004. doi:10.1137/S0097539700381851.

14 L. A. Goldberg, P. D. MacKenzie, M. Paterson, and A. Srinivasan. Contention resolution with constant expected delay. *J. ACM*, 47(6):1048–1096, 2000. doi:10.1145/355541.355567.

15 J. Goodman, A. G. Greenberg, N. Madras, and P. March. Stability of binary exponential backoff. *J. ACM*, 35(3):579–602, 1988. doi:10.1145/44483.44488.

16 Z. J. Haas and J. Deng. On optimizing the backoff interval for random access schemes. *IEEE Trans. Communications*, 51(12):2081–2090, 2003. doi:10.1109/TCOMM.2003.820754.

17 J. Håstad, F. Thomson Leighton, and B. Rogoff. Analysis of Backoff Protocols for Multiple Access Channels. *SIAM J. Comput.*, 25(4):740–774, 1996. doi:10.1137/S0097539792233828.

18 M. Herlihy and J. E. B. Moss. Transactional Memory: Architectural Support for Lock-Free Data Structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA)*, pages 289–300, 1993. doi:10.1145/165123.165164.

19 V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM)*, pages 314–329, 1988. doi:10.1145/52324.52356.

20 J. F. Kurose and K. W. Ross. *Computer networking: a top-down approach*, volume 4. Addison-Wesley, Boston, 2009.

21 K. Li, I. Nikolaidis, and J. J. Harms. The analysis of the additive-increase multiplicative-decrease MAC protocol. In *Proceedings 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 122–124, 2013. doi:10.1109/WONS.2013.6578335.

22 R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976.

23 V. A. Mikhailov and B. S. Tsybakov. Upper bound for the capacity of a random multiple access system. *Problemy Peredachi Informatsii*, 17(1):90–95, 1981.

24 A. Mondal and A. Kuzmanovic. Removing exponential backoff from TCP. *Computer Communication Review*, 38(5):17–28, 2008. doi:10.1145/1452335.1452338.

25 J. Mosely and P. A. Humblet. A Class of Efficient Contention Resolution Algorithms for Multiple Access Channels. *IEEE Trans. Communications*, 33(2):145–151, 1985. doi:10.1109/TCOM.1985.1096261.

26 N. Pippenger. Bounds on the performance of protocols for a multiple-access broadcast channel. *IEEE Trans. Information Theory*, 27(2):145–151, 1981. doi:10.1109/TIT.1981.1056332.

27 R. Rajwar and J. R. Goodman. Speculative lock elision: enabling highly concurrent multithreaded execution. In *Proceedings of the 34th Annual International Symposium on Microarchitecture (MICRO)*, pages 294–305, 2001. doi:10.1109/MICRO.2001.991127.

28 N.-O. Song, B.-J. Kwak, and L. E. Miller. Analysis of EIED backoff algorithm for the IEEE 802.11 DCF. In *Proceedings 62nd IEEE Vehicular Technology Conference (VTC)*, volume 4, pages 2182–2186, 2005.

29 B. S. Tsybakov and V. A. Mikhailov. Slotted multiaccess packet broadcasting feedback channel. *Problemy Peredachi Informatsii*, 14(4):32–59, 1978.

A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum

Ce Jin
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
jinc16@mails.tsinghua.edu.cn

Hongxun Wu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
wuhx18@mails.tsinghua.edu.cn

Abstract

Given a multiset S of n positive integers and a target integer t, the Subset Sum problem asks to determine whether there exists a subset of S that sums up to t. The current best deterministic algorithm, by Koiliaris and Xu [SODA'17], runs in $\tilde{O}(\sqrt{nt})$ time, where $\tilde{O}$ hides poly-logarithm factors. Bringmann [SODA'17] later gave a randomized $\tilde{O}(n+t)$ time algorithm using two-stage color-coding. The $\tilde{O}(n+t)$ running time is believed to be near-optimal.

In this paper, we present a simple and elegant randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time. Our new algorithm actually solves its counting version modulo prime $p > t$, by manipulating generating functions using FFT.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques

Keywords and phrases subset sum, formal power series, FFT

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.17

Acknowledgements The authors would like to thank the anonymous reviewers for their helpful comments.

1 Introduction

Given a multiset S of n positive integers and a target integer t, the Subset Sum problem asks to determine whether there exists a subset of S that sums up to t. It is one of Karp's original NP-complete problems [9], and is widely taught in undergraduate algorithm classes. In 1957, Bellman gave the well-known dynamic programming algorithm [2] in time $O(nt)$. Pisinger [12] first improved it to $O(nt/\log t)$ on word-RAM models. Recently, Koiliaris and Xu gave a deterministic algorithm [10, 11] in time $\tilde{O}(\sqrt{nt})$, which is the best deterministic algorithm so far. Bringmann [4] later improved the running time to randomized $\tilde{O}(n+t)$ using color-coding and layer splitting techniques. Abboud et al. [1] recently showed that Subset Sum has no $O(t^{1-\epsilon}n^{O(1)})$ algorithm for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) is false, so the $\tilde{O}(n+t)$ time bound is likely to be near-optimal.

In this paper, we present a new randomized algorithm matching the $\tilde{O}(n+t)$ running time by Bringmann [4]. The basic idea of our approach is quite straightforward. For prime $p > t$, we give an $\tilde{O}(n+t)$ algorithm for $\#_p$Subset Sum, the counting version of Subset Sum problem modulo p. Then the decision version can be solved with high probability by randomly picking a sufficiently large prime p.

A closely related problem is #Knapsack, which asks for the number of subsets S such that $\sum_{s\in S} s \le t$. There are extensive studies on approximation algorithms for the #Knapsack problem [6, 8, 13, 7]. Our algorithm can solve the modulo p version $\#_p$Knapsack in near-linear pseudopolynomial time for prime $p > t$.

Compared to the previous near-linear time algorithm for Subset Sum by Bringmann [4], our algorithm is simpler and more practical. The precise running time of our algorithm is $O(n + t\log^2 t)$ with error probability $O((n+t)^{-1})$. If a faster algorithm for manipulating formal power series by Brent [3] is applied, it can be improved to $O(n + t\log t)$ time (see Remark on Lemma 2), which is faster than Bringmann's algorithm by a factor of $\log^4 n$.

1.1 Main ideas of our algorithm

The Subset Sum instance can be encoded as a generating function $A(x) = \prod_{i=1}^{n}(1 + x^{s_i})$, where $s_1, \ldots, s_n$ are the input integers, and our goal is to compute the t-th coefficient of $A(x)$ and see whether it is zero or not.

Instead of directly expanding $A(x)$, we consider its logarithm $B(x) = \ln(A(x))$. Using basic properties of the logarithm function and its power series, it's possible to compute the first $t+1$ coefficients of $B(x)$ in $\tilde{O}(t)$ time. Then we can recover the first $t+1$ coefficients of $A(x) = \exp(B(x))$ in $\tilde{O}(t)$ time using a simple divide and conquer algorithm with FFT (or a slightly faster algorithm by Brent [3]).

The coefficients involved in the algorithm could be exponentially large. To avoid dealing with high-precision numbers, we pick a prime p and perform arithmetic operations efficiently in the finite field $\mathbb{F}_p$, and in the end check whether the result is zero modulo p. By picking random p from a large interval, the algorithm succeeds with high probability.

2 Preliminaries

2.1 Subset sum problem

Given n (not necessarily distinct) positive integers $s_1, s_2, \ldots, s_n$ and a target sum t, the Subset Sum problem is to decide whether there exists a subset of indices $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i\in I} s_i = t$. We also consider the $\#_p$Subset Sum problem, which asks for the number of such subsets I modulo p. We use the word RAM model with word length $w = \Theta(\log t)$ throughout this paper.

2.2 Polynomials and formal power series

Formal power series

Let $R[x]$ denote the ring of polynomials over a ring R, and $R[[x]]$ denote the ring of formal power series over R. A formal power series $f(x) = \sum_{i=0}^{\infty} f_i x^i$ is a generalization of a polynomial with possibly an infinite number of terms. Polynomial addition and multiplication naturally generalize to $R[[x]]$. Composition $(f \circ g)(x) = f(g(x)) = \sum_{i=0}^{\infty} f_i \Big(\sum_{j=1}^{\infty} g_j x^j\Big)^i$ is well-defined for $f(x) = \sum_{i=0}^{\infty} f_i x^i \in R[[x]]$ and $g(x) = \sum_{j=1}^{\infty} g_j x^j \in xR[[x]]$. Here $xR[[x]]$ (or $xR[x]$) denotes the set of series in $R[[x]]$ (or polynomials in $R[x]$) with zero constant term.

Exponential and logarithm

We are familiar with the following two series in $\mathbb{Q}[[x]]$,

$$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}x^k}{k}, \tag{1}$$

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}, \tag{2}$$

satisfying

$$\exp\big(\ln(1+f(x))\big) = 1 + f(x), \tag{3}$$

and

$$\ln\big((1+f(x))(1+g(x))\big) = \ln(1+f(x)) + \ln(1+g(x)) \tag{4}$$

for any $f(x), g(x) \in x\mathbb{Q}[x]$.

Modulo x^{t+1}

Our algorithm only deals with the first $t+1$ terms of any formal power series. For $f(x), g(x) \in R[[x]]$, we write $f(x) \equiv g(x) \pmod{x^{t+1}}$ if $[x^i]f(x) = [x^i]g(x)$ for all $0 \le i \le t$, where $[x^i]f(x)$ denotes the i-th coefficient of $f(x)$.

As an example, define

$$\exp_t(x) = \sum_{i=0}^{t} \frac{x^i}{i!} \tag{5}$$

as a t-th degree polynomial in $\mathbb{Q}[x]$. Then $\exp(f(x)) \equiv \exp_t(f(x)) \pmod{x^{t+1}}$ clearly holds for any $f(x) \in x\mathbb{Q}[[x]]$.

2.3 Modulo prime p

To avoid dealing with large fractions or floating-point numbers, we will work in the finite field $\mathbb{F}_p = \{\bar{0}, \bar{1}, \ldots, \overline{p-1}\}$ of prime order $p = 2^{\Theta(\log t)}$. Addition and multiplication in $\mathbb{F}_p$ take $O(1)$ time in the word RAM model. Finding the multiplicative inverse of a nonzero element in $\mathbb{F}_p$ takes $O(\log p)$ time using extended Euclidean algorithm [5, Section 31.2].

Our algorithm will regard polynomial coefficients as elements from $\mathbb{F}_p$. The coefficients can be rational numbers, but their denominators should not have prime factor p. Formally, let

$$\mathbb{Z}_{p\mathbb{Z}} = \{r/s \in \mathbb{Q} : r, s \text{ are coprime integers, } p \text{ does not divide } s\} \tag{6}$$

and apply the canonical homomorphism from $\mathbb{Z}_{p\mathbb{Z}}[x]$ to $\mathbb{F}_p[x]$, determined by

$$r/s \mapsto \bar{s}^{-1}\bar{r}, \quad x \mapsto x. \tag{7}$$

We use $\bar{A}$ or $A \bmod p$ to denote A's image in $\mathbb{F}_p[x]$.

From now on we assume $p > t$, so that $\exp_t(x) \in \mathbb{Z}_{p\mathbb{Z}}[x]$ (see equation (5)), and let $\overline{\exp_t}(x)$ denote its image in $\mathbb{F}_p[x]$.

```
procedure COMPUTE(l, r) ▷ after COMPUTE(l, r) returns, all values g_1, ..., g_r are ready
    if l < r then
        m ← ⌊(l + r)/2⌋
        COMPUTE(l, m)
        for i ← m + 1, m + 2, ..., r do
            g_i ← g_i + i^{-1} Σ_{j=l}^{m} (i − j) f_{i−j} g_j
        end for
        COMPUTE(m + 1, r)
    end if
end procedure

procedure MAIN
    Initialize g_0 ← 1, g_i ← 0 (1 ≤ i ≤ t)
    COMPUTE(0, t)
end procedure
```

Figure 1 Algorithm for computing $g_1, \ldots, g_t$.

2.4 Computing exponential using FFT

▶ **Lemma 1** (FFT). *Given two polynomials $f(x), g(x) \in \mathbb{F}_p[x]$ of degree at most t, one can compute their product $f(x)g(x)$ in $O(t \log t)$ time.*

Proof. The classic FFT algorithm [5, Chapter 30] can multiply $f(x)$ and $g(x)$, regarded as polynomials in $\mathbb{Z}[x]$, in $O(t \log t)$ time. Then take the remainder of each coefficient modulo p. ◀

Lemma 2 is a classical result on manipulating formal power series, and is the main building block of our algorithm.

▶ **Lemma 2** (Brent [3]). *Given a polynomial $f(x) \in x\mathbb{F}_p[x]$ of degree at most t $(t < p)$, one can compute a polynomial $g(x) \in \mathbb{F}_p[x]$ in $\tilde{O}(t)$ time such that $g(x) \equiv \overline{\exp_t}(f(x)) \pmod{x^{t+1}}$.*

▶ Remark. Brent's algorithm [3] uses Newton's iterative method and runs in time $O(t \log t)$. Here we describe a simpler $O(t \log^2 t)$ algorithm by standard divide and conquer. We present the algorithm as over $\mathbb{Q}$ for notational simplicity.

Proof. Let $f(x) = \sum_{i=1}^{t} f_i x^i$ and $g(x) = \exp(f(x)) = \sum_{i=0}^{\infty} g_i x^i$. Then $g'(x) = g(x)f'(x)$. Comparing the $(i-1)$-th coefficients on both sides gives a recurrence relation

$$g_i = i^{-1} \sum_{j=0}^{i-1} (i-j) f_{i-j} g_j \tag{8}$$

with initial value $g_0 = 1$. The desired coefficients $g_1, \ldots, g_t$ can be computed using the algorithm in Figure 1, which simply reorganizes the computation of recurrence formula (8) as a recursion.

To speed up this algorithm, define polynomial $F(x) = \sum_{k=0}^{r-l} k f_k x^k, G(x) = \sum_{j=0}^{m-l} g_{j+l} x^j$ and use FFT to compute $H(x) = F(x)G(x)$ in $O((r-l)\log(r-l))$ time after COMPUTE(l, m) returns. Then $\sum_{j=l}^{m} (i-j) f_{i-j} g_j = [x^{i-l}]H(x)$, and hence the **for** loop runs in $O(r-m)$ time. The total running time is $T(t) = 2T(t/2) + O(t \log t) = O(t \log^2 t)$. ◀

3 Main algorithm

Recall that we are given n positive integers $s_1, \ldots, s_n$ and a target sum t. Consider the generating function $A(x)$ defined by

$$A(x) = \prod_{i=1}^{n} (1 + x^{s_i}). \tag{9}$$

The number of subsets that sum up to t is $[x^t]A(x)$. The Subset Sum instance has a solution if and only if $[x^t]A(x) \neq 0$.

▶ **Lemma 3.** *Suppose $[x^t]A(x) \neq 0$. Let p be a uniform random prime from $[t+1, (n+t)^3]$. With probability $1 - O((n+t)^{-1})$, p does not divide $[x^t]A(x)$.*

Proof. Notice that $[x^t]A(x) \leq 2^n$, so it has at most n prime factors. Since there are $\Omega((n+t)^2)$ primes in the interval, the probability that p divides $[x^t]A(x)$ is $O((n+t)^{-1})$. ◀

▶ **Lemma 4.** *Let $B(x) = \ln(A(x)) \in \mathbb{Q}[[x]]$. For prime $p > t$, in $\tilde{O}(t)$ time one can compute $([x^r]B(x)) \bmod p$ for all $0 \leq r \leq t$.*

Proof. By definition of $B(x)$,

$$B(x) = \ln\Big(\prod_{i=1}^{n}(1 + x^{s_i})\Big) = \sum_{i=1}^{n} \ln(1 + x^{s_i}) = \sum_{i=1}^{n}\sum_{j=1}^{\infty} \frac{(-1)^{j-1}}{j} x^{s_i j}. \tag{10}$$

Let a_k be the size of the set $\{j : s_j = k\}$, and define polynomial

$$B_t(x) = \sum_{i=1}^{n}\sum_{j=1}^{\lfloor t/s_i \rfloor} \frac{(-1)^{j-1}}{j} x^{s_i j} = \sum_{k=1}^{t}\sum_{j=1}^{\lfloor t/k \rfloor} \frac{a_k(-1)^{j-1}}{j} x^{jk}. \tag{11}$$

Then $[x^r]B_t(x) = [x^r]B(x)$ for all $0 \leq r \leq t$.

Note that the denominators j in (11) do not have prime factor p. After preparing the multiplicative inverses $\bar{j}^{-1}$ for each $1 \leq j \leq t$, we can compute all $([x^r]B_t(x)) \bmod p$ by simply iterating over k, j in equation (11), which only takes $\sum_{k=1}^{t} \lfloor t/k \rfloor = O(t \log t)$ time. ◀

▶ **Lemma 5.** *For prime $p > t$, one can compute $([x^r]A(x)) \bmod p$ for all $0 \leq r \leq t$ in $\tilde{O}(t)$ time.*

Proof. Let $B(x) = \ln(A(x))$. Then $A(x) = \exp(B(x)) \equiv \exp_t(B_t(x)) \pmod{x^{t+1}}$, where $B_t(x) = \sum_{i=0}^{t}([x^i]B(x))x^i$. We use Lemma 4 to compute $B_t(x)$'s image $\overline{B_t}(x) \in \mathbb{F}_p[x]$, and then use Lemma 2 to compute the first $t+1$ terms of $\overline{\exp_t}(\overline{B_t}(x))$, which give the values of $([x^r]A(x)) \bmod p$ for all $0 \leq r \leq t$. ◀

▶ **Theorem 6.** *The Subset Sum problem can be solved in time $\tilde{O}(n+t)$ by a randomized algorithm with one-sided error probability $O((n+t)^{-1})$.*

Proof. By sampling and using Miller-Rabin primality test [5, Section 31.8], we can pick a uniform random prime p from interval $[t+1, (n+t)^3]$ in $(\log(n+t))^{O(1)}$ time with $O((n+t)^{-1})$ failure probability. Then the theorem immediately follows from Lemma 3 and Lemma 5. ◀

References

1. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019. To appear. URL: http://arxiv.org/abs/1704.04546.
2. Richard E. Bellman. *Dynamic programming*. Princeton University Press, 1957.
3. Richard P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic Computational Complexity*, pages 151–176. Elsevier, 1976. doi:10.1016/B978-0-12-697560-4.50014-9.
4. Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017. doi:10.1137/1.9781611974782.69.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 3rd edition, 2009.
6. Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 693–699, 2003. doi:10.1145/780542.780643.
7. Paweł Gawrychowski, Liran Markin, and Oren Weimann. A Faster FPTAS for #Knapsack. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 64:1–64:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.64.
8. Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Štefankovic, Santosh Vempala, and Eric Vigoda. An FPTAS for #knapsack and related counting problems. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 817–826, 2011. doi:10.1109/FOCS.2011.32.
9. Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.
10. Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017. doi:10.1137/1.9781611974782.68.
11. Konstantinos Koiliaris and Chao Xu. Subset Sum Made Simple. *CoRR*, abs/1807.08248, 2018. URL: http://arxiv.org/abs/1807.08248.
12. David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999. doi:10.1006/jagm.1999.1034.
13. Romeo Rizzi and Alexandru I. Tomescu. Faster FPTASes for counting and random generation of knapsack solutions. In *European Symposium on Algorithms (ESA)*, pages 762–773, 2014. doi:10.1007/978-3-662-44777-2_63.

Submodular Optimization in the MapReduce Model

Paul Liu
Stanford University, USA
paulliu@stanford.edu

Jan Vondrak
Stanford University, USA
jvondrak@stanford.edu

Abstract

Submodular optimization has received significant attention in both practice and theory, as a wide array of problems in machine learning, auction theory, and combinatorial optimization have submodular structure. In practice, these problems often involve large amounts of data, and must be solved in a distributed way. One popular framework for running such distributed algorithms is MapReduce. In this paper, we present two simple algorithms for cardinality constrained submodular optimization in the MapReduce model: the first is a $(1/2 - o(1))$-approximation in 2 MapReduce rounds, and the second is a $(1 - 1/e - \epsilon)$-approximation in $\frac{1+o(1)}{\epsilon}$ MapReduce rounds.

2012 ACM Subject Classification Theory of computation → MapReduce algorithms, Theory of computation → Distributed computing models, Theory of computation → Algorithm design techniques, Theory of computation → Submodular optimization and polymatroids

Keywords and phrases mapreduce, submodular, optimization, approximation algorithms

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.18

1 Introduction

Let $f : 2^V \to \mathbb{R}^+$ be a function satisfying $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ for all $A \subseteq B$ and $e \notin B$. Such a function is called *submodular*. When f satisfies the additional property $f(A \cup \{e\}) - f(A) \geq 0$ for all A and $e \notin A$, we say f is *monotone*.

Many combinatorial optimization problems can be cast as submodular optimization problems. Such problems include classics such as max cut, min cut, maximum coverage, and minimum spanning tree [6]. Although submodular optimization encompasses several NP-Hard problems, well-known greedy approximation algorithms are known [11]. We focus on the special case of monotone submodular maximization under a cardinality constraint k, i.e.

$$OPT := \max_{S \subseteq V, |S| \leq k} f(S), \ f \text{ is monotone.}$$

In particular, it is known that one can approximate a cardinality constrained monotone submodular maximization problem to a factor of $1 - 1/e$ of optimal.

Due to rapidly growing datasets, recent focus has been on submodular optimization in distributed models [1, 3, 4, 5, 8, 10]. In this work, we focus on the MapReduce model, where complexity is measured as the number of synchronous communication rounds between the machines involved. The current state of the art for cardinality constrained submodular maximization is the algorithm of Barbosa et al. [5], which achieves a $1/2 - \epsilon$ approximation

in 2 rounds and was the first to achieve a $1 - 1/e - \epsilon$ approximation in $O\left(\frac{1}{\epsilon}\right)$ rounds. Both algorithms actually require significant duplication of the ground set (each element being sent to $\Omega(\frac{1}{\epsilon})$ machines). Since this might be an issue in practice, [5] mentions that without duplication, the two algorithms could be implemented in $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ and $O(\frac{1}{\epsilon^2})$ rounds, respectively. Earlier, Mirrokni and Zadimoghaddam [10] gave a 0.27-approximation in 2 rounds without duplication and a 0.545-approximation with $\Theta(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ duplication.

Our contribution. We focus on the most practical regime of MapReduce algorithms for cardinality constrained submodular maximization, which is a small constant number of rounds and no duplication of the dataset. To our knowledge, the 0.27-approximation of [10] has been the best result in this regime so far.

We describe a simple thresholding algorithm which achieves the following: In 2 rounds of MapReduce, with one random partitioning of the dataset (no duplication), we obtain a $(1/2 - \epsilon)$-approximation. In 4 rounds, we obtain a 5/9-approximation. More generally, in $2t$ rounds, we obtain a $(1 - (1 - \frac{1}{t+1})^t - \epsilon)$-approximation, which we show to be optimal for this type of algorithm. Crucially, the parameter ϵ does not affect the number of rounds, and only mildly affects the memory (in that ϵ can be taken to $\tilde{O}(\sqrt{k/n})$ without asymptotically increasing the memory).

Our algorithm is inspired by the work of Kumar et al. [8] and McGregor-Vu [9] in the streaming setting. It is also similar to a recent algorithm of Assadi-Khanna [2], who study the communication complexity of the maximum coverage problem. As such, our algorithm is not particularly novel, but we believe that our analysis of its performance in the MapReduce model is, thus simplifying and improving the previous work of [5] and [10].

Open question. The most intriguing remaining question in our opinion (for the cardinality constrained submodular problem) is whether $\Theta(1/\epsilon)$ rounds are necessary to achieve a $(1 - 1/e - \epsilon)$-approximation. So far there is no evidence that a $(1 - 1/e)$-approximation in a constant number of rounds is impossible.

1.1 The MapReduce Model

There are many variants of MapReduce models, and algorithms between the different models are largely transferable. We use a variant of the $\mathcal{MRC}$ model of Karloff et al. [7]. In this model, an input of size N is distributed across $O(N^\delta)$ machines, each with $O(N^{1-\delta})$ memory. We relax the model slightly, and allow one central machine to have memory slightly expanded to $\tilde{O}(N^{1-\delta})$.

Computation then proceeds in a sequence of synchronous communication rounds. In each round, each machine receives an input of size $O(N^{1-\delta})$. Each machine then performs computations on that input, and produces output messages which are delivered to other machines (specified in the message) as input at the start of the next round. The total size of these output messages must also be $O(N^{1-\delta})$ per machine. We refer the reader to the work of Karloff et al. [7] for additional details.

In our applications, we assume the input is a set of elements V and a cardinality parameter k. Each machine has an oracle that allows it to evaluate f. Under these constraints, we assume that each machine has memory $O(\sqrt{nk})$ (except for a single 'central' machine with $O(\sqrt{nk}\log k)$ memory) and that there are $\sqrt{n/k}$ machines in total.

Algorithm 1: ThresholdGreedy(S, G, τ).

Input: An input set S, a partial greedy solution G with $|G| \leq k$, and a threshold τ.
Output: A set $G' \supseteq G$ such that $f_{G'}(e) < \tau$ for all $e \in S$ if $|G| < k$ or $f(G) \geq \tau k$.

```
G' ← G
for e ∈ S do
    if f_{G'}(e) ≥ τ and |G'| < k then G' ← G' ∪ {e}
return G'
```

Algorithm 2: ThresholdFilter(S, G, τ).

Input: An input set S, a partial greedy solution G, and a threshold τ.
Output: A set $S' \subseteq S$ such that $f_G(e) \geq \tau$ for all $e \in S'$.

```
S' ← S
for e ∈ S do
    if f_G(e) < τ then S' ← S' \ {e}
return S'
```

2 A thresholding algorithm for submodular maximization

In the following algorithms, let $f : 2^V \to \mathbb{R}^+$ be a monotone submodular function, $n = |V|$, and $f_S(e) = f(S \cup \{e\}) - f(S)$. We refer to $f_S(e)$ as the *marginal* of e with respect to S. Let k be the maximum cardinality of the solution, and $m = \sqrt{n/k}$ be the number of machines.

2.1 A $1/2 - o(1)$ approximation in 2 rounds

First, we present a simple 1/2-approximation in 2 rounds, assuming we know the exact value of OPT. We will relax this assumption later. The algorithm requires two helper functions ThresholdGreedy and ThresholdFilter, which forms the basis of all of our algorithms. Roughly speaking, ThresholdGreedy greedily adds to a set of elements while there exists an element of high marginal in the input set. ThresholdFilter filters elements of low marginal out of the input set.

We define an additional function PartitionAndSample which simply initializes all of our algorithms by partitioning the input set randomly and drawing a random sample from it.

Using these three helper algorithms, our approximation algorithm is quite easy to implement, and can be found in Algorithm 4.

▶ **Lemma 1.** *The approximation ratio of Algorithm 4 is at least* 1/2.

Proof. The following lemma is folklore, but we present it for completeness.

First, we note that G_0 is the same on each machine so long as the loop iterating through S is done in a fixed order. We assume that this is the case. From this, it is clear that Algorithm 4 returns a set G for which $f_G(e) < \frac{OPT}{2k}$ for any $e \in V$.

Let G be the set returned at the end of the algorithm. Either $|G| = k$, or there is no $e \in V$ for which the marginal with respect to G is greater than $OPT/2$. In the former case,

[1] Note that S and the V_i are not stored on one machine by PartitionAndSample. We simply use the assignment to denote that the variables have been initialized and sent to their respective machines.

Algorithm 3: PARTITIONANDSAMPLE(V).

$S \leftarrow$ sample each $e \in V$ with probability $p = 4\sqrt{k/n}$
partition V randomly into sets $V_1, V_2, \ldots V_m$ to the m machines (one set per machine)
send S to each machine and a central machine C

Algorithm 4: A simple 2-round 1/2 approximation, assuming OPT is known.

round 1:
$S, V_1, \ldots, V_m \leftarrow$ PARTITIONANDSAMPLE(V)[1]
on *each machine* M_i *(in parallel)* **do**
  $\tau \leftarrow \frac{OPT}{2k}$
  $G_0 \leftarrow$ THRESHOLDGREEDY $(S, \emptyset, \tau)$
  if $|G_0| < k$ **then** $R_i \leftarrow$ THRESHOLDFILTER (V_i, G_0, τ)
  else $R_i \leftarrow \emptyset$
  send R_i to a central machine C
round 2 (only on C):
compute G_0 from S as in first round
$G \leftarrow$ THRESHOLDGREEDY $(\cup_i R_i, G_0, \tau)$
return G

we have k elements of value at least $\frac{OPT}{2k}$ so we are done. In the latter case, let O be the optimal solution. By monotonicity and submodularity,

$$OPT = f(O) \leq f(O \cup G) \leq f(G) + \sum_{e \in O \setminus G} f_G(e) \leq f(G) + k \cdot \frac{OPT}{2k}. \quad \blacktriangleleft$$

Lemma 1 shows that the algorithm is correct. Each machine in round 1 clearly uses $O(\sqrt{nk})$ memory. It remains to bound the memory of the central machine in round 2.

▶ **Lemma 2.** *With probability* $1 - e^{-\Omega(k)}$*, the number of elements sent to the central machine* C *has cardinality at most* $\sqrt{nk}$*.*

Proof. The expected number of elements in S is $4\sqrt{nk}$. By a Chernoff bound (Theorem 9) the probability that $|S| < 3\sqrt{nk}$ is at most $e^{-\Omega(\sqrt{nk})} \leq e^{-\Omega(k)}$. So we can assume that $|S| \geq 3\sqrt{nk}$. Let N_S denote the number of elements of marginal at least $OPT/(2k)$ with respect to G_0. The number of elements sent to C in round two is exactly $N_S + |S|$.

Consider breaking the sample set S into $3k$ blocks of size $\sqrt{n/k}$ and processing each block sequentially. If before each block, there are at least $\sqrt{nk}$ remaining elements of marginal value at least $OPT/(2k)$, we have probability at least $1 - \left(1 - \sqrt{\frac{k}{n}}\right)^{\sqrt{\frac{n}{k}}} > 1/2$ of adding an additional element to G_0. This happens conditioned on any prior history of the algorithm, since we can imagine that the blocks are sampled independently one at a time. Therefore, we can use a martingale argument to bound the number of elements selected in S. If X_i is the indicator random variable for the event that at least one element is selected from the i-th block, then we have $E[X_i \mid X_1, \ldots, X_{i-1}] \geq 1/2$. Hence we can define $Y_i = \sum_{j=1}^{i}(X_i - 1/2)$ and the sequence $Y_1, Y_2, \ldots$ is a submartingale, which means $E[Y_i \mid Y_1, \ldots, Y_{i-1}] \geq Y_{i-1}$. Moreover, $|Y_i - Y_{i-1}| \leq 1$. By Azuma's inequality (Theorem 10), $\Pr[Y_{3k} < -\frac{1}{2}k] < e^{-\Omega(k)}$. This means that with probability $1 - e^{-\Omega(k)}$, $\sum_{j=1}^{3k} X_j = Y_{3k} + \frac{3}{2}k \geq k$, and we include at least k elements overall. In that case, we are done and do not send anything to the central machine. Otherwise, the number of remaining elements of marginal value at least $OPT/(2k)$ drops below $\sqrt{nk}$. $\blacktriangleleft$

Remaining issues. Since we do not know the exact value of OPT, we will need to guess the value within a factor of ϵ without increasing the number of rounds. This will increase memory usage on the central machine by a factor of $\frac{1}{\epsilon}\log k$. To do this, we classify the inputs into two classes: when the input contains more than $\sqrt{nk}$ elements of value at least $\frac{OPT}{2k}$, and when there are less than $\sqrt{nk}$ such elements. We call the former class of inputs "dense" and the latter class "sparse". For each input class, we design a 1/2-approximation in 2 rounds. Given the input, we can run both in parallel and return the better of the two solutions: each machine simply runs both algorithms at the same time, keeping the number of machines the same. The full analysis is given in the Appendix, but we outline the algorithms below.

A 2-round algorithm for "dense" inputs

Let v be the maximum value of a single element of the random sample S in Algorithm 4. When the input is dense, v is likely to be at least $\frac{OPT}{2k}$ and at most OPT. A straightforward analysis shows that $\tau_j := v(1+\epsilon)^j$ is within a $(1+\epsilon)$ multiplicative factor of $OPT/2$ for some $j \in \{1, \ldots, \frac{1}{\epsilon}\log k\}$. Running Algorithm 4 with τ_j instead of $OPT/2$ produces an approximation of value at least $\frac{OPT}{2(1+\epsilon)} > \frac{OPT}{2}(1-\epsilon)$. Thus if each machine runs $\frac{1}{\epsilon}\log k$ copies of Algorithm 4, the best solution must have value at least $\frac{OPT}{2}(1-\epsilon)$.

A 2-round algorithm for "sparse" inputs

Call an element e "large" if $f(e) \geq \frac{OPT}{2k}$. The algorithm simply sends all the large elements of the input onto one machine and then runs a sequential algorithm in the second round. To get all the large elements onto one machine, we randomly partition the input set onto the m machines, and then send the $O(k)$ largest elements on each machine to the central machine. On the central machine, we can run the same thresholding procedure as in the "dense" case to find a threshold close to $OPT/(2k)$. We then run a sequential version of Algorithm 4.

In both the algorithms, ϵ can be taken to $\tilde{O}(\sqrt{k/n})$ without asymptotically increasing the memory, so we have a $(1/2 - o(1))$-approximation.

2.2 A $1 - \left(1 - \frac{1}{t+1}\right)^t$ approximation in $2t$ rounds

Here we show how our algorithm extends to t thresholds. The number of MapReduce rounds becomes $2t+2$. This can be reduced to $2t$ using tricks similar to Section 2.1, but we omit this here. The approximation factor with t thresholds is $1-(1-\frac{1}{t+1})^t$, which converges to $1-1/e$. We note that we need $\Theta(1/\epsilon)$ rounds to obtain a $(1-1/e-\epsilon)$-approximation, similar to Barbosa et al. [5], but in contrast we do not need any duplication of the ground set. Barbosa et al. does not specify the constant factor in $\Theta(1/\epsilon)$ but it seems that our dependence is better; a calculation yields that we need $(1+o(1))/\epsilon$ rounds to get a $(1-1/e-\epsilon)$-approximation.

For now, we assume (as in Algorithm 4) that we know the exact value of OPT. We deal with this assumption later. In a nutshell our algorithm works just like Algorithm 1 but with multiple thresholds used in a sequence. We set the threshold values as follows:

$$\alpha_\ell = \left(1 - \frac{1}{t+1}\right)^\ell \frac{OPT}{k}$$

for $1 \leq \ell \leq t$. (Note that for $t = 1$, we get $\alpha_1 = \frac{OPT}{2k}$ as in Algorithm 1.) For each threshold, we first select elements above the threshold from a random sample set, and then use this partial solution to prune the remaining elements. Finally, the solution at this threshold is completed on a central machine, and we proceed to the next threshold. The full description of the algorithm is presented in Algorithm 5. The analysis is as follows.

▶ **Lemma 3.** *The approximation ratio of Algorithm 5 is at least* $1 - \left(1 - \frac{1}{t+1}\right)^t$.

Proof. By induction, we prove the following statement: The value of the first $\frac{\ell}{t}k$ elements selected by the algorithm is at least $(1 - (1 - \frac{1}{t+1})^\ell)OPT$. (If $\frac{\ell}{t}k$ is not an integer, we count the marginal value of the $\lceil \frac{\ell}{k}k \rceil$-th selected element weighted by its respective fraction.)

Clearly this is true for $\ell = 0$. Assume that the claim is true for $\ell - 1$. We consider two cases.

Either all the elements among the first $\lceil \frac{\ell}{t}k \rceil$ are selected above the α_ℓ threshold. This means that since the value of the first $\frac{\ell-1}{t}k$ elements was at least $(1 - (1 - \frac{1}{t+1})^{\ell-1})OPT$, and the marginal value of each additional element is at least α_ℓ, the total value of the first $\frac{\ell}{k}OPT$ (with fractional elements counted appropriately) is at least

$$\left(1 - \left(1 - \frac{1}{t+1}\right)^{\ell-1}\right) OPT + \frac{1}{t} \cdot \left(1 - \frac{1}{t+1}\right)^{\ell} OPT = \left(1 - \left(1 - \frac{1}{t+1}\right)^{\ell}\right) OPT.$$

The other case is that not all these elements are selected above the α_ℓ threshold, which means that if we denote by S_ℓ the set of the first $\lfloor \frac{\ell}{t}k \rfloor$ selected elements, then there are no elements with marginal value more than α_ℓ with respect to S_ℓ. But then for the optimal solution O, we get

$$OPT - f(S_\ell) \leq f_{S_\ell}(O) \leq k\alpha_\ell = \left(1 - \frac{1}{t+1}\right)^{\ell} OPT$$

which means that $f(S_\ell) \geq (1 - (1 - \frac{1}{t+1})^\ell)OPT$.

For $\ell = t$, we obtain the statement of the lemma. ◀

The probabilistic analysis of the number of pruned elements that need to be sent to the central machine is exactly the same as in Section 2.1. The requirement of knowing OPT can be also handled in the same way — we can use an extra initial round to determine the maximum-value element on the input, which gives us an estimate of the optimum within a factor of k. Then we can try $O(\frac{1}{\epsilon}\log k)$ different estimates of OPT to ensure that one of them is within a relative error of $1 + \epsilon$ of the correct value. Finally, we use an extra final round to choose the best of the solutions that we found for different estimates of OPT. Alternatively, we can use additional tricks as in Section 2.1 to eliminate these 2 extra rounds, but we omit the details here.

3 Optimality of our choice of thresholds

Here we present a proof that there is no way to modify the thresholding algorithm and achieve a better approximation factor with a different choice of thresholds.

▶ **Theorem 4.** *The thresholding algorithm with t thresholds cannot achieve a factor better than* $1 - \left(1 - \frac{1}{t+1}\right)^t$.

Proof. Assume that the optimum O consists of k elements of total value kv^*. Since we are proving a hardness result, we can assume that the algorithm has this information and we can even let it choose v^*; in the following, we denote this choice $v^* = \alpha_0$. In addition, the algorithm chooses thresholds $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_t$. It might be the case that $\alpha_0 < \alpha_1$, but then we can ignore all the thresholds above α_0 and design our hard instance based on the thresholds below α_0, which would reduce to a case with fewer thresholds. Thus we can assume $\alpha_0 \geq \alpha_1 \geq \ldots \geq \alpha_t$.

Algorithm 5: A $2t$-round $1 - \left(\frac{t}{t+1}\right)^t$ approximation, assuming OPT is known.

$G \leftarrow \emptyset$
for $\ell = 1, \dots t$ **do**
  round $2\ell - 1$:
  $S, V_1, \dots, V_m \leftarrow$ PartitionAndSample(V)
  on *each machine* M_i *(in parallel)* **do**
    $G_0 \leftarrow$ ThresholdGreedy (S, G, α_ℓ)
    if $|G_0| < k$ **then** $R_i \leftarrow$ ThresholdFilter (V_i, G_0, α_ℓ)
    else $R_i \leftarrow \emptyset$
    send R_i to a central machine C
  round 2ℓ (only on C):
  compute G_0 from S as in first round
  $G \leftarrow$ ThresholdGreedy $(\cup_i R_i, G_0, \alpha_\ell)$
return G

We design an adversarial instance as follows. In addition to the k elements of value v^*, we have a set S of other elements where element i has value v_i, such that $\sum_{i \in S} v_i \leq k v^*$. The objective function is defined as follows: for $O' \subseteq O$ and $S' \subseteq S$,

$$f(S' \cup O') = \sum_{i \in S'} v_i + \left(1 - \frac{\sum_{i \in S'} v_i}{k v^*}\right) |O'| v^*.$$

It is easy to verify that this is a monotone submodular function. (It can be realized as a coverage function, which we leave as an exercise.)

Now we specify more precisely the values of elements in S. We will have n_ℓ elements of value α_ℓ, for each $1 \leq \ell \leq t$. The idea is that the algorithm will pick these n_ℓ elements at threshold value α_ℓ, at which point the marginal value of the optimal elements drops below α_ℓ, so we have to move on to the next threshold. A computation yields that we should have $n_\ell = (\frac{\alpha_{\ell-1}}{\alpha_\ell} - 1)k$.[2] The total value of these elements is $\sum_{i \in S} v_i = \sum_{\ell=1}^t n_\ell \alpha_\ell = \sum_{\ell=1}^t (\alpha_\ell - \alpha_{\ell-1})k = (\alpha_0 - \alpha_t)k \leq v^* k$ as required above.

Then, assuming that the marginal value of the optimum after processing $\ell - 1$ thresholds was $\alpha_{\ell-1}k$, the marginal value after processing the ℓ-th threshold will be $\alpha_{\ell-1}k - n_\ell \alpha_\ell = \alpha_\ell k$. By induction, the algorithm selects exactly n_ℓ elements of value α_ℓ, unless the constraint of k selected elements is reached. Let us denote by n'_ℓ the actual number of elements selected by the algorithm at threshold level α_ℓ. We have $n'_\ell \leq n_\ell$, and $\sum_{\ell=1}^t n'_\ell \leq k$, as discussed above.

The total value collected by the algorithm is $\sum_{\ell=1}^t n'_\ell \alpha_\ell$. Since we have $n'_\ell \leq n_\ell = (\frac{\alpha_{\ell-1}}{\alpha_\ell} - 1)k$, and $\alpha_\ell \leq \alpha_{\ell-1}$, we can define inductively $\alpha'_0 = \alpha_0$ and $\alpha'_\ell \geq \alpha_\ell$ such that $n'_\ell = (\frac{\alpha'_{\ell-1}}{\alpha'_\ell} - 1)k$. Then the total value collected by the algorithm is

$$\sum_{\ell=1}^t n'_\ell \alpha_\ell \leq \sum_{\ell=1}^t n'_\ell \alpha'_\ell = \sum_{\ell=1}^t (\alpha'_{\ell-1} - \alpha'_\ell)k = (\alpha'_0 - \alpha'_t)k.$$

Let us denote $\beta'_\ell = \frac{\alpha'_{\ell-1}}{\alpha'_\ell}$. We have $\sum_{\ell=1}^t (\beta'_\ell - 1) = \frac{1}{k} \sum_{\ell=1}^t n'_\ell \leq 1$, hence $\sum_{\ell=1}^t \beta'_\ell \leq$

[2] We ignore the issue that n_ℓ might not be an integer. For large k, it is easy to see that the rounding errors are negligible.

$t+1$. Recall that the value achieved by the algorithm is $\sum_{\ell=1}^{t} n'_\ell \alpha_\ell \leq (\alpha'_0 - \alpha'_t)k = (1 - 1/\prod_{\ell=1}^{t} \beta'_\ell)v^*k$. By the AMGM inequality, $\prod_{\ell=1}^{t} \beta'_\ell$ is maximized subject to $\sum_{\ell=1}^{t} \beta'_\ell \leq t+1$ when all the β'_ℓ are equal, $\beta'_\ell = \frac{t+1}{t}$. Then, the value achieved by the algorithm is $(1 - 1/\prod_{\ell=1}^{t} \beta'_\ell)v^*k = (1 - (\frac{t}{t+1})^t)OPT$. ◀

References

1 Sepehr Assadi and Sanjeev Khanna. Randomized Composable Coresets for Matching and Vertex Cover. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017.

2 Sepehr Assadi and Sanjeev Khanna. Tight Bounds on the Round Complexity of the Distributed Maximum Coverage Problem. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2412–2431, 2018. doi:10.1137/1.9781611975031.155.

3 Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone Submodular Maximization in Exponentially Fewer Iterations. *CoRR*, abs/1807.11462, 2018. arXiv:1807.11462.

4 Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An Exponential Speedup in Parallel Running Time for Submodular Maximization without Loss in Approximation. *CoRR*, abs/1804.06355, 2018. arXiv:1804.06355.

5 Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A New Framework for Distributed Submodular Maximization. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science*, 2016.

6 Satoru Fujishige. *Submodular functions and optimization*, volume 58. Elsevier, 2005.

7 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.

8 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast Greedy Algorithms in MapReduce and Streaming. *TOPC*, 2(3):14:1–14:22, 2015. doi:10.1145/2809814.

9 Andrew McGregor and Hoa T. Vu. Better Streaming Algorithms for the Maximum Coverage Problem. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 22:1–22:18, 2017. doi:10.4230/LIPIcs.ICDT.2017.22.

10 Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized Composable Core-sets for Distributed Submodular Maximization. In *ACM Symposium on Theory of Computing (STOC)*, pages 153–162, 2015.

11 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

12 Martin Raab and Angelika Steger. "Balls into Bins" — A Simple and Tight Analysis. In Michael Luby, José D. P. Rolim, and Maria Serna, editors, *Randomization and Approximation Techniques in Computer Science*, pages 159–170, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

Algorithm 6: A $1/2-\epsilon$ approximation in 2 rounds for "dense" inputs.

round 1:
$S, V_1, \ldots, V_m \leftarrow$ PartitionAndSample(V)
on *each machine* M_i *(in parallel)* **do**
  $v \leftarrow \max_{e \text{ on } M_i} f(\{e\})$
  for $j = 1, \ldots, \frac{1}{\epsilon}\log k$ **do**
    $\tau_j \leftarrow v(1+\epsilon)^j/k$
    $G_{0,j} \leftarrow$ ThresholdGreedy$(S, \emptyset, \tau_j)$
    if $|G_{0,j}| < k$ **then** $R_{i,j} \leftarrow$ ThresholdFilter$(V_i, G_{0,j}, \tau_j)$
    else $R_{i,j} \leftarrow \emptyset$
  send all $R_{i,j}$ to a central machine C
round 2 (only on C):
compute v, τ_j, and $G_{0,j}$ from S as in first round
for $j = 1, \ldots, \frac{1}{\epsilon}\log k$ **do**
  $G_j \leftarrow$ ThresholdGreedy$(\cup_i R_{i,j}, G_{0,j}, \tau_j)$
$G^\star = \text{argmax}_{G_j} f(G_j)$
return $G^\star$

A Appendix

In Algorithm 6, we design a 2-round $1/2-\epsilon$ approximation for "dense" inputs.

▶ **Lemma 5.** *Algorithm 6 returns a* $1/2-\epsilon$ *approximation.*

Proof. Note that Algorithm 6 essentially runs Algorithm 4 with $O(\frac{1}{\epsilon}\log k)$ guesses for $OPT/2$. To get a $1/2-\epsilon$ approximation, one of these guesses needs to be within a multiplicative factor of $1+\epsilon$ from $\frac{OPT}{2k}$. By the denseness assumption on the input, we know $\frac{OPT}{2k} \le v \le OPT$ with high probability. Suppose we try $j = 1, \ldots, \ell$ for some number of thresholds ℓ. For one of the τ_j's to be within a multiplicative factor of $1+\epsilon$ from $\frac{OPT}{2k}$, we require $v(1+\epsilon)^\ell/k < \frac{OPT}{2k} < v$. The upper bound is already satisfied by the denseness assumption, and the lower bound is achieved by taking ℓ for which $\ell \ge \log\left(\frac{OPT}{2kv\log(1+\epsilon)}\right) \ge \frac{1}{\epsilon}$. ◀

▶ **Lemma 6.** *The number of elements sent to the central machine is* $O\left(\frac{1}{\epsilon}\sqrt{nk}\log k\right)$.

Proof. This follows from Lemma 2 and the fact that there are only $\frac{\log k}{\epsilon}$ thresholds. ◀

Next, we design a 2-round $1/2-\epsilon$ approximation for "sparse" inputs (Algorithm 7).

▶ **Lemma 7.** *Algorithm 7 gives a* $1/2-\epsilon$ *approximation.*

Proof. There are two things to check: that one of the τ_j's is close to $OPT/2$, and that the machine C is not missing any of the elements that it needs.

For the former, its clear that by similar reasoning to Lemma 5, one of the τ_j's will be within a $1+\epsilon$ multiplicative factor to $OPT/2$.

For the latter, note that the "sparseness" assumption implies that with high probability, the $\sqrt{nk}$ large elements will be equally distributed among the machines, and each machine will get k elements in expectation. Since we send $O(k)$ elements to the central machine, C will have all the large elements in V with high probability. This can be shown by a standard balls-and-bins analysis [12]. ◀

Algorithm 7: A $1/2 - \epsilon$ approximation in 2 rounds for "sparse" inputs.

round 1:
partition V uniformly at random to the m machines
on *each machine* M_i **do**
  send its $O(k)$ largest elements to a central machine C
round 2 (only on C):
$S \leftarrow$ all elements sent to C
$v \leftarrow \max_{e \in S} f(\{e\})$
for $j = 1, \ldots, \frac{1}{\epsilon} \log k$ **do**
  $\tau_j \leftarrow v(1+\epsilon)^j / k$
  $G_j \leftarrow \textsc{ThresholdGreedy}\,(S, \emptyset, \tau_j)$
$G^\star = \operatorname{argmax}_{G_j} f(G_j)$
return $G^\star$

Finally, we note that the total memory use on C is $O(\sqrt{nk})$ since each machine sends $O(k)$ elements and there are $O(\sqrt{\frac{n}{k}})$ machines in total.

▶ **Theorem 8.** *By running Algorithms 6 and 7 in parallel, we have a 2-round* $1/2 - \epsilon$ *approximation.*

B Auxiliary Results

▶ **Theorem 9** (Chernoff bound)**.** *Let* $X_1, \ldots, X_n$ *be independent random variables such that* $X_i \in [0, 1]$ *with probability 1. Define* $X = \sum_{i=1}^{n} X_i$ *and let* $\mu = \mathbb{E}X$*. Then, for any* $\epsilon > 0$*, we have*

$$\Pr[X \geq (1+\epsilon)\mu] \leq \exp\left(-\frac{\min\{\epsilon, \epsilon^2\}\mu}{3}\right).$$

▶ **Theorem 10** (Azuma's inequality)**.** *Suppose* $X_1, \ldots, X_n$ *is a submartingale and* $|X_i - X_{i+1}| \leq c_i$*. Then, we have*

$$\Pr[X_n - X_0 \leq -t] \leq \exp\left(\frac{-t^2}{2\sum_i c_i^2}\right).$$

Compressed Sensing with Adversarial Sparse Noise via L1 Regression

Sushrut Karmalkar
Department of Computer Science, The University of Texas at Austin, 2317 Speedway, Austin, TX 78712, USA
sushrutk@cs.utexas.edu

Eric Price[1]
Department of Computer Science, The University of Texas at Austin, 2317 Speedway, Austin, TX 78712, USA
ecprice@cs.utexas.edu

Abstract

We present a simple and effective algorithm for the problem of *sparse robust linear regression*. In this problem, one would like to estimate a sparse vector $w^* \in \mathbb{R}^n$ from linear measurements corrupted by sparse noise that can arbitrarily change an adversarially chosen η fraction of measured responses y, as well as introduce bounded norm noise to the responses.

For Gaussian measurements, we show that a simple algorithm based on L1 regression can successfully estimate w^* for any $\eta < \eta_0 \approx 0.239$, and that this threshold is tight for the algorithm. The number of measurements required by the algorithm is $O(k \log \frac{n}{k})$ for k-sparse estimation, which is within constant factors of the number needed without any sparse noise.

Of the three properties we show – the ability to estimate sparse, as well as dense, w^*; the tolerance of a large constant fraction of outliers; and tolerance of adversarial rather than distributional (e.g., Gaussian) dense noise – to the best of our knowledge, no previous polynomial time algorithm was known to achieve more than two.

2012 ACM Subject Classification Theory of computation → Mathematical optimization

Keywords and phrases Robust Regression, Compressed Sensing

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.19

Acknowledgements The authors would like to thank the anonymous reviewers and Aravind Gollakota for helpful suggestions and comments about the writeup.

1 Introduction

Linear regression is the problem of estimating a signal vector from noisy linear measurements. It is a classic problem with applications in almost every field of science. In recent decades, it has also become popular to impose a sparsity constraint on the signal vector. This is known as "sparse recovery" or "compressed sensing", and (when the assumption holds) can lead to significant savings in the number of measurements required for accurate estimation.

A well-known problem with the most standard approaches to linear regression and compressed sensing is that they are not robust to outliers in the data. If even a single data point (x_i, y_i) is perturbed arbitrarily, the estimates given by the algorithms can also be perturbed arbitrarily far. Addressing this for linear regression is one of the primary focuses

[1] This work was done in part while the author was visiting the Simons Institute for the Theory of Computing.

of the field of robust statistics [9]. Unfortunately, while the problem is clear, the solution is not – no fully satisfactory robust algorithms exist, particularly for high-dimensional data.

In this paper, we consider the model of robustness in which only the responses y_i, not the features x_i, are corrupted by outliers. In this model, if the features x_i are i.i.d. normal, we show that the classic algorithm of L1 minimization performs well and has fairly high robustness, for both dense and sparse linear regression. In particular, we consider the observation model

$$y = Xw^* + \zeta + d \tag{1}$$

where $X \in \mathbb{R}^{m\times n}$ is the observation matrix, $w^* \in \mathbb{R}^n$ is the k-sparse signal, $\zeta \in \mathbb{R}^m$ is an ηm-sparse noise vector, and $d \in \mathbb{R}^m$ is a (possibly dense) noise vector. We will focus on the case of X having i.i.d. $N(0,1)$ entries, but the core lemmas and techniques can apply somewhat more generally.

Without adversarial corruptions – i.e., if $\eta = 0$ so $\zeta = 0$ – this would be the compressed sensing problem. The most standard solution for compressed sensing [4] is L1 minimization: if $m > \Theta(k \log \frac{n}{k})$ then with high probability

$$\widehat{w} := \operatorname*{arg\,min}_{\|y - Xw\|_2 \le \sigma} \|w\|_1$$

for any $\sigma > \|d\|_2$ will satisfy $\|\widehat{w} - w^*\|_2 \le O(\sigma/\sqrt{m})$. Unfortunately, this algorithm is not robust to sparse noise of large magnitude: a single faulty measurement y_i can make the $(\|y - Xw\|_2 \le \sigma)$ ball infeasible.

To make the algorithm robust to sparse measurement noise, a natural approach is to replace the (non-robust) ℓ_2 norm with the (robust) ℓ_1 norm, as well as to swap the objective and the constraint. This ensures that the constrained parameter does not involve outliers. In this paper we show that this approach works, i.e., we show that

$$\widehat{w} := \operatorname*{arg\,min}_{\|w\|_1 \le \lambda} \|y - Xw\|_1 \tag{2}$$

is a robust estimator for w^*. In the following theorem, we show that (2) is robust to any fraction of corruptions η less than $\eta_0 := 2\left(1 - \Phi(\sqrt{2\log 2})\right) \approx 0.239$, where $\Phi : \mathbb{R} \to [0,1]$ is the standard normal CDF. If $\lambda = \|w^*\|_1$, the reconstruction error is $O(\|d\|_1/m)$; for larger λ, it additionally grows with $\lambda - \|w^*\|_1$:

▶ **Theorem 1** (Sparse Case). *Let $\eta < \eta_0 - \epsilon$ where $\epsilon > 0$, and let $X \in \mathbb{R}^{m\times n}$ have i.i.d. $N(0,1)$ entries with $m > C\frac{\alpha^2}{\epsilon^2} k \log(\frac{en}{\alpha^2 \epsilon k})$ for some large enough constant C and parameter $\alpha \ge \frac{2}{\epsilon}$. Then with probability $1 - e^{-\Omega(\epsilon^2 m)}$ the matrix X will have the following property: for any $y = Xw^* + d + \zeta$ with $\|w^*\|_0 \le k$ and $\|\zeta\|_0 \le \eta m$,*

$$\widehat{w} := \operatorname*{arg\,min}_{\|w\|_1 \le \lambda} \|y - Xw\|_1$$

for $\lambda \ge \|w^*\|_1$ satisfies

$$\|w^* - \widehat{w}\|_2 \le O\left(\frac{1}{\epsilon - \frac{1}{\alpha}}\left(\frac{1}{m}\|d\|_1\right) + \frac{\lambda - \|w^*\|_1}{\alpha\sqrt{k}}\right).$$

In the case where w^* is not sparse, the reconstruction error is shown to be $O(\|d\|_1/m)$ in $O(n)$ samples using essentially the same proof.

▶ **Theorem 2** (Dense Case). *Let $\eta < \eta_0 - \epsilon$ where $\epsilon > 0$, and let $X \in \mathbb{R}^{m \times n}$ have i.i.d. $N(0,1)$ entries with $m > C\frac{n}{\epsilon^2}$ for some large enough constant C. Then with probability $1 - e^{-\Omega(\epsilon^2 m)}$ the matrix X will have the following property: for any $y = Xw^* + d + \zeta$ with $\|\zeta\|_0 \leq \eta m$,*

$$\widehat{w} := \arg\min_w \|y - Xw\|_1$$

satisfies

$$\|\widehat{w} - w^*\|_2 \leq O\left(\frac{\|d\|_1}{\epsilon m}\right)$$

Robustness threshold η_0.

We show in Section 6 that the threshold η_0 in Theorem 1 is tight for the algorithm: for any $\eta > \eta_0$, there exist problem instances where the algorithm given by (2) is not robust. It remains an open question whether any polynomial time algorithm can be robust for all $\eta < 0.5$.

1.1 Proof outline

Our main result follows from a simple analysis of the fact that for well-behaved matrices X, ℓ_1 regression recovers from adversarial corruptions. In this section we consider the illustrative case where there is no dense noise, in the limit of infinitely many samples. Let (X_g, y_g) and (X_b, y_b) denote the submatrices of (X, y) corresponding to the uncorrupted and corrupted samples respectively and let $\widehat{w}$ denote the solution of ℓ_1 regression. By definition $\widehat{w}$ satisfies

$$\|X\widehat{w} - y\|_1 \leq \|Xw^* - y\|_1.$$

Partitioning these 1-norms into terms corresponding to good and bad samples, we get

$$\begin{aligned} 0 &\geq \|X\widehat{w} - y\|_1 - \|Xw^* - y\|_1 \\ &= (\|X_g\widehat{w} - y_g\|_1 - \|X_g w^* - y_g\|_1) + (\|X_b\widehat{w} - y_b\|_1 - \|X_b w^* - y_b\|_1) \end{aligned}$$

Observe that since we have no dense noise, $X_g w^* = y_g$. An application of the triangle inequality then results in

$$\begin{aligned} 0 &\geq \|X_g(\widehat{w} - w^*)\|_1 + (\|X_b\widehat{w} - y_b\|_1 - \|X_b w^* - y_b\|_1) \\ &\geq \|X_g(\widehat{w} - w^*)\|_1 - \|X_b(\widehat{w} - w^*)\|_1 \end{aligned}$$

i.e.,

$$0 \geq \|X_g(\widehat{w} - w^*)\|_1 - \|X_b(\widehat{w} - w^*)\|_1 \tag{3}$$

We now show that as long as $\eta < \eta_0 - \epsilon$ for constant $\epsilon > 0$, the right hand side above is $\geq C(\epsilon)\|\widehat{w} - w^*\|_2$ for some $C(\epsilon) > 0$ whenever X is a Gaussian matrix. This will force $\widehat{w} = w^*$.

Equation (3) is minimized when the adversary corrupts the entries with the largest value for $|\langle x_i, w^* - \widehat{w}\rangle|$. For any vector v, observe that in the limit of infinitely many samples, the histogram of the entries of Xv is the same as that of $N(0, \|v\|_2^2)$. Let t be chosen such that $\frac{1}{\sqrt{2\pi}}\int_t^\infty e^{-\frac{x^2}{2}}dx = \frac{\eta}{2}$. This makes (3) proportional to

$$\|w^* - \widehat{w}\|_2 \left(\int_0^t xe^{-\frac{x^2}{2}}dx - \int_t^\infty xe^{-\frac{x^2}{2}}dx\right). \tag{4}$$

At $\eta = \eta_0$ the ℓ_1 norms of the largest η and $1-\eta$ fraction of samples drawn from a Gaussian distribution are equal, and hence (4) is 0. If $\eta < \eta_0 - \epsilon$ for constant ϵ, this difference is proportional to the standard deviation, i.e., $\|\widehat{w} - w^*\|_2$. Our proof proceeds by showing that a minor variant of this argument works even in the presence of dense noise, and in $O(k \log \frac{n}{k})$ samples the empirical ℓ_1 norms involved in the proof are close to the ℓ_1 norms of the Gaussian distribution.

1.2 Related Work

Classical robust statistics

The classical robust statistics literature on regression (see [9]) has developed a number of estimators with breakdown point 0.5 (i.e., that are robust for any $\eta < 0.5$). However, all such known estimators need time exponential in the data dimension n; the results also typically do not deal with sparsity in w^* and have distributional assumptions on the dense noise d. On the other hand, the results in this literature usually also protect against corruption in X_i, not just y_i; the L1 estimator is not robust to such corruptions.

Recent progress in robust statistics

There has been a lot of progress in the last year in the field of robust statistics, leading to polynomial time algorithms with positive breakdown points that are robust to corruptions in both X and y [5, 6, 13, 10]. However, these results all focus on the performance for small η (often required to be less than a non-explicit constant), do not consider sparse w^*, and have additional restrictions on the dense noise (typically that it be i.i.d. Gaussian, although [10] is somewhat more general). In Section 7 we empirically compare L1 regression to the algorithm of [5] for the dimension 1 case, and find that the algorithm seems to have the same breakdown point η_0 as L1 minimization under corruptions to y.

L1 minimization in statistics

Known as L1 minimization or Least Absolute Deviation, the idea of minimizing $\|y - Xw\|_1$ actually predates minimizing $\|y - Xw\|_2$, originating in the 18th century with Boscovich and Laplace [3]. It is widely known to be more robust to outliers in the y_i. However the extent to which this holds depends on the distribution of X. Surprisingly, we have not been able to find a rigorous analysis of L1 minimization for Gaussian X that simultaneously achieves these three features of our analysis: (1) an estimate of the breakdown point η_0 under corruptions to the y_i; (2) an extension of the algorithm to sparse w^*; or (3) a tolerance for adversarial d, rather than with a distributional assumption.

L1 minimization is typically dismissed in the statistics literature as being "inefficient" in the sense that, if the noise d is i.i.d. Gaussian, L1 minimization requires about 56% more samples than least squares [18] to achieve the same accuracy. However from the typical perspective of theoretical computer science, in which constant factors are less important than the avoidance of distributional assumptions, we find that L1 minimization is a very competitive algorithm.

L1 minimization in compressed sensing

Our error bound of $\|d\|_1/m$ is always better than the traditional $\|d\|_2/\sqrt{m}$ bound for compressed sensing. The two bounds match up to constant factors if the noise has a consistent magnitude, but our bound is significantly better if the noise is heavy tailed. The fact that our bound can drop the top η fraction of noise elements makes the distinction even more pronounced.

Robust regression in the presence of label corruptions

The past few years have featured a number of polynomial time algorithms for the problem considered in this paper, of sparse regression in the presence of sparse corruptions to the labels. As is typical in compressed sensing, there are approaches based on convex programming and on iterative methods.

One natural algorithm for the problem is to try to learn both the (sparse) signal and (sparse) noise, treating this as a single compressed sensing problem with the bigger "measurement matrix" of X atop a (scaled) identity matrix. With scaling $1/\lambda$, the standard L1 minimization approach to compressed sensing is equivalent to the following algorithm: minimize $\|w\|_1 + \lambda\|\zeta\|_1$ subject to $\|Xw + \zeta - y\|_2 \leq \epsilon$. If the adjoined measurement matrix satisfies an RIP-like property, then w (and ζ) will both be recovered.

Such an approach was first introduced in [11] with $\lambda = 1$, giving an algorithm that could tolerate up to about $\eta \approx 1/(\log n)$ fraction sparse corruptions. This was then improved by [12] by setting $\lambda = \frac{1}{\sqrt{\log(en/m)}}$, improving the breakdown point η to an unspecified constant; naively following the proof would give a value below 1%. We suspect that this approach – which recovers ζ as well as w – does not have a breakdown point close to η_0.

The second class of algorithms for the problem are based off iterative hard thresholding, where in each iteration one ignores the samples that make a large error with the ℓ_2 minimizer. In [2] it was shown that without any dense noise, this yields exact recovery with a breakdown point of $\eta < 0.015$. [1] provided an algorithm that can handle dense Gaussian noise, but the perturbations are required to be oblivious to the matrix X and the breakdown point is 0.0001.

Another line of work, including [14, 8, 15], considers non-adversarial corruption. For example, if the corruptions are in random locations, and the signs of the signal vector are random, then one can tolerate corruption of nearly 100% of the y_i [14]. Finally, [16] considers the (essentially equivalent) LASSO version of our proposed algorithm (2), and shows that it is robust to i.i.d. heavy-tailed median-zero noise.

Thus, for sparse regression with both adversarial corruption of the labels and dense noise, no previous polynomial-time algorithm had a breakdown point above 0.015. We improve that to 0.239 with a simple algorithm.

LP Decoding and Privacy

Very closely related to our work is that of [7], which gets very similar results to our dense-case results (Theorem 2) in the service of a privacy application. This work observes the same threshold η_0 as we do for the same L1-regression algorithm, but with a somewhat weaker error guarantee (requiring a bound on $\|d\|_\infty$ not $\|d\|_1$). [7] also proves that if X is i.i.d. ± 1 rather than Gaussian, the breakdown point would be positive but strictly below η_0. The subsequent work [17] also observes that ℓ_p regression for $p < 1$ would yield greater breakdown points than η_0 for Gaussian X, similar to our Section 5.

2 Definitions and notation

We start by defining a notion of robustness that we will use later. A matrix X is said to be (η, q)-robust if for any submatrix consisting of an η fraction of the rows, the ℓ_q norm of the submatrix times a unit vector is upper bounded by a constant times $m^{1/q}$. Also, for any submatrix consisting of a $1 - \eta$ fraction of the rows, the ℓ_q norm of the submatrix times a unit vector is *lower bounded* by a constant times $m^{1/q}$.

▶ **Definition 3.** A matrix $X \in \mathbb{R}^{m\times n}$ is said to be (η, q)-*robust* with respect to $U \subset \mathbb{R}^n$ if there exist constants $S^{\max}_{U,\eta}$ and $S^{\min}_{U,\eta}$ satisfying the following conditions for all $v \in U$.

$$\max_{\substack{S\subset[m]\\|S|\le\eta m}} \|(Xv)_S\|_q^q \le m \cdot S^{\max}_{U,\eta} \cdot \|v\|_2$$

$$\min_{\substack{S\subset[m]\\|S|\ge(1-\eta)m}} \|(Xv)_S\|_q^q \ge m \cdot S^{\min}_{U,\eta} \cdot \|v\|_2.$$

We now define some notation. $S^*_{k,*}$ and S^*_η will be used to refer to the robustness constants with respect to k-sparse vectors and $\mathbb{R}^n$ respectively. v_T will denote the vector v with all entries whose indices are outside T set to 0.

We use Φ to denote the CDF of $N(0,1)$. $B(\gamma)$ and $G(\gamma)$ will be used to refer to the ℓ_1 norm of the largest (in absolute value) γ fraction and the smallest $1-\gamma$ fraction with respect to the Gaussian distribution respectively, i.e.,

$$B(\gamma) = \frac{2}{\sqrt{2\pi}} \int_{\Phi^{-1}(1-\frac{\gamma}{2})}^{\infty} z e^{-\frac{z^2}{2}} dz = \sqrt{\frac{2}{\pi}} \left(e^{-(\mathrm{erf}^{-1}(1-\gamma))^2} \right)$$

and

$$G(\gamma) = \frac{2}{\sqrt{2\pi}} \int_{0}^{\Phi^{-1}(1-\frac{\gamma}{2})} z e^{-\frac{z^2}{2}} dz = \sqrt{\frac{2}{\pi}} \left(1 - e^{-(\mathrm{erf}^{-1}(1-\gamma))^2} \right).$$

Define η_0 to be the largest η such that $G(\eta) \ge B(\eta)$. Using the expressions above, one can solve for η to get $\eta_0 = 2(1-\Phi(\sqrt{2\log 2})) \approx 0.239$.

Also, we will use $f(x) \lesssim g(x)$ to mean there are constants X and C such that $\forall x > X. |f(x)| \le C|g(x)|$.

3 Robustness of Gaussian matrices

In the following lemma, we show that Gaussian matrices are $(\eta, 1)$-robust with constants in terms of $B(\cdot), G(\cdot)$ defined earlier.

▶ **Lemma 4.** *Let X be an $m \times n$ Gaussian matrix, where $m \ge \frac{C}{\epsilon^2} \cdot \left(k \log \frac{en}{k\epsilon} + \log \frac{1}{\delta}\right)$ for a large enough constant C and $\epsilon < 1$. Then with probability $1-\delta$, X is $(\eta, 1)$ robust with constants*

$$S^{\min}_{k,\eta} = G(\eta - \epsilon) - \epsilon$$

$$S^{\max}_{k,\eta} = B(\eta + \epsilon) + \epsilon.$$

Proof. Rearranging terms in the definition we see that we would like to show

$$\frac{1}{m} \max_{\substack{S\subset[m]\\|S|\le\eta m}} \left\| \left(X \cdot \frac{v}{\|v\|} \right)_S \right\|_1 \le B(\eta+\epsilon) + \epsilon$$

$$\frac{1}{m} \min_{\substack{S\subset[m]\\|S|\ge(1-\eta)m}} \left\| \left(X \cdot \frac{v}{\|v\|} \right)_S \right\|_1 \ge G(\eta-\epsilon) - \epsilon.$$

Without loss of generality it is sufficient to prove the above for all (k-sparse) unit vectors. Let x_i denote the i^{th} row of X and let $S_v = \{\langle x_i, v\rangle \mid i \in m\}$. Note that S_v look like samples

from $N(0,1)$ for any fixed unit vector v. Before we continue, we define some notation. Let $\widehat{G}_v(\eta)$ denote the smallest possible ℓ_1 norm of a subset of S_v of size $(1-\eta)m$ and let $\widehat{B}_v(\eta)$ be defined similarly to denote the largest possible ℓ_1 norm of any subset of size ηm. What we want to prove is

$$\widehat{B}_v(\eta) < B(\eta+\epsilon)+\epsilon$$

and

$$\widehat{G}_v(\eta) > G(\eta-\epsilon)-\epsilon$$

for all k-sparse unit vectors v. To do this, we will first prove that the relationship holds with high probability for all k-sparse unit vectors in a fine enough net on the sphere, and then say that the deviation cannot be very large for points outside the net.

We will need the following fact proven in Appendix A. Here $\widehat{G}(\eta)$ refers to the ℓ_1 norm of the smallest $(1-\eta)$ fraction of S with respect to the uniform distribution and $\widehat{B}(\eta)$ refers to the ℓ_1 norm of the largest η fraction of S with respect to the uniform distribution.

▶ **Fact 5.** *Let $S = \{z_1, \ldots, z_m\}$ be i.i.d. samples from $N(0,1)$. Then with probability $1 - O\left(e^{\frac{m\epsilon^2}{2}}\right)$,*

$$\widehat{G}(\eta) > G(\eta-\epsilon)-\epsilon$$

and

$$\widehat{B}(\eta) < B(\eta+\epsilon)+\epsilon$$

For now, let v be a fixed vector and let $\tau > 0$ be a parameter. Define the following bad events

$$\mathcal{G}_v = \left\{\widehat{G}_v(\eta) < G(\eta-\epsilon)-\epsilon\right\}$$

$$\mathcal{B}_v = \left\{\widehat{B}_v(1-\eta) > B(1-\eta+\epsilon)+\epsilon\right\}$$

$$\mathcal{N} = \left\{\forall i \in [m], \|x_i\|_2 > \sqrt{n+\tau}\right\}.$$

These events correspond to either the ℓ_1 norms of the smallest $1-\eta$ fraction or the largest η fraction not being close enough to the expectation, or the 2-norm of the Gaussian vectors not being close enough to texpectation. Applications of Fact 5 for η and $1-\eta$, and concentration for χ^2 random variables then implies

$$\Pr\left(\mathcal{G}_v \vee \mathcal{B}_v\right) \lesssim e^{\frac{m\epsilon^2}{2}}$$

and

$$\Pr\left(\mathcal{N}\right) \lesssim m e^{-\frac{n\tau^2}{8}}.$$

For a unit ℓ_2 ball in a k-dimensional subspace of $\mathbb{R}^n$, there exists a γ-net of size $(1+\frac{2}{\gamma})^k < (\frac{3}{\gamma})^k$. Let C be the union of these nets over all subspaces corresponding to k-sparse vectors. A union bound now gives us

$$\Pr\left(\exists v \in C : \mathcal{G}_v \vee \mathcal{B}_v\right) \lesssim \binom{n}{k}\left(\frac{3}{\gamma}\right)^k \left(e^{\frac{m\epsilon^2}{2}}\right)$$

We will now move from the net to the union of all k-sparse unit balls. Let $u \in \mathbb{R}^n$ be a k-sparse unit vector. Then for any t, there exist $v_0, \ldots, v_t \in C$ having the same support as u and a unit vector d also having the same support as u, such that

$$u = \sum_{i=0}^{t} \gamma^i v_i + \gamma^{t+1} d.$$

This follows from choosing v_0 to be the closest point in the net to u, choosing v_1 to be the closest point in the net to $(u - v_0)/\gamma$ and so on.

Let $U \subset [m]$ be the set of indices of X corresponding to the smallest (in absolute value) $(1-\eta)$ fraction of elements of S_u. Conditioning on the bad events not happening (i.e., on the event $\overline{(\exists v \in C. \mathcal{G}_v \vee \mathcal{B}_v) \vee \mathcal{N}}$) we see

$$\begin{aligned}
\widehat{G}_u(\eta) &= \frac{1}{m} \sum_{i \in U} \left| \left\langle x_i, \sum_{j=0}^{t} \gamma^j v_j + \gamma^{t+1} d \right\rangle \right| \\
&\geq \frac{1}{m} \sum_{i \in U} |\langle x_i, v_0 \rangle| - \frac{1}{m} \sum_{i \in U} \left| \left\langle x_i, \sum_{j=1}^{t} \gamma^j v_j + \gamma^{t+1} d \right\rangle \right| \\
&\geq \frac{1}{m} \sum_{i \in U} |\langle x_i, v_0 \rangle| - \sum_{j=1}^{t} \left(\frac{\gamma^j}{m} \sum_{i \in U} |\langle x_i, v_j \rangle| \right) - \frac{\gamma^{t+1}}{m} \sum_{i \in U} |\langle x_i, d \rangle| \\
&\geq \widehat{G}_{v_0}(\eta) - \sum_{j=1}^{t} \widehat{B}_{v_j}(1-\eta)\gamma^j - \frac{\gamma^{t+1}}{m} \sum_{i \in U} \|x_i\| \|d\| \\
&\geq (G(\eta - \epsilon) - \epsilon) - (B(1 - \eta + \epsilon) + \epsilon) \sum_{j=1}^{t} \gamma^j - \gamma^{t+1} \sqrt{n + \tau} \\
&\geq G(\eta - \epsilon) - \epsilon - 2\gamma(B(1 - \eta + \epsilon) + \epsilon) - \gamma^{t+1} \sqrt{n + \tau} \\
&\geq G(\eta - \epsilon) - \epsilon - 4\gamma - \gamma^{t+1} \sqrt{n + \tau} \\
&\geq G(\eta - \epsilon) - 2\epsilon
\end{aligned}$$

The first few inequalities are a consequence of the definitions of G_v and B_v and the third inequality follows from an application of Cauchy-Schwartz. The second to last inequality follows by noting that $\epsilon < 1$ and $B(1 - \eta + \epsilon) < 1$, and the final inequality follows by setting $t > \log \frac{n+\tau}{\epsilon}$ and $\gamma = \frac{\epsilon}{10}$. This means

$$\begin{aligned}
&\Pr\left(\text{There exists a } k\text{-sparse unit } v \text{ such that } G(\eta - \epsilon) - \widehat{G}_v(\eta) > 2\epsilon\right) \\
&\lesssim \binom{n}{k} \left(\frac{30}{\epsilon}\right)^k \left(e^{-\frac{m\epsilon^2}{2}}\right) + m e^{-\frac{n\tau^2}{8}} \\
&\lesssim e^{k \log \frac{en}{k} + k \log\left(\frac{30}{\epsilon}\right) - \frac{m\epsilon^2}{2}} + m e^{-\frac{n\tau^2}{8}}
\end{aligned}$$

Setting $\tau = 10mn \log \frac{1}{\delta}$ and $m \gtrsim \frac{1}{\epsilon^2} \cdot \left(k \log \frac{en}{k\epsilon} + \log \frac{1}{\delta}\right)$ makes the bound on the probability above $\lesssim \delta$. The result now follows by rescaling ϵ and δ appropriately. ◀

The previous lemma showed that the Gaussian matrix is robust with respect to truly k-sparse vectors. However, we will need to show that it is robust with respect to $(w^* - \widehat{w})$, i.e., the difference between the true vector and the solution of ℓ_1 regression. To do this, we will use a standard shelling argument to transfer upper and lower bounds for the

restricted eigenvalues over $(1+\alpha^2)k$-sparse vectors to the restricted eigenvalues over the cone $V_S = \{v \in \mathbb{R}^n \mid \Delta + \|v_S\|_1 \geq \|v_{\overline{S}}\|_1\}$ for some S satisfying $|S| = k$, which is the cone in which this difference lies. This is the content of the following lemma from Appendix B.

▶ **Lemma 6** (Shelling Argument). *Let $A \in \mathbb{R}^{m\times n}$ satisfy*

$$L\|v\|_2 \leq \|Av\|_1 \leq U\|v\|_2$$

for all $(1+\alpha^2)k$-sparse vectors v. If $S \subset [m]$ is fixed and of cardinality k, then A satisfies

$$\frac{L}{1+\alpha}\left(\alpha - \frac{U}{L}\right)\|v\|_2 - \frac{2U\Delta}{\alpha\sqrt{k}} \leq \|Av\|_1 \leq U\left(1+\frac{1}{\alpha}\right)\|v\|_2 + \frac{U\Delta}{\alpha\sqrt{k}}$$

for all

$$v \in V_S = \{v \in \mathbb{R}^n \mid \Delta + \|v_S\|_1 \geq \|v_{\overline{S}}\|_1\}.$$

We can now prove the main lemma which will be used to say that Gaussian matrices are robust with respect to the vector $w^* - \widehat{w}$.

▶ **Lemma 7** (Main Lemma). *Let $\eta < \eta_0$, $\epsilon \in (0,1)$, $\alpha > 1$ and $\Delta > 0$ be free parameters, and let $S \subset [n]$ be a fixed subset of size k. Let $X \in \mathbb{R}^{m\times n}$ be a matrix with entries drawn from $N(0,1)$ and suppose $m > \frac{C}{\epsilon^2}\cdot\left(k\alpha^2 \log\frac{en}{k\alpha^2\epsilon} + \log\frac{1}{\delta}\right)$ for some large enough constant C. Then with probability $1-\delta$, for all*

$$v \in V_S = \{v \in \mathbb{R}^n \mid \Delta + \|v_S\|_1 \geq \|v_{\overline{S}}\|_1\}$$

and for all $T \subset [m]$ such that $|T| \leq \eta m$,

$$\|(Xv)_{\overline{T}}\|_1 - \|(Xv)_T\|_1 \gtrsim m\|v\|_2\left((G(\eta-\epsilon) - B(\eta+\epsilon) - 2\epsilon) - \frac{1}{\alpha}\right) - \frac{m\Delta}{\alpha\sqrt{k}}.$$

Proof. The matrix X is both $(\eta, 1)$ and $(1-\eta, 1)$ robust for all $k(1+\alpha^2)$-sparse vectors. An application of Lemma 3.3 for any submatrix A of X consisting of an η fraction of it's rows gives us

$$\|Av\|_1 \leq mS^{\max}_{k(1+\alpha^2),\eta}\left(1+\frac{1}{\alpha}\right)\|v\|_2 + \frac{mS^{\max}_{k(1+\alpha^2),\eta}\Delta}{\alpha\sqrt{k}}.$$

This proves

$$\max_{\substack{S\subset[m]\\|S|\leq\eta m}} \|(Xv)_S\|_1 \leq mS^{\max}_{k(1+\alpha^2),\eta}\left(1+\frac{1}{\alpha}\right)\|v\|_2 + \frac{mS^{\max}_{k(1+\alpha^2),\eta}\Delta}{\alpha\sqrt{k}}$$

A similar application proves that for any matrix A consisting of a $(1-\eta)$ fraction of the rows of X, we get

$$\|Av\|_1 \geq \frac{mS^{\min}_{k(1+\alpha^2),\eta}}{1+\alpha}\left(\alpha - \frac{S^{\max}_{k(1+\alpha^2),1-\eta}}{S^{\min}_{k(1+\alpha^2),\eta}}\right)\|v\|_2 - \frac{2mS^{\max}_{k(1+\alpha^2),\eta}\Delta}{\alpha\sqrt{k}}$$

i.e.,

$$\min_{\substack{S\subset[m]\\|S|\geq(1-\eta)m}} \|(Xv)_{\overline{S}}\|_1 \geq \frac{mS^{\min}_{k(1+\alpha^2),\eta}}{1+\alpha}\left(\alpha - \frac{S^{\max}_{k(1+\alpha^2),1-\eta}}{S^{\min}_{k(1+\alpha^2),\eta}}\right)\|v\|_2 - \frac{2mS^{\max}_{k(1+\alpha^2),\eta}\Delta}{\alpha\sqrt{k}}$$

To complete the proof, we now estimate the parameters involved. If $\eta < \eta_0$ and $\epsilon < 1$ and the underlying matrix is an $m \times n$ Gaussian matrix where $m \geq \frac{C}{\epsilon^2} \cdot \left(k\alpha^2 \log \frac{en}{k\alpha^2\epsilon} + \log \frac{1}{\delta}\right)$, Lemma 4 yields

$$S^{\max}_{k(1+\alpha^2),1-\eta} < B(1-\eta+\epsilon) + \epsilon < 1 + 1 = 2.$$

Similar applications of Lemma 4 give us that $S^{\max}_{k(1+\alpha^2),\eta}$ and $\frac{S^{\max}_{k(1+\alpha^2),1-\eta}}{S^{\min}_{k(1+\alpha^2),\eta}}$ are also upper bounded by constants. This results in the following bounds

$$\max_{\substack{S\subset[m]\\|S|\leq\eta m}} \|(Xv)_S\|_1 \leq m S^{\max}_{k(1+\alpha^2),\eta}\|v\|_2\left(1+\frac{1}{\alpha}\right) + \frac{2m\Delta}{\sqrt{k\alpha}},$$

$$\min_{\substack{S\subset[m]\\|S|\geq(1-\eta)m}} \|(Xv)_S\|_1 \geq m S^{\min}_{k(1+\alpha^2),\eta}\|v\|_2\left(\frac{\alpha-C}{\alpha+1}\right) - \frac{4m\Delta}{\sqrt{k\alpha}}.$$

By taking the difference of the above inequalities and simplyfing, we get the following for any $T \subset [m]$ such that $|T| \leq \eta m$

$$\begin{aligned}
&\|(Xv)_{\overline{T}}\|_1 - \|(Xv)_T\|_1 \\
&\geq m\|v\|_2\left(S^{\min}_{k(1+\alpha^2),\eta}\left(\frac{\alpha-C}{\alpha+1}\right) - S^{\max}_{k(1+\alpha^2),\eta}\left(1+\frac{1}{\alpha}\right)\right) - \frac{6m\Delta}{\alpha\sqrt{k}} \\
&\geq m\|v\|_2\left(S^{\min}_{k(1+\alpha^2),\eta} - S^{\max}_{k(1+\alpha^2),\eta} - \left(S^{\min}_{k(1+\alpha^2),\eta} + S^{\max}_{k(1+\alpha^2),\eta}\right)\frac{C+1}{\alpha}\right) - \frac{6m\Delta}{\alpha\sqrt{k}} \\
&\geq m\|v\|_2\left((G(\eta-\epsilon) - B(\eta+\epsilon) - 2\epsilon) - \frac{4(C+1)}{\alpha}\right) - \frac{6m\Delta}{\alpha\sqrt{k}} \\
&\geq m\|v\|_2\left((G(\eta-\epsilon) - B(\eta+\epsilon) - 2\epsilon) - \frac{1}{\alpha}\right) - \frac{m\Delta}{\alpha\sqrt{k}}
\end{aligned}$$

◀

4 Proof of main theorem

▶ **Theorem 1** (Sparse Case). *Let $\eta < \eta_0 - \epsilon$ where $\epsilon > 0$, and let $X \in \mathbb{R}^{m\times n}$ have i.i.d. $N(0,1)$ entries with $m > C\frac{\alpha^2}{\epsilon^2}k\log(\frac{en}{\alpha^2\epsilon k})$ for some large enough constant C and parameter $\alpha \geq \frac{2}{\epsilon}$. Then with probability $1 - e^{-\Omega(\epsilon^2 m)}$ the matrix X will have the following property: for any $y = Xw^* + d + \zeta$ with $\|w^*\|_0 \leq k$ and $\|\zeta\|_0 \leq \eta m$,*

$$\widehat{w} := \arg\min_{\|w\|_1\leq\lambda} \|y - Xw\|_1$$

for $\lambda \geq \|w^*\|_1$ satisfies

$$\|w^* - \widehat{w}\|_2 \leq O\left(\frac{1}{\epsilon - \frac{1}{\alpha}}\left(\frac{1}{m}\|d\|_1\right) + \frac{\lambda - \|w^*\|_1}{\alpha\sqrt{k}}\right).$$

Proof. Let X_g and X_b denote X restricted to the rows that are not corrupted, and to the rows that are corrupted respectively. Let y_g and y_b denote the corresponding y terms. By the definition of $\widehat{w}$ and noting that w^* is feasible for the program,

$$\begin{aligned}
0 &\geq \|X\widehat{w} - y\|_1 - \|Xw^* - y\|_1 \\
&= (\|X_g\widehat{w} - y_g\|_1 - \|X_gw^* - y_g\|_1) + (\|X_b\widehat{w} - y_b\|_1 - \|X_bw^* - y_b\|_1) \\
&\geq \|X_g(\widehat{w} - w^*)\|_1 - 2\|X_gw^* - y_g\|_1 + (\|X_b\widehat{w} - y_b\|_1 - \|X_bw^* - y_b\|_1) \\
&\geq \|X_g(\widehat{w} - w^*)\|_1 - 2\|X_gw^* - y_g\|_1 - \|X_b(\widehat{w} - w^*)\|_1
\end{aligned}$$

Where the second equality follows from $\|Xv - y\|_1 = \|X_g v - y_g\|_1 + \|X_b v - y_b\|_1$, and the inequalities are just applications of the triangle inequality. Rearranging terms now gives us

$$2\|X_g w^* - y_g\|_1 \geq \|X_g(\widehat{w} - w^*)\|_1 - \|X_b(\widehat{w} - w^*)\|_1 \tag{5}$$

Let $\widehat{w} = w^* + h$ and let S be the support of w^*. Then

$$\begin{aligned} \lambda &\geq \|\widehat{w}\|_1 \\ &= \|h + w^*\|_1 \\ &\geq \|w^*\|_1 + \|h_{\overline{S}}\|_1 - \|h_S\|_1 \\ \implies (\lambda - \|w^*\|_1) + \|h_S\|_1 &\geq \|h_{\overline{S}}\|_1 \end{aligned}$$

Setting $\Delta = (\lambda - \|w^*\|_1)$ and T to be the set of corrupted indices in Lemma 7 implies that if $m \gtrsim \frac{1}{\epsilon^2} \cdot \left(k\alpha^2 \log \frac{en}{k\alpha^2\epsilon} + \log \frac{1}{\delta}\right)$, then with probability $1 - \delta$

$$\begin{aligned} &\|X_g(\widehat{w} - w^*)\|_1 - \|X_b(\widehat{w} - w^*)\|_1 \\ &= \|(Xh)_{\overline{T}}\|_1 - \|(Xh)_T\|_1 \\ &\gtrsim m\|h\|_2 \left(G\left(\eta - \frac{\epsilon}{2}\right) - B\left(\eta + \frac{\epsilon}{2}\right) - \epsilon - \frac{1}{\alpha}\right) - \frac{m(\lambda - \|w^*\|_1)}{\alpha\sqrt{k}} \end{aligned}$$

Combining this with (5), as long as the coefficient of $\|h\|_2$ is positive, we get

$$\|\widehat{w} - w^*\|_2 \lesssim \frac{1}{\left(G\left(\eta - \frac{\epsilon}{2}\right) - B\left(\eta + \frac{\epsilon}{2}\right) - \epsilon - \frac{1}{\alpha}\right)} \left(\frac{\|d\|_1}{m} + \frac{(\lambda - \|w^*\|_1)}{\alpha\sqrt{k}}\right) \tag{6}$$

It turns out

$$\left(G\left(\eta - \frac{\epsilon}{2}\right) - B\left(\eta + \frac{\epsilon}{2}\right) - \epsilon\right) \gtrsim \epsilon.$$

This follows by a simple lower bound via the Taylor expansion of $1 - 2e^{-(\mathrm{erf}^{-1}((1-\eta_0)+x))}$ around $x = 0$.

$$\begin{aligned} G\left(\eta - \frac{\epsilon}{2}\right) - B\left(\eta + \frac{\epsilon}{2}\right) - \epsilon &\geq \sqrt{\frac{2}{\pi}}\left(1 - 2e^{-(\mathrm{erf}^{-1}(1-\eta+\frac{\epsilon}{2}))^2}\right) - \epsilon \\ &= \sqrt{\frac{2}{\pi}}\left(1 - 2e^{-(\mathrm{erf}^{-1}(1-\eta_0+(\epsilon+\frac{\epsilon}{2})))^2}\right) - \epsilon \\ &\geq 3\sqrt{2\log 2} \cdot \epsilon - \epsilon \\ &\gtrsim \epsilon \end{aligned}$$

i.e.,

$$G\left(\eta - \frac{\epsilon}{2}\right) - B\left(\eta + \frac{\epsilon}{2}\right) - \epsilon - \frac{1}{\alpha} \gtrsim \epsilon - \frac{1}{\alpha} \tag{7}$$

Substituting our terms back into (6) gives us,

$$O\left(\frac{1}{\epsilon - \frac{1}{\alpha}} \cdot \frac{\|d\|_1}{m} + \frac{\lambda - \|w^*\|_1}{\alpha\sqrt{k}}\right) \geq \|\widehat{w} - w^*\|_2. \qquad \blacktriangleleft$$

We also note that in the case that w^* is not sparse, one can directly use Lemma 4 once we get to (5) and continue the proof from there. This results in the following theorem.

▶ **Theorem 2** (Dense Case). *Let $\eta < \eta_0 - \epsilon$ where $\epsilon > 0$, and let $X \in \mathbb{R}^{m \times n}$ have i.i.d. $N(0,1)$ entries with $m > C\frac{n}{\epsilon^2}$ for some large enough constant C. Then with probability $1 - e^{-\Omega(\epsilon^2 m)}$ the matrix X will have the following property: for any $y = Xw^* + d + \zeta$ with $\|\zeta\|_0 \leq \eta m$,*

$$\widehat{w} := \arg\min_w \|y - Xw\|_1$$

satisfies

$$\|\widehat{w} - w^*\|_2 \leq O\left(\frac{\|d\|_1}{\epsilon m}\right)$$

Note that if there is no dense noise (i.e., $d = 0$), the above theorem immediately gives exact recovery when the fraction of corruptions is $\eta < \eta_0 - \epsilon$.

5 ℓ_p regression for $0 < p < 1$

Define ℓ_p regression to be the problem of recovering a signal by minimizing the p^{th} power of the ℓ_p norm, i.e.,

$$\widehat{w} = \arg\min_v \sum_{i=1}^{m} |\langle x_i, v\rangle - y_i|^p.$$

Observe that $0 < p < 1$ implies

$$\left(\sum_i a_i\right)^p \leq \sum_i a_i^p.$$

This allows a proof similar to that of Theorem 1 to go through. We make the following claim.

▶ **Claim 8.** *Let X be an (η, p)-robust matrix where $p \in (0, 1]$. Then for any $\eta < \alpha$ the solution of ℓ_p regression, $\widehat{w}$ satisfies*

$$\frac{1}{(S_\eta^{\min} - S_\eta^{\max})} \cdot \frac{\|d\|_p}{m} \gtrsim \|\widehat{w} - w^*\|_2^p,$$

where $\|d\|_p = \sum_{i=1}^m |d_i|^p$ and α is the threshold below which $(S_\alpha^{\min} - S_\alpha^{\max}) > 0$ begins to hold.

If X is a Gaussian matrix, then as $p \to 0$, in the limit of a large number of samples, the value of η at which the condition

$$(S_\eta^{\min} - S_\eta^{\max}) > 0$$

begins to hold goes from η_0 to 0.5. We plot the breakdown point against the norm in Figure 1. Unfortunately, ℓ_p regression in general seems to be NP-hard as well as approximation resistant.

6 Lower bounds

In this section, we show that for the case of adversarial dense noise our results are tight for the ℓ_1 regression algorithm. Recall our notation: X is the matrix of x_i, $y = Xw^* + \zeta + d$ where $\|\zeta\|_0 \leq \eta m$ and d is the dense noise and v_T denotes the vector v with all entries with indices outside T set to 0.

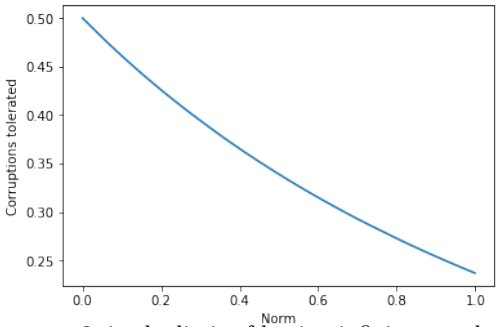


Figure 1 As the norm goes to 0, in the limit of having infinite samples, ℓ_p regression can tolerate almost half the samples being corrupted.

▶ **Theorem 9.** *Let $m \gtrsim \frac{n}{\epsilon^2}$ and $0 < \epsilon < 0.2$,*

1. *If $\eta > \eta_0 + \epsilon$ and $d = 0$ (i.e., there is no dense noise), then there exists a choice for ζ such that ℓ_1 regression does not exactly recover the original signal vector.*
2. *Even if $\zeta = 0$ (i.e., there are no sparse corruptions), there exists a choice for d such that the solution of ℓ_1 regression, $\widehat{w}$, satisfies*

$$\|\widehat{w} - w^*\|_2 \gtrsim \frac{\|d\|_1}{m}$$

Proof. Let T be the support of the largest ηm entries of (Xw^*). For the first part, let $\zeta = -(Xw^*)_T$ and observe that since $d = 0$, the loss of the 0 vector with respect to y is $\|X_{\overline{T}}w^*\|_1$ and the loss of w^* is $\|X_T w^*\|_1$. Since $m \gtrsim \frac{n}{\epsilon^2}$ we know that with probability $1 - e^{-Cn}$ for some constant C,

$$\|X_T w^*\|_1 > \left(B\left(\eta - \frac{\epsilon}{2}\right) - \frac{\epsilon}{2}\right) \cdot m$$

and

$$\|X_{\overline{T}}w^*\|_1 < \left(G\left(\eta + \frac{\epsilon}{2}\right) + \frac{\epsilon}{2}\right) \cdot m.$$

Hence,

$$\|X_T w^*\|_1 - \|X_{\overline{T}}w^*\|_1 > \left(B\left(\eta - \frac{\epsilon}{2}\right) - G\left(\eta + \frac{\epsilon}{2}\right) - \epsilon\right) \cdot m \gtrsim m\epsilon.$$

The final inequality follows from a calculation similar to the one used to show (7), by looking at the Taylor expansion of $B(\eta_0 + \frac{x}{2}) - G(\eta_0 + \frac{3x}{2}) - x$ around $x = 0$. This implies $\|X_T w^*\|_1 > \|X_{\overline{T}}w^*\|_1$ and so ℓ_1 regression cannot return w^* as the answer.

Let T' be the support of the smallest $(1 - (\eta_0 + \frac{\epsilon}{2}))m$ entries of (Xw^*). For the second part, set $d = -(Xw^*)_{T'}$. Now, more than $(1 - \eta_0)m$ entries of y are 0, and so ℓ_1 regression will recover 0. The resulting error in 2-norm is $\|\widehat{w} - w^*\|_2 = \|w^*\|_2$. Since $d = -(Xw^*)_{T'}$, $\|d\|_1$ is the ℓ_1 norm of Xw^* over the smallest $1 - \eta_0 - \frac{\epsilon}{2}$ fraction of the indices. By arguments similar to earlier

$$\|d\|_1 = \|(Xw^*)_{T'}\|_1 > m\left(G\left(\eta_0 - \frac{\epsilon}{2}\right) - \epsilon\right) \cdot \|w^*\|_2.$$

It can be checked whenever $\epsilon < 0.2$, $G\left(\eta_0 - \frac{\epsilon}{2}\right) - \epsilon > 0.4$. Hence,

$$\|\widehat{w} - w^*\|_2 = \|w^*\|_2 \geq \frac{1}{G\left(\eta_0 - \frac{\epsilon}{2}\right) - \epsilon}\left(\frac{\|d\|_1}{m}\right) \gtrsim \frac{\|d\|_1}{m}.$$ ◀

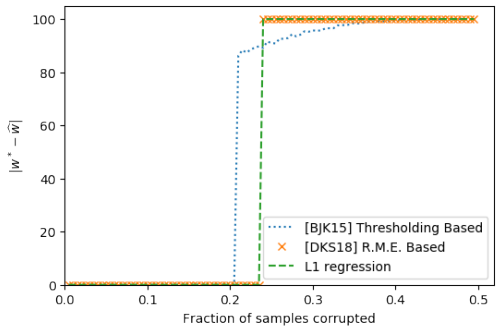


Figure 2 Empirically in the one-dimensional case, the recovery threshold for ℓ_1 regression and the robust mean estimation-based algorithm of [5] match at η_0.

7 Empirical comparisons to prior work

We compare the tolerance of ℓ_1 regression to algorithms from two recent papers [5] and [2]. We study the fraction of corruptions these algorithms can tolerate in the limit of a large number of samples. Our experiment is the following - we study the one-dimensional case where $w^* = 100$ and the adversarial noise is selected by setting the largest η fraction of observed y's to 0. We run the three algorithms on a dataset of 1000 samples for η ranging from 0 to 0.5 and consider the point when the algorithm stops providing exact recovery. In Figure 2 we plot the the error of the recovered $\widehat{w}$ from w^* against the fraction of corruptions.

While the fraction of corruptions tolerated by the algorithm from [2] for our example is more than what they prove in general (which is $\frac{1}{65}$), the fraction of corruptions it can tolerate is still less than that of ℓ_1 regression on this example. For ℓ_1 regression we observe what we have already proven earlier, that this example achieves our upper bound - i.e., it tolerates no more than an $\eta_0 \approx 0.239$ fraction of corruptions.

Curiously, the robust mean estimation based algorithm by [5] on this example tolerates exactly the same fraction of corruptions as ℓ_1 regression.

References

1 Kush Bhatia, Prateek Jain, Parameswaran Kamalaruban, and Purushottam Kar. Consistent Robust Regression. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2110–2119. Curran Associates, Inc., 2017. URL: `http://papers.nips.cc/paper/6806-consistent-robust-regression.pdf`.

2 Kush Bhatia, Prateek Jain, and Purushottam Kar. Robust regression via hard thresholding. In *Advances in Neural Information Processing Systems*, pages 721–729, 2015.

3 P. Bloomfield and W. Steiger. Least Absolute Deviations Curve-Fitting. *SIAM Journal on Scientific and Statistical Computing*, 1(2):290–301, 1980. `doi:10.1137/0901019`.

4 E. J. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. Pure Appl. Math.*, 59(8):1208–1223, 2006.

5 I. Diakonikolas, W. Kong, and A. Stewart. Efficient Algorithms and Lower Bounds for Robust Linear Regression. *ArXiv e-prints*, May 2018. `arXiv:1806.00040`.

6 Ilias Diakonikolas, Gautam Kamath, Daniel M. Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A Robust Meta-Algorithm for Stochastic Optimization. *CoRR*, abs/1803.02815, 2018. `arXiv:1803.02815`.

7 Cynthia Dwork, Frank McSherry, and Kunal Talwar. The price of privacy and the limits of LP decoding. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 85–94. ACM, 2007.

8 Rina Foygel and Lester Mackey. Corrupted sensing: Novel guarantees for separating structured signals. *IEEE Transactions on Information Theory*, 60(2):1223–1247, 2014.

9 Peter J Huber. Robust statistics. In *International Encyclopedia of Statistical Science*, pages 1248–1251. Springer, 2011.

10 Adam R. Klivans, Pravesh K. Kothari, and Raghu Meka. Efficient Algorithms for Outlier-Robust Regression. In *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018.*, pages 1420–1430, 2018. URL: `http://proceedings.mlr.press/v75/klivans18a.html`.

11 Jason N Laska, Mark A Davenport, and Richard G Baraniuk. Exact signal recovery from sparsely corrupted measurements through the pursuit of justice. In *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pages 1556–1560. IEEE, 2009.

12 Xiaodong Li. Compressed sensing and matrix completion with constant proportion of corruptions. *Constructive Approximation*, 37(1):73–99, 2013.

13 Liu Liu, Yanyao Shen, Tianyang Li, and Constantine Caramanis. High Dimensional Robust Sparse Regression. *arXiv preprint arXiv:1805.11643*, 2018.

14 Nasser M Nasrabadi, Trac D Tran, and Nam Nguyen. Robust lasso with missing and grossly corrupted observations. In *Advances in Neural Information Processing Systems*, pages 1881–1889, 2011.

15 Nam H Nguyen and Trac D Tran. Exact Recoverability From Dense Corrupted Observations via L1-Minimization. *IEEE transactions on information theory*, 59(4):2017–2035, 2013.

16 Hansheng Wang, Guodong Li, and Guohua Jiang. Robust Regression Shrinkage and Consistent Variable Selection through the LAD-Lasso. *Journal of Business & Economic Statistics*, 25(3):347–355, 2007. URL: `http://www.jstor.org/stable/27638939`.

17 Meng Wang, Weiyu Xu, and Ao Tang. The Limits of Error Correction with lp Decoding. *CoRR*, abs/1006.0277, 2010.

18 Chun Yu and Weixin Yao. Robust linear regression: A review and comparison. *Communications in Statistics-Simulation and Computation*, 46(8):6261–6282, 2017.

Appendix

A Facts about N(0,1)

In this section, let Φ and $\widehat{\Phi}$ denote the CDF of $N(0,1)$ and the CDF of the uniform distribution over a set of samples drawn from $N(0,1)$ respectively – the set will be clear from context. $B(\gamma)$ and $G(\gamma)$ refer to the ℓ_1 norm of the largest (in absolute value) γ fraction of the entries and the smallest $1-\gamma$ fraction of the entries with respect to the Gaussian distribution, and $\widehat{G}(\gamma)$ and $\widehat{B}(\gamma)$ are defined similarly but for the uniform distribution over samples from $N(0,1)$.

▶ **Fact 10.** *Let $S = \{z_1, \ldots, z_m\}$ be i.i.d. samples from $N(0,1)$. Then for any $\tau, \gamma \in [0,1]$ the following holds with probability $1 - 4e^{-2m\tau^2}$.*

$$\Phi^{-1}(\gamma - \tau) < \widehat{\Phi}^{-1}(\gamma) < \Phi^{-1}(\gamma + \tau).$$

Proof. The Dvoretzky-Kiefer-Wolfowitz inequality states

$$\Pr\Big(\sup_{x\in\mathbb{R}}\big(|\widehat{\Phi}(x) - \Phi(x)|\big) > \varepsilon\Big) \leq 2e^{-2m\varepsilon^2} \qquad \text{for every } \varepsilon \geq \sqrt{\tfrac{1}{2m}\ln 2}. \tag{8}$$

If $t = \Phi^{-1}(\eta)$, Equation 8 then tells us that for any ϵ independent of m (i.e., constant ϵ),

$$\Pr\left(|\widehat{\Phi}(t) - \eta| > \varepsilon\right) \leq 2e^{-2m\varepsilon^2}$$

i.e.,

$$\Pr\left(\widehat{\Phi}(t) \leq \eta + \epsilon\right) \leq 2e^{-2m\varepsilon^2}.$$

Setting $\eta = \gamma - \tau$ and $\epsilon = \tau$ we see

$$\Pr\left(\widehat{\Phi}(t) \leq \gamma\right) \leq 2e^{-2m\tau^2}.$$

Monotonicity of $\widehat{\Phi}$ then proves the first inequality. The second inequality follows similarly. ◀

▶ **Fact 11.** *Let $S = \{z_1, \ldots, z_m\}$ be i.i.d. samples from $N(0,1)$ and let $\eta < \eta_0$. Then with probability $1 - O\left(e^{\frac{m\epsilon^2}{2}}\right)$,*

$$\widehat{G}(\eta) > G(\eta - \epsilon) - \epsilon$$

and

$$\widehat{B}(\eta) < B(\eta + \epsilon) + \epsilon.$$

Proof. Consider the random variable Y where Y is $N(0,1)$ conditional from being drawn from $[-t, t]$ (i.e., Y has the PDF of a truncated Gaussian distribution). If $\gamma < \frac{1}{2}$, for $t = \Phi^{-1}(1 - \frac{\gamma}{2})$

$$E[|Y|] = \frac{1}{1-\gamma}\int_{-t}^{t} |x| e^{-x^2/2} dx = \frac{1}{1-\gamma} G(\gamma).$$

Observe that one can sample from Y by sampling from Z which is distributed as $N(0,1)$ and discarding samples outside $[-t, t]$. Since the PDF is scaled, we have to scale the empirical distribution as well

$$\widehat{E}[|Y|] = \frac{1}{m(1-\gamma)} \sum_{i=1}^{m} |z_i| \cdot 1_{[-t,t]}(z_i)$$

Let γ be such that $E|Y| \leq \frac{1}{2(1-\gamma)}$, then $|Y|$ has subgaussian tails with some constant parameter. To see this, observe that

$$\begin{aligned} E\left[e^{\lambda(|Y| - E[|Y|])}\right] &= \frac{2}{1-\gamma}\int_{0}^{t} e^{-x^2/2 + \lambda x - \lambda E[|Y|]} dx \\ &\lesssim e^{\frac{\lambda^2}{2} - (2\cdot(1-\gamma))^{-1}\lambda} \\ &\lesssim e^{O(\lambda^2/2 - \lambda)} \end{aligned}$$

This implies concentration for the expectation

$$\Pr\left(\left|\widehat{E}[|Y|] - E[|Y|]\right| > \epsilon\right) < O\left(e^{-\frac{m\epsilon^2}{2}}\right).$$

Multiplying both sides inside the probability by $(1-\gamma)$ and noting that since $\gamma < \frac{1}{2}$ this is bounded by $\frac{1}{2}$ we see

$$\Pr\left(\left|\frac{1}{m}\sum_{i=1}^{m} |z_i| \cdot 1_{[-t,t]}(z_i) - G(\gamma)\right| > \frac{\epsilon}{2}\right) < O\left(e^{-\frac{m\epsilon^2}{2}}\right)$$

We now set $\gamma = \eta - \epsilon$. Since $\eta < \eta_0 \approx 0.239$ and $\epsilon > 0$, $\gamma < \frac{1}{2}$. Fact 10 now implies that with probability $1 - 2e^{-2m\epsilon^2}$, at most an $1 - \eta$ fraction of the samples lie in $[-t, t]$. These have to be smaller in absolute value than the remaining samples. Since $\widehat{G}(\eta)$ is defined to be the ℓ_1 norm of the $1 - \eta$ fraction of points smallest in absolute value, we see

$$\widehat{G}(\eta) > \frac{1}{m}\sum_{i=1}^{m} |z_i| \cdot 1_{[-t,t]}(z_i).$$

This implies

$$\Pr\left(\widehat{G}(\eta) > G(\eta - \epsilon) - \epsilon\right) < O\left(e^{-2m\epsilon^2} + e^{-\frac{m\epsilon^2}{2}}\right).$$

The other direction is done similarly, however in this case Y is the random variable gotten by conditioning samples from $N(0, 1)$ to be outside $[-t, t]$. ◀

B Shelling argument

▶ **Lemma 6** (Shelling Argument). *Let $A \in \mathbb{R}^{m\times n}$ satisfy*

$$L\|v\|_2 \le \|Av\|_1 \le U\|v\|_2$$

for all $(1 + \alpha^2)k$-sparse vectors v. If $S \subset [m]$ is fixed and of cardinality k, then A satisfies

$$\frac{L}{1+\alpha}\left(\alpha - \frac{U}{L}\right)\|v\|_2 - \frac{2U\Delta}{\alpha\sqrt{k}} \le \|Av\|_1 \le U\left(1 + \frac{1}{\alpha}\right)\|v\|_2 + \frac{U\Delta}{\alpha\sqrt{k}}$$

for all

$$v \in V_S = \{v \in \mathbb{R}^n \mid \Delta + \|v_S\|_1 \ge \|v_{\overline{S}}\|_1\}.$$

Proof. The goal is to transfer bounds from the eigenvalues of A restricted over the sparse vectors, to the eigenvalues of A restricted over V_S. To this end we will select an element of V_S and express it as a sum of sparse vectors. Applications of standard inequalities will then let us transfer bounds.

For any $v \in V_S$ partition $[n]$ into $S, T_1, \ldots, T_{\frac{n-k}{\alpha^2 k}}$ where T_i is the set of indices corresponding to the i^{th} largest $\alpha^2 k$-sized set of elements from $v_{\overline{S}}$.

We will now prove the upper and lower bounds on the eigenvalues for vectors restricted to the set V_S. The triangle inequality implies

$$\|Av_{S\cup T_1}\|_1 - \sum_{i>1}\|Av_{T_i}\|_1 \le \|Av\|_1 \le \|Av_{S\cup T_1}\|_1 + \sum_{i>1}\|Av_{T_i}\|_1$$

Since $v_{S\cup T_1}$ and v_{T_i} are all at most $(1 + \alpha^2)k$-sparse,

$$L\|v_{S\cup T_1}\|_2 - \sum_{i>1}\|Av_{T_i}\|_1 \le \|Av\|_1 \le U\|v_{S\cup T_1}\|_2 + \sum_{i>1}\|Av_{T_i}\|_1$$

We now prove an upper bound on the quantity $\sum_{i>1}\|Av_{T_i}\|_1$. This will give us both the upper and lower bounds we need. To this end, observe that all coordinates of $v_{T_{i-1}}$ are greater than or equal to all coordinates of v_{T_i}. This implies

$$\|v_{T_i}\|_\infty \le \frac{\|v_{T_{i-1}}\|_1}{\alpha^2 k}$$

which, in turn, implies

$$\|v_{T_i}\|_2 \leq \frac{1}{\alpha\sqrt{k}}\|v_{T_{i-1}}\|_1.$$

Using the bounds on the restricted sparse eigenvalues from the statement, we get

$$\begin{aligned}\sum_{i>1}\|Av_{T_i}\|_1 &\leq U\cdot\sum_{i>1}\|v_{T_i}\|_2\\ &\leq \frac{U}{\alpha\sqrt{k}}\|v_{\overline{S}}\|_1\\ &\leq \frac{U}{\alpha\sqrt{k}}\left(\|v_S\|_1+\Delta\right)\\ &\leq \frac{U}{\alpha}\cdot\|v_S\|_2+\frac{U}{\alpha\sqrt{k}}\cdot\Delta\\ &\leq \frac{U}{\alpha}\cdot\|v_{S\cup T_1}\|_2+\frac{U}{\alpha\sqrt{k}}\cdot\Delta\end{aligned}$$

Using the inequality above in addition to the bounds on $\|Av\|_1$, we get after some rearrangement

$$\left(L-\frac{U}{\alpha}\right)\|v_{S\cup T_1}\|_2-\frac{U}{\alpha\sqrt{k}}\cdot\Delta\leq\|Av\|_1\leq U\left(1+\frac{1}{\alpha}\right)\|v_{S\cup T_1}\|_2+\frac{U}{\alpha\sqrt{k}}\cdot\Delta$$

The bounds above are in terms of $\|v_{S\cup T_1}\|_2$, however we need bounds in terms of $\|v\|_2$. For the upper bound, it is sufficient to note that $\|v_{S\cup T_1}\|_2<\|v\|_2$. For the lower bound, we need the inequalities below.

The definition of T_i and applications of the Cauchy-Schwartz inequality gives us

$$\|v_{\overline{S\cup T_1}}\|_2\leq\sum_{i\geq 2}\|v_{T_i}\|_2\leq\frac{1}{\alpha\sqrt{k}}\sum_{i\geq 1}\|v_{T_{i-1}}\|_1\leq\frac{\|v_{\overline{S}}\|_1}{\alpha\sqrt{k}}\leq\frac{\|v_S\|_1+\Delta}{\alpha\sqrt{k}}\leq\frac{\|v_S\|_2}{\alpha}+\frac{\Delta}{\alpha\sqrt{k}}.$$

This, in turn, results in an upper bound on $\|v\|_2$ in terms of $\|v_{S\cup T_1}\|_2$,

$$\begin{aligned}\|v\|_2&\leq\|v_{S\cup T_1}\|_2+\|v_{\overline{S\cup T_1}}\|_2\\ &\leq\|v_{S\cup T_1}\|_2+\sum_{i\geq 2}\|v_{T_i}\|_2\\ &\leq\|v_{S\cup T_1}\|_2+\frac{\|v_S\|_2}{\alpha}+\frac{\Delta}{\alpha\sqrt{k}}\\ &\leq\left(1+\frac{1}{\alpha}\right)\|v_{S\cup T_1}\|_2+\frac{\Delta}{\alpha\sqrt{k}}\\ &\implies\|v_{S\cup T_1}\|_2\geq\frac{\alpha}{1+\alpha}\left(\|v\|_2-\frac{\Delta}{\alpha\sqrt{k}}\right)\end{aligned}$$

and so

$$\frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\left(\|v\|_2-\frac{\Delta}{\alpha\sqrt{k}}\right)-\frac{U\Delta}{\alpha\sqrt{k}}\leq\|Av\|_1\leq U\left(1+\frac{1}{\alpha}\right)\|v_{S\cup T_1}\|_2+\frac{U\Delta}{\alpha\sqrt{k}}$$

At this point, we have the upper bound, to complete the proof of the lower bound, observe

that standard manipulations give us

$$\begin{aligned}\frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\left(\|v\|_2-\frac{\Delta}{\alpha\sqrt{k}}\right) &= \frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\|v\|_2-\frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\frac{\Delta}{\alpha\sqrt{k}}\\ &= \frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\|v\|_2-\frac{\alpha L-U}{1+\alpha}\frac{\Delta}{\alpha\sqrt{k}}\\ &\geq \frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\|v\|_2-\frac{U\Delta}{\alpha\sqrt{k}}\end{aligned}$$

This gives us the Lemma,

$$\frac{L}{1+\alpha}\left(\alpha-\frac{U}{L}\right)\|v\|_2-\frac{2U\Delta}{\alpha\sqrt{k}}\leq\|Av\|_1\leq U\left(1+\frac{1}{\alpha}\right)\|v_{S\cup T_1}\|_2+\frac{U\Delta}{\alpha\sqrt{k}}.$$ ◀

Approximating Maximin Share Allocations

Jugal Garg
University of Illinois at Urbana-Champaign
jugal@illionis.edu

Peter McGlaughlin
University of Illinois at Urbana-Champaign
mcglghl2@illionis.edu

Setareh Taki
University of Illinois at Urbana-Champaign
staki2@illionis.edu

Abstract

We study the problem of fair allocation of M indivisible items among N agents using the popular notion of maximin share as our measure of fairness. The maximin share of an agent is the largest value she can guarantee herself if she is allowed to choose a partition of the items into N bundles (one for each agent), on the condition that she receives her least preferred bundle. A maximin share allocation provides each agent a bundle worth at least their maximin share. While it is known that such an allocation need not exist [9, 7], a series of work [9, 8, 1, 2] provided 2/3 approximation algorithms in which each agent receives a bundle worth at least 2/3 times their maximin share. Recently, [6] improved the approximation guarantee to 3/4. Prior works utilize intricate algorithms, with an exception of [2] which is a simple greedy solution but relies on sophisticated analysis techniques. In this paper, we propose an alternative 2/3 maximin share approximation which offers both a simple algorithm and straightforward analysis. In contrast to other algorithms, our approach allows for a simple and intuitive understanding of why it works.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Fair division, Maximin share, Approximation algorithm

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.20

Funding Work on this paper partly supported by NSF CRII Award 1755619.

1 Introduction

We study the problem of allocating M indivisible items among N agents with additive valuations in a fair way, using the popular notion of maximin share [5] as our measure for fairness. There is an extensive literature for fair allocation of divisible items, starting with the cake cutting problem [10]. Standard notions of fairness include: envy-freeness where every agent prefers their allocation over any other agents' allocation, and proportionality where every agent receives at least a $1/N$ share of all the items.

In the case of indivisible items, a simple counter example shows that no algorithm can provide either envy-freeness or proportionality. Consider allocating a single item between $N > 1$ agents. Clearly, $N - 1$ agents envy the one lucky agent that received the item and there is no way to ensure all agents receive a bundle of items with value at least $1/N$. This motivates the need for an alternate concept of fairness. Recently, Budish [5] introduced an intriguing option, a maximin share. The idea is a natural generalization of the well known cut and choose protocol in the cake cutting problem. Suppose we allow agent i to choose a partition of the items into N bundles (one for each agent), with the caveat that the other

$N-1$ agents get to choose a bundle before her. In the worst case, she receives her least preferred bundle. Clearly, in this situation i will choose a partition that maximizes the value of her least preferred bundle. We call the value of this bundle i's maximin share (MMS). Since all other agents may have the same valuations as her, i's MMS is the most she can guarantee for herself in this scenario. In this paper, we focus on the case of additive valuations. Let us note that, computing any agent's MMS is NP-hard, but a PTAS exists [11].

A maximin share gives an intuitive local measure of fairness of an allocation, that is a specific objective for each agent. This raises the natural question: Is there an allocation where each agent receives a bundle worth at least their MMS? An allocation satisfying this property is said to be maximin share allocation (MMS allocation), and if it exists, an MMS allocation provides strong fairness guarantees to each individual agent. Bouveret and Lemaître [3] show that an MMS allocation always exist in some special settings, e.g., when there are only two agents or if agents' valuations for items are 0 or 1, but leave the general case as an open problem.

Procaccia and Wang [9] obtain the surprising result that MMS allocations might not exist, by means of a clever counter example. However, they show that a 2/3 MMS allocation always exists, that is an allocation where each agent receives a bundle worth at least 2/3 of their maximin share, and they provide a polynomial time algorithm to find a 2/3 MMS allocation when the number of agents N is constant. In the special case where $N \leq 4$, their algorithm finds a 3/4 MMS allocation. Amanatidis et al. [1] improve this result by addressing the requirement for a constant number of agents, obtaining a PTAS which finds a $(2/3 - \epsilon)$ MMS allocation for an arbitrary number of agents; see [8] for an alternate proof. [1] also shows that a 7/8 MMS allocation always exists when there are three agents.

Barman and Murthy [2] take an alternate approach from [9, 1], utilizing key insights from [3] to obtain a greedy algorithm to find a 2/3 MMS allocation. While the algorithm itself is fairly simple, the proof is not.

The recent results of Ghodsi et al. [6] breaks new ground, establishing existence of 3/4 MMS allocation, and, building on the work of [9, 1], provides a PTAS to find a $(3/4 - \epsilon)$ MMS allocation. They also show that when $N = 4$, an 4/5 MMS allocation exists and proposed a algorithm to find it.

Our Contribution. We present an algorithm to find a 2/3 MMS allocation for agents with additive valuations. Our approach combines the insights of [3, 2] with the concepts developed in [6] to obtain an algorithm that is both simple to implement and analyze. Like [6], our algorithm consists of two phases: *matching* and *bag filling*. However, unlike [6], our phases are much simpler, and we do not need to compute agents' MMS values. Bag filling is a simple, greedy method to allocate 'low' valued items. We add one item at a time to a bag. If the value of the bag is worth at least 2/3 of any agent's MMS, then we assign the bag to that agent, picking an arbitrary agent when there are more than one satisfying this condition. It is easily shown, in Section 2.2, that bag filling provides a 2/3 MMS allocation as long as no agent values any item more than 1/3. Thus, the real difficultly lies in distributing 'high' value items, i.e., items worth more than 1/3 to some agent. Drawing on the insights of [3, 2], we show a combination of maximum matching and greedy assignment suffices for this purpose. This gives our algorithm the basic structure: repeated maximum matching and greedy assignment to remove high valued items, followed by bag filling to allocate low valued items. Our approach allows for far simpler and more intuitive analysis than [1, 2, 6].

2 Preliminaries

We consider the fair allocation of $M = \{1, \ldots, m\}$ indivisible items among $N = \{1, \ldots, n\}$ agents with additive valuations. That is, v_{ij} is agent i's value for item j, and i's valuation of any bundle of items $S \subseteq M$ is: $v_i(S) = \sum_{j \in S} v_{ij}$. For simplicity, we also use $v_i(j)$ instead of $v_i(\{j\})$. Denote the set of valuation functions, $v_i : 2^M \to \mathbb{R}^+$, as: $V = \{v_1, \ldots, v_n\}$. An allocation $A = \{A_1, \ldots, A_n\}$ is a partition of the items into n bundles (one for each agent). We define fair allocations in terms of maximin shares. Agent i's maximin share (μ_i) is the maximum value she can guarantee herself if she is allowed to choose the allocation A, on the condition that she receives her least preferred bundle. Formally, let A be an allocation and $\mathcal{A} = \left\{A = \{A_1, \ldots, A_n\} : \ A_i \cap A_j = \emptyset, \forall i, j; \ \cup_k A_k = M\right\}$ be the set of all feasible allocations. Agent i's maximin share is:

$$\mu_i = \max_{A \in \mathcal{A}} \min_{A_k \in A} v_i(A_k). \tag{1}$$

We say an allocation A is MMS if each agent i receives a bundle A_i worth at least her maximin share: $v_i(A_i) \geq \mu_i$. An allocation is α approximate MMS (or simplify α-MMS) if each agent i receives a bundle A_i worth at least: $v_i(A_i) \geq \alpha\mu_i$, for some $\alpha \in (0, 1)$.

2.1 Properties of Maximin Share

Our approximation algorithm exploits a few key properties of maximin shares. We note that these are standard results which appear in [1, 6]. We include proofs for sake of completeness.

▶ **Proposition 1** (Scale Invariance). *Let $A = \{A_1, \ldots, A_n\}$ be an α-MMS allocation for the problem instance $I = (N, M, V)$ with additive valuations. For any agent $i \in N$ and any $c \in \mathbb{R}^+$, if we create an alternate instance $I' = (N, M, V')$ where i's valuations are scaled by c, i.e., $v'_{ij} := cv_{ij}, \forall j \in M$, then A is still an α-MMS allocation for (N, M, V').*

Proof. Let μ_i and μ'_i be agent i's MMS in instance I and I' respectively. For any bundle $S \subseteq M$, we have $v'_i(S) = cv_i(S)$. Therefore, $\mu'_i = c\mu_i$. Let $A = \{A_1, \ldots, A_n\}$ be the allocation i selects to create her μ_i. Then, $v'_i(A_k) = cv_i(A_k) \geq c\alpha\mu_i = \alpha\mu'_i, \forall k$. ◀

▶ **Proposition 2** (Normalized Valuation). *For problem instance $I = (N, M, V)$, if agent i's valuation function satisfies:*

$$v_i(M) = \sum_{j \in M} v_{ij} = |N|, \tag{2}$$

then $\mu_i \leq 1$.

Proof. For contradiction, suppose $v_i(M) = |N|$ but $\mu_i > 1$. Let $A = \{A_1, \ldots, A_n\}$ be the allocation i selects to create her μ_i. From the definition of μ_i (1), $v_i(A_k) \geq \mu_i \ \forall A_k \in A$, so $|N| = v_i(M) = \sum_k v_i(A_k) \geq |N|\mu_i > |N|$, a contradiction. ◀

We say agent i's valuation is normalized for $I = (N, M, V)$ when (2) holds, or simply normalized when the underlying problem instance is clear. In view of Proposition 1, normalizing agents' valuations provides a convenient upper bound on μ_i's without affecting performance guarantees. In addition, this removes the problem of comparing the relative value of a bundle of items between agents whose scale of valuations differs in orders of magnitude.

2.2 What Makes Finding Approximate MMS Allocations Hard?

In this section, we build intuition for what exactly makes finding α approximate MMS allocations difficult. We begin with a definition. Let $I = (N, M, V)$ be a problem instance, and $L \subset N$ be subset of agents and $S \subset M$ be subset of items. We say

$$I' = (N', M', V') = (N \setminus L, M \setminus S, V), \tag{3}$$

is a reduced instance of I. In words, we create a reduced instance by removing some subset of agents, and some subset of items. We call the agents $N' = N \setminus L$ the remaining agents of the reduced instance I'. The following simple observation plays an important role in finding approximate MMS allocations.

▶ **Proposition 3.** *Let $I = (N, M, V)$ be a problem instance, and let μ_i be the MMS for agent $i \in N$. If we remove one agent $k \in N$ and one item $j \in M$, then the MMS of all remaining agents in the reduced instance $I' = (N \setminus \{k\}, M \setminus \{j\}, V)$, is at least as large as their MMS in I, i.e., $\mu'_i \geq \mu_i$. In words, removing one agent and one item from a problem instance does not reduce the MMS guarantees for any remaining agent in the reduced instance.*

Proof. Suppose agent $k \in N$ and item $j \in M$ are removed from the instance, and let i be any remaining agent. Consider the allocation $A = \{A_1, \ldots, A_n\}$ she makes to calculate her MMS in the original instance I, and note that $A_l \geq \mu_i$ for all $A_l \in A$ by the definition of MMS. In the reduced instance $I' = (N \setminus \{k\}, M \setminus \{j\}, V)$, agent i needs to make one less bundle but has one less item. Let $A_l \in A$ be the bundle containing the removed item j. Suppose she simply takes the items of $A_l \setminus \{j\}$ and distributes them arbitrarily to the other bundles of A to create a new allocation $\hat{A} = \{\hat{A}_1, \ldots, \hat{A}_{n-1}\}$. Clearly, $\hat{A}$ is a feasible allocation, and $\hat{A}_l \geq \mu_i$ for all $\hat{A}_l \in \hat{A}$. Therefore, $\mu'_i \geq \mu_i$. ◀

Proposition 3 shows that removing one agent and one item does not reduce MMS guarantees for remaining agents in the reduced instance. It is straightforward to generalize the above argument to show that removing k agents and k items does not decrease MMS guarantees in the reduced instance.

▶ **Corollary 4.** *Let $L \subset N$ and $S \subset M$, and μ_i be the MMS for each agent i in an instance $I = (N, M, V)$. If $|L| = |S|$, then the MMS μ'_i of any remaining agent in the reduced instance $I' = (N \setminus L, M \setminus S, V)$ is at least as large as in the original instance, i.e., $\mu'_i \geq \mu_i$.*

Combining Proposition 3 with the simple greedy allocation method *bag filling*, yields an almost trivial 1/2-MMS allocation algorithm. Suppose we seek an α-MMS allocation. The bag filling algorithm is as follows: We add one, arbitrary item at a time to a bag S until an agent k values the bag at least $\alpha\mu_k$, i.e., $v_k(S) \geq \alpha\mu_k$. If another agent k' values the bag at least $\alpha\mu'_k$, then pick one arbitrarily. We assign k the bag S, and remove agent k and the items of S from the instance. We show that bag filling provides an efficient way to allocate low value items. We note that a similar result also appears in [6].

▶ **Proposition 5.** *Assume agents' valuations are normalized as defined in* (2)*, and that no agent values any item more than $0 < \delta < 1/2$: $v_{ij} \leq \delta$ for all $j \in M$, for all $i \in N$. Then, the bag filling algorithm gives a $(1 - \delta)$ MMS allocation.*

Proof. By the definition of normalized valuations and Proposition 2, we have $v_i(M) = |N|$ and $\mu_i \leq 1$ for all $i \in N$. Therefore, it is enough to show each agent receives a bag worth at least $1 - \delta$. Clearly, the agent that receives the bag in each iteration gets at least $1 - \delta$, so the claim amounts to showing remaining agents do not lose too much value when the

bag is assigned. Let j be the last item added to the bag S. Note that before adding j, all agents valued S less than $1-\delta$, i.e., $v_i(S \setminus j) < 1-\delta$ for all $i \in N$. Now, since valuations are additive and $v_{ij} < \delta$ for all agents $i \in N$, we have $v_i(S) \leq 1$. That is, no agent values the bag S more than 1. This means after removing agent k and the items of S, all remaining agents satisfy $v_i(M \setminus S) = v_i(M) - v_i(S) \geq |N| - 1$. Since this condition is an invariant of bag filling algorithm, all agents get at least $(1-\delta)$ of their maximin share. ◀

Combining Propositions 2, 3, and 5 yields a simple greedy algorithm to compute a 1/2 MMS allocation. We start by normalizing valuations as defined in (2). By Proposition 2, this ensures $\mu_i \leq 1$ for all agents $i \in N$. If some agent, say k, has valuation $v_{kj} \geq 1/2$ for some item j, then we assign item j to agent k. If more than one agent satisfies this condition, then pick one arbitrarily. By Proposition 3, the MMS μ'_i of agents in the reduced instance $I' = (N \setminus k, M \setminus j, V)$ is at least as large as their MMS μ_i in the original instance. Next we normalize valuations for the reduced instance I', and repeat the process, greedily assigning one item at a time to any agent who values the item at least 1/2 and then normalizing valuations, until either all agents are removed or $v_{ij} < 1/2$, $\forall j \in M$, $\forall i \in N$. In the later case, Proposition 5 shows that the bag filling algorithm provides all remaining agents a bundle worth at least 1/2 of their maximin share.

A natural approach to extending the above algorithm to give a 2/3 MMS allocation requires splitting the items M into three sets based on their value: high valued items for which $v_{ij} \geq 2/3$ for some agent $i \in N$, low valued items for which $v_{ij} < 1/3$ for all agents $i \in N$, and medium valued items for which $v_{ij} \geq 1/3$ for at least one agent $i \in N$ but $v_{ij} < 2/3$ for all $i \in N$. Notice that, similar to the 1/2 MMS algorithm above, we may greedily assign high valued items to give at least 2/3 MMS to the agent receiving the item without decreasing the MMS of any remaining agent in the reduced instance. Also, if all items are low valued $v_{ij} < 1/3$ $\forall i \in N$, then the bag filling algorithm easily yields a 2/3 MMS allocation. The real challenge lies in managing the medium valued items. These items are not valuable enough individually to satisfy $2/3\mu_i$ for an agent i, yet they are too valuable, to some agent, to distribute haphazardly through bag filling. Thus, we seek a simple, efficient means to allocate medium valued items.

2.3 Results of Bouveret and Lemaître, and Barman and Murthy

Our algorithm relies on some results of [4, 2] to obtain the means to properly manage 'medium valued' items as defined at the end of the last section. We start with a definition. A problem instance $I = (N, M, V)$ is ordered if:

$$v_{i1} \geq v_{i2} \geq \cdots \geq v_{im}, \quad \forall i \in N. \tag{4}$$

In words, in an ordered instance all agents have the same order of preference over items. Roughly speaking, this maximizes the competition between agents, and, intuitively, should make it more difficult to provide an MMS allocation. Indeed, Bouveret and Lemaître [4] show ordered instances are worst case. Further, they provide a reduction from any arbitrary instance $I = (N, M, V)$ to an ordered instance $I' = (N, M, V')$, and show that if A' is an MMS allocation for I', then one can find an MMS allocation A for I in polynomial time. Barman and Murthy [2] generalize these results for α approximate MMS allocations.

▶ **Proposition 6** (Section 2.1 of Barman and Murthy [2]). *Given any instance $I = (N, M, V)$, one can find an ordered instance $I' = (N, M, V')$ in polynomial time.*

Algorithm 1: Converting to an Ordered Instance.

Input : Original Instance (N, M, V)
Output : V': Valuations for Ordered Instance

```
1 for j = 1 to m do
2     for i = 1 to n do
3         j* = agent i's jth most valuable item ;
4         v'_ij ← v_i(j*) ;
```

Algorithm 2: α-MMS Allocation for Unordered Instance.

Input : Allocation $A' = (A'_1, \ldots, A'_n)$ for Ordered Normalized Instance $I' = (N, M, V')$ such that $v'_i(A'_i) \geq \alpha$ for all $i \in N$.
Output : Allocation $A = (A_1, \ldots, A_n)$ for Original Normalized Instance $I = (N, M, V)$ such that $v_i(A_i) \geq \alpha$ for all $i \in N$.

```
1 A = (∅, ..., ∅) and R ← M ;
2 for j = 1 to m do
3     a ← i : j ∈ A'_i (pick the agent assigned item j in A') ;
4     g ← arg max_{k∈R} v_ak;
5     A_i ← A_i ∪ {g} and R ← M \ {g};
```

Algorithm 1 gives explicit details for the process of converting any instance I into an ordered instance I'. We call I' constructed this way the ordered instance of I.

▶ **Theorem 7** (Theorem 2 and Corollary 1 of Barman and Murthy [2]). *For any instance I, let I' be its ordered instance. If A' is an α approximate MMS allocation for I', then using A' we can find allocation A which is an α approximate MMS allocation for I in polynomial time.*

Algorithm 2 shows how to obtain an α approximate MMS allocation A for the original instance I given an α approximate MMS allocation for the ordered instance I'. For the sake of completeness, we provide a brief proof of Theorem 7.

Proof. (Theorem 7) Clearly, both Algorithms 1 and 2 run in polynomial time. Notice that Algorithm 2 allocates each item $j \in M$ to at most one agent $i \in N$ and that one item is allocated in each iteration. Let k_j be the item allocated in the jth iteration of Algorithm 2, lines 2 through 5. Consider the agent i assigned $j \in A'_i$, meaning that $k_j \in A_i$. At the beginning of the jth iteration, exactly $j-1$ items have been allocated. Therefore, k_j is among the top j most valuable items for agent i. From the construction of the ordered instance I', it follows that for all $j \in A'_i$, $v_i(k_j) \geq v'_i(j)$. Therefore, $v_i(A_i) = \sum_{j \in A'_i} v_i(k_j) \geq \sum_{j \in A'_i} v'_i(j) = v'_i(A'_i) \geq \alpha$. ◀

Proposition 6 and Theorem 7 show that it suffices to consider ordered instances. A total ordering over the set of items M provides precious information to us as algorithm designers since we know precisely which items are best (favored by all agents). In other words, the ordering over M essentially means all items fall into three categories: low, medium, and high valued, corresponding to low, medium, and high in the ordering respectively. In Section 3, we show that a total ordering over the items M allows for a simple generalization of the 1/2 MMS allocation algorithm described in Section 2.2 to give 2/3 MMS allocations.

3 A 2/3 MMS Approximation

In this section we present an algorithm to find 2/3 approximate MMS allocations. Our method involves a preprocessing step with Proposition 6 to ensure the instance is ordered. We show how to obtain a 2/3 MMS allocation for the ordered instance, then use a post processing step with Theorem 7 to obtain a 2/3 MMS allocation for the original instance. From this point on, we assume the instance is ordered as defined in (4).

The algorithm builds one bundle of items at a time, assigns it to some agent i who values it at least $2/3\mu_i$, and then removes that agent and the bundle from the instance. The basic structure of the algorithm closely resembles the simple 1/2 MMS algorithm discussed in Section 2.2. In fact, the same simple strategies guide the algorithm's design.

Assuming valuations are normalized as defined in (2), our algorithm handles allocation of items based on their value: low, medium or high. For this we use the clustering approach of [6], and define the following sets of items:

$$\begin{aligned} S_H &= \{j \in M : \ \exists i \in N \text{ s.t. } v_{ij} \geq 2/3\} \\ S_M &= \{j \in M : \ \exists i \in N \text{ s.t. } 1/3 \leq v_{ij}, \ v_{ij} < 2/3, \ \forall i \in N\} \\ S_L &= \{j \in M : \ v_{ij} < 1/3, \ \forall i \in N\}, \end{aligned} \tag{5}$$

which correspond to high, medium, and low valued items respectively. Second, for any bundle $S \subseteq M$, we define the set $N(S)$ as the agent's with value at least 2/3 for S:

$$N(S) = \{i : \ i \in N, \ v_i(S) \geq 2/3\}. \tag{6}$$

By using the preprocessing step of Proposition 6, we ensure a total ordering on the items (4). Thus, for any agent i if $v_{ik} > 1/3$ for some item k, then $v_{ij} > 1/3$ for all $j \leq k$. Similarly, if $v_{ik} < 2/3$, then $v_{ij} < 2/3$ for all $j \geq k$.

3.1 2/3 MMS Algorithm

At a high level, our algorithm mirrors the simple 1/2 MMS algorithm, consisting of two phases: matching and bag filling. Like the 1/2 MMS algorithm, we allocate high value items S_H through a maximum matching, and assign all low value items S_L through bag filling. For medium valued items S_M, the total ordering on the items simplifies allocation decisions based on $|S_M|$. If $|S_M|$ is sufficiently large, we greedily assign a bundle containing the two 'least valuable' items of $|S_M|$ to any agent that values it at least 2/3, using a generalization of Proposition 3. Otherwise, we use a modified version of the bag filling algorithm. We make the treatment of medium valued items more precise shortly, but note that, the total ordering of items allows for small adjustments to the matching and bag filling stages of the 1/2 MMS algorithm to improve the approximation guarantees to 2/3 MMS. Further, our approach makes the analysis of each stage nearly as simple as in the 1/2 MMS algorithm. We now explain the algorithm in more detail, see Algorithm 3 for a formal description.

Matching Procedure. The initial phase of the algorithm allocates high value items S_H through a maximum matching we call the Matching Procedure. First, we normalize valuations which ensures $\mu_i \leq 1$ for all agents by Proposition 2. Next, we form a bipartite graph with agents of on the left hand side and items of S_H on the right. We create an edge between agent i and item j, if $v_{ij} \geq 2/3$. In words, the graph's edges connect agents with items they value at least 2/3. Next, we solve a maximum matching T, and assign i bundle $A_i = j$, if $(i, j) \in T$. All matched items and agents are removed from the instance and we normalize valuations for the remaining agents. This process repeats until $|S_H| = 0$, i.e., there are no more high valued items.

Greedy Assignment from S_M. After completing the first phase, all high value items are allocated. Next, we determine how to distribute medium and low value items among the remaining agents. Our preprocessing step with Proposition 6 ensures the instance is ordered (4), meaning there is a least preferred item in any set of items (5). More precisely, $j^* = \arg\max_{j \in S_M} j$ is the least preferred item of S_M (medium value items). When $|S_M| > |N|$, each agent must create at least one bundle containing two or more items of S_M when calculating their μ_i, by pigeon hole principle. Similar to the matching stage of the 1/2 MMS algorithm, we greedily assign the two least preferred items of S_M, $S = \{j^* - 1, j^*\}$, to an arbitrary agent i with valuation $v_i(S) \geq 2/3$. This ensures the i receiving S gets at least $2/3\mu_i$, and the MMS of all remaining agents k in the reduced instance $I' = (N \setminus i, M \setminus S, V)$ satisfy: $\mu'_k \geq \mu_k$. Agent's valuations are then normalized, and process repeats.

Modified Bag Filling. After allocating the bulk of medium value items $|S_M| \leq |N|$, we create bundles for the remaining agents through a slightly modified version of bag filling. Here, we simply initialize the bag S using one, arbitrary item from S_M, and then fill the bag with items from S_L (low valued items) until some agent i values S at least $v_i(S) \geq 2/3$. Once $|S_M| = 0$, we use the standard bag filling algorithm.

Recall that the challenge of improving the approximation guarantees of the 1/2 MMS algorithm requires proper management of the medium valued items S_M. Our approach, using Proposition 6 and Theorem 7 to ensure the instance is ordered, enables a simple and natural extension of the straight-forward 1/2 MMS algorithm to provide improved 2/3 MMS guarantees.

The algorithm consists of two phases, matching and bag filling. The phases use different allocation procedures based on $|S_H|$ and $|S_M|$ respectively. We consider these procedures separately, starting with the matching procedure.

▶ **Lemma 8.** *Let $I = (N, M, V)$ be a problem instance where agents' valuations are normalized as defined in (2), and let μ_i be agent i's MMS. Suppose $|S_H| > 0$, as defined in (5), and that the Matching Procedure is used to create a maximum matching T. Let L be the agents of T and S be the items of T. Then,*

- **(i)** *$|L| = |S| > 0$.*
- **(ii)** *All removed agents i receive at least $2/3\mu_i$.*
- **(iii)** *Let μ'_i be the MMS of remaining agent i in the reduced instance $I' = (N \setminus L, M \setminus S, V)$. Then, $\mu'_i \geq \mu_i$.*

Proof. Recall the Matching Procedure creates a bipartite graph $G = (V, E)$ where the vertices V consist of agents on the left side and items on the right. An edge $e \in E$ is created between agent i and item j if $v_{ij} \geq 2/3$. Finally, a maximum matching T is determined. By definition of S_H and the fact $|S_H| > 0$, the set of edges E of G is non-empty. Since T is a maximum matching, part i) is obvious. Next, recall that Proposition 2 shows that $\mu_i \leq 1$ for all $i \in N$ since valuations are normalized. Part ii) then follows by the construction of G. Finally, since $|L| = |S|$, Corollary 4 guarantees $\mu'_i \geq \mu_i$ for all remaining agents i. ◀

We now consider the second procedure of the algorithm's matching phase.

▶ **Lemma 9.** *Let $I = (N, M, V)$ be an ordered problem instance with normalized valuations, and let μ_i be agent i's MMS. Suppose that $|S_H| = 0$ and $|S_M| > |N|$. Define $j^* = \arg\max_{j \in S_M} j$, and let $S = \{j^*, j^* - 1\}$ be the two least preferred items of S_M. Suppose bundle S is assigned an arbitrary agent k satisfying $v_k(S) \geq 2/3$. Then,*

- **(i)** *$v_k(S) \geq 2/3\mu_k$.*
- **(ii)** *Let μ'_i be the MMS of any remaining agent i in the reduced instance $I' = (N \setminus k, M \setminus S, V)$. Then, $\mu'_i \geq \mu_i$.*

Algorithm 3: 2/3-MMS Allocation.

Input : Ordered Instance $\langle N, M, V \rangle$
Output : 2/3 Approximate Maximin Share Allocation

while $|N| > 0$ **do**
  Normalize Valuations ;
  if $|S_H| > 0$ **then**
    Matching Procedure ;
  else if $|S_M| > |N|$ **then**
    $j^* \leftarrow \max_{j \in S_M} j$; // lowest value item of S_M
    $N(j^*) \leftarrow \{i : \ i \in N, \ v_i(j^*, j^* - 1) \geq 2/3\}$;
    $i \in N(j^*)$; $A_i \leftarrow \{j^*, j^* - 1\}$; // assign i the bundle $\{j^*, j^* - 1\}$
    $N \leftarrow N \setminus i$; $M \leftarrow M \setminus \{j^*, j^* - 1\}$;
  else
    while $|N| \geq |S_M|$ **do**
      if $|S_M| > 0$ **then**
        $S \leftarrow j \in S_M$; // create a bag with arbitrary item of S_M
      else
        $S \leftarrow j \in S_L$; // create a bag with arbitrary item of S_L
      $N(S) = \{i : i \in N, v_i(S) \geq 2/3\}$; // $N(S)$ changes with S
      while $|N(S)| = 0$ **do**
        $j \in S_L$; $S \leftarrow S \cup j$; // add arbitrary low value item to the bag
      $i \in N(S)$; $A_i \leftarrow S$; // assign i the bundle S
      $N \leftarrow N \setminus i$; $M \leftarrow M \setminus S$;

Proof. The argument is a simple generalization of Proposition 3. From normalized valuations and Proposition 2, $\mu_i \leq 1$ for all $i \in N$. By definition of the set S_M, for all items $j \in S_M$ there exists an agent $k \in N$ so that $v_{kj} \geq 1/3$. Since the instance is ordered, if $v_{kj} \geq 1/3$, then $v_{kj'} \geq 1/3$ for all $j' \leq j$. Since $|S_M| > |N| > 0$ and $j^* \in S_M$, there exists at least one agent $k \in N$ so that $v_k(S) \geq 2/3\mu_k$, showing part i).

We now show part ii). For any remaining agent i in the reduced instance I', consider the bundles $A = \{A_1, \ldots, A_n\}$ she makes while computing her μ_i in the original instance I. Note that $v_i(A_j) \geq \mu_i$ for all $A_j \in A$. In the reduced instance I', agent i must create one less bundle, but has two fewer items, specifically j^* and $j^* - 1$. We show how to construct a feasible allocation $A' = \{A'_1, \ldots, A'_{n-1}\}$ so that $v_i(A'_j) \geq \mu_i$ for all $A'_j \in A'$. Notice that the condition $|S_M| > |N|$ guarantees that at least one bundle, say A_k, must contain at least two items, say $u, v \in S_M$, by the pigeon hole principle. Wlog we may assume $v_i(u) \leq v_i(v)$. Since we take the two lowest valued items of S_M, $S = \{j^*, j^* - 1\}$, then $v_i(j^*) \leq v_i(u)$ and $v_i(j^* - 1) \leq v_i(v)$. Let A_{j^*} and A_{j^*-1} be the bundles of A containing items j^* and $j^* - 1$ respectively. Suppose agent i swaps item $u \in A_k$ with item $j^* \in A_{j^*}$ and swaps item $v \in A_k$ with item $j^* - 1 \in A_{j^*-1}$ to create A'_k, A'_{j^*}, and A'_{j^*-1}. Finally, i distributes the items of $A'_k \setminus S$ to other bundles arbitrarily to create a new set of bundles $A' = \{A'_1, \ldots, A'_{n-1}\}$. It is clear that A' is a feasible allocation and that $v_i(A'_j) \geq \mu_i$. Therefore, agent i's MMS μ'_i in the reduced instance I' satisfies $\mu'_i \geq \mu_i$. ◀

We now consider the algorithm's second phase, bag filling.

▶ **Lemma 10.** *Let $I = (N, M, V)$ be an ordered problem instance with normalized valuations. Suppose that $|S_H| = 0$ and $0 < |S_M| \leq |N|$. Then, the modified bag filling algorithm ensures all agents receive a bundle worth at least $2/3$ of their maximin share.*

Proof. This argument is a simple generalization of Proposition 5. In modified bag filling, we simply initialize the bag S to an arbitrary item $j \in S_M$. Notice that, this initialization ensures the condition $|S_M| \leq |N|$ holds in each iteration since we always remove one agent and one item of S_M. As valuations are normalized, it is enough to show all agents receive a bundle worth at least $2/3$.

First, note that $|S_L| > 0$, since from normalized valuations and the fact that $v_{ij} < 2/3$ $\forall j \in S_M$, we see that $\forall i \in N$: $v_i(S_L) = v_i(M) - v_i(S_M) \geq |N| - 2/3|S_M| \geq |N|/3 > 0$. We now show that some agent i eventually values the bag $v_i(S) \geq 2/3$. Let $j \in S_M$ be the initial item of the bag. If there exists an agent $i \in N$ such that $v_{ik} < 1/3$ for all $k \in M$, then, clearly there exists some $S' \subset S_L$ so that $v_i(j \cup S') \geq 2/3$. Suppose that no such agent exists. Note that from the definition of S_M, there exists some agent i such that $v_{ij} \geq 1/3$. Given that $v_i(S_L) \geq |N|/3$, we see that $v_i(j \cup S_L) \geq 1/3 + |N|/3 \geq 2/3$ for $|N| \geq 1$. Therefore, there exists $S' \subset S_L$ so that $v_i(j \cup S') \geq 2/3$. This establishes the bag is eventually assigned to an agent who values it at least $2/3$.

Let k be the agent assigned the bag S. Now, we show that $v_i(S) \leq 1$ for all other agents $i \in N \setminus k$. Before adding the final item of the bag $j' \in S$, $v_i(S \setminus j') < 2/3$ for all $i \in N$. The final item added to the bag comes from S_L so $v_i(j') < 1/3$ for all $i \in N$. Therefore, $v_i(S) < 1$ for all $i \in N$. This means that for each agent i, $v_i(M) \geq |N|$ and $v_i(S_L) \geq |N|/3$ are invariants of the algorithm. Then, it is easy to see all agents receive a bundle worth $2/3$. Finally, when $|S_M| = 0$, all agents receive a bundle worth at least $2/3$ by Proposition 5. ◀

From Lemmas 8, 9, and 10, we get the following theorem.

▶ **Theorem 11.** *Algorithm 3 provides a 2/3 approximate MMS allocation.*

▶ Remark. Lemmas 9 and 10 are really just simple generalizations of Propositions 3 and 5 (respectively) designed to manage medium valued items S_M. In this sense, our algorithm is natural generalization of the simple $1/2$ MMS algorithm of Section 2.2 which improves performance guarantees to $2/3$ MMS.

4 Discussion

In this paper we investigate fair division of indivisible items using maximin share as our measure of fairness of an allocation. We propose a simple greedy approximation algorithm to obtain a $2/3$ MMS allocation. Further, we show that our algorithm can be seen as a natural extension of the $1/2$ MMS algorithm discussed in Section 2.2. This allows for a far simpler, and more intuitive analysis as compared to other existing $2/3$ MMS approximations.

Our approach does not seem to generalize to provide better performance guarantees. Consider designing an algorithm to give a $3/4$ MMS allocation. Suppose we naively create three clusters of items: high $v_{ij} \geq 3/4$ for at least one agent i, medium $v_{ij} \geq 1/4$ for at least one agent but $v_{ij} < 3/4$ for all agents, and low $v_{ij} < 1/4$ for all agents. Similar to the $2/3$ case, we allocate high valued items through maximum matching, and if all items are low valued, then bag filling suffices to distribute all remaining items. Notice that, we must assign two or three medium valued items to ensure an agent receives at bundle worth at least $3/4$. If $|S_M| > 2|N|$, then we can guarantee each agent must create at least one bundle containing three items from S_M when computing their MMS, and therefore, may justify

greedily assigning a bundle containing the three lowest valued items of S_M to any agent who values it at least 3/4. When $2|N| \geq |S_M| > |N|$, the situation is less clear. We know each agent creates at least one bundle containing two items from S_M when computing their MMS, but we can't guarantee that some agent will value a bundle containing only the two lowest valued items of S_M at least 3/4. Further, we can't guarantee that we may start bag filling where we initialize the bag to the two least valuable items of S_M since some agent might value some set of two 'better' (more valuable) items of S_M more than 1. If we initialize the bag to only the lowest valued item of S_M, then we might 'run' out of low valued items, leaving only medium valued items and no way to ensure each remaining agent receives at least 3/4.

Attempting a finer partitioning of S_M significantly complicates analysis as it creates numerous special cases based on the number of items within each sub-cluster of medium valued items. Further, it is not clear that a simple allocation decision exists for all possible special cases. For these reasons, it seems the approach presented in this paper is only capable of producing a 2/3 MMS allocation. However, as our algorithm is closely related to the simple 1/2 MMS approximation, we find our approach more intuitive than other existing 2/3 MMS algorithms.

References

1. Georgios Amanatidis, Evangelos Markakis, Afshin Nikzad, and Amin Saberi. Approximation algorithms for computing maximin share allocations. *ACM Transactions on Algorithms (TALG)*, 13(4):52, 2017.
2. Siddharth Barman and Sanath Kumar Krishna Murthy. Approximation algorithms for maximin fair division. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 647–664. ACM, 2017.
3. Sylvain Bouveret and Michel Lemaître. Characterizing conflicts in fair division of indivisible goods using a scale of criteria. *Autonomous Agents and Multi-Agent Systems*, 30(2):259–290, 2016.
4. Sylvain Bouveret and Michel Lemaître. Efficiency and sequenceability in fair division of indivisible goods with additive preferences. *arXiv preprint arXiv:1604.01734*, 2016.
5. Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.
6. Mohammad Ghodsi, MohammadTaghi HajiAghayi, Masoud Seddighin, Saeed Seddighin, and Hadi Yami. Fair allocation of indivisible goods: Improvement and generalization. In *EC*, 2018.
7. David Kurokawa, Ariel D Procaccia, and Junxing Wang. When can the maximin share guarantee be guaranteed? In *AAAI*, volume 16, pages 523–529, 2016.
8. David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair Enough: Guaranteeing Approximate Maximin Shares. *J. ACM*, 65(2):8:1–8:27, 2018.
9. Ariel D Procaccia and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 675–692. ACM, 2014.
10. Hugo Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.
11. Gerhard J Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20(4):149–154, 1997.